In the Name of God

University of Tehran

Department of Electrical and Computer Engineering

Machine Learning

Final Project (phase 2)

Parsa Darban – 810100141

July 2024

Contents:

Introduction

In this project, EEG signals are analyzed using advanced preprocessing and feature extraction methods. The noninvasive EEG signal, recorded by placing electrodes on the scalp, provides valuable insights into brain activity. Our study covers the entire EEG signal processing pipeline, including data preprocessing, feature extraction, classification, and application. Specifically, we perform the following steps:

- **Data Preprocessing:**
  - We preprocess the EEG data obtained from files BCICIV_calib_ds1a and BCICIV_calib_ds1d.
  - First, we apply denoising techniques to enhance the quality of the raw data.
- **Frequency Filtering:**
  - Next, we apply a Butterworth filter to isolate frequencies between 8 Hz and 31 Hz.
  - This step helps focus on relevant brain activity.
- **Spatial Filtering:**
  - We employ spatial filters, including the Common Average Reference (CAR) filter and Laplacian filter.
  - These filters enhance the spatial representation of EEG signals.
- **Feature Extraction:**
  - Feature extraction is performed using the Common Spatial Patterns (CSP) method and wavelength analysis.
  - These features capture relevant information for subsequent classification.
- **Classification:**
  - We classify the preprocessed EEG data using machine learning algorithms:
    - K-Nearest Neighbors (KNN)
    - Multi-Layer Perceptron (MLP)
    - Logistic Regression
  - Evaluation metrics such as confusion matrices, ROC curves, and class-specific error rates are used to compare the performance of these classifiers.

Preprocessing for BCICIV_calib_ds1a

```
sample_rate = m['nfo']['fs'][0][0][0][0]
EEG = m['cnt'].T
nchannels, nsamples = EEG.shape

channel_names = [s[0] for s in m['nfo']['clab'][0][0][0]]
event_onsets = m['mrk'][0][0][0]
event_codes = m['mrk'][0][0][1]
labels = np.zeros((1, nsamples), int)
labels[0, event_onsets] = event_codes

cl_lab = [s[0] for s in m['nfo']['classes'][0][0][0]]
cl1 = cl_lab[0]
cl2 = cl_lab[1]
nclasses = len(cl_lab)
nevents = len(event_onsets)
```

First, the data is loaded. Using the scipy.io library, the BCICIV_calib_ds1a.mat dataset is loaded. Relevant information from the variables is extracted and organized into matrices. The EEG signals from each channel are stored in a matrix of size (59,190594). Additionally, the temporal data points are mapped to different positions using the event_onset matrix (200), and labels are associated with each event using event_codes (200).

Next, the EEG data for each brain channel is segmented based on the 200 available events. Given that BCI has 8-second intervals, the EEG signals are divided into segments starting from the event onsets. Note that wherever time is referred to using fs=100, it is multiplied by 100. As a result, 200 matrices of size (59,800) are obtained. This segmentation is necessary because during rest periods, the EEG signals lack specific features, and the signals during these times exhibit similar patterns.

The information bellows define the datasets:

```
Shape of EEG: (59, 190594)
Sample rate: 100
Number of channels: 59
Channel names: ['AF3', 'AF4', 'F5', 'F3', 'F1', 'Fz', 'F2', 'F4', 'F6', 'FC5', 'FC3',
'FC1', 'FCz', 'FC2', 'FC4', 'FC6', 'CFC7', 'CFC5', 'CFC3', 'CFC1', 'CFC2', 'CFC4',
'CFC6', 'CFC8', 'T7', 'C5', 'C3', 'C1', 'Cz', 'C2', 'C4', 'C6', 'T8', 'CCP7', 'CCP5',
'CCP3', 'CCP1', 'CCP2', 'CCP4', 'CCP6', 'CCP8', 'CP5', 'CP3', 'CP1', 'CPz', 'CP2',
'CP4', 'CP6', 'P5', 'P3', 'P1', 'Pz', 'P2', 'P4', 'P6', 'PO1', 'PO2', 'O1', 'O2']
Number of events: 1
Event codes: [-1  1]
Class labels: ['left', 'foot']
Number of classes: 2
```

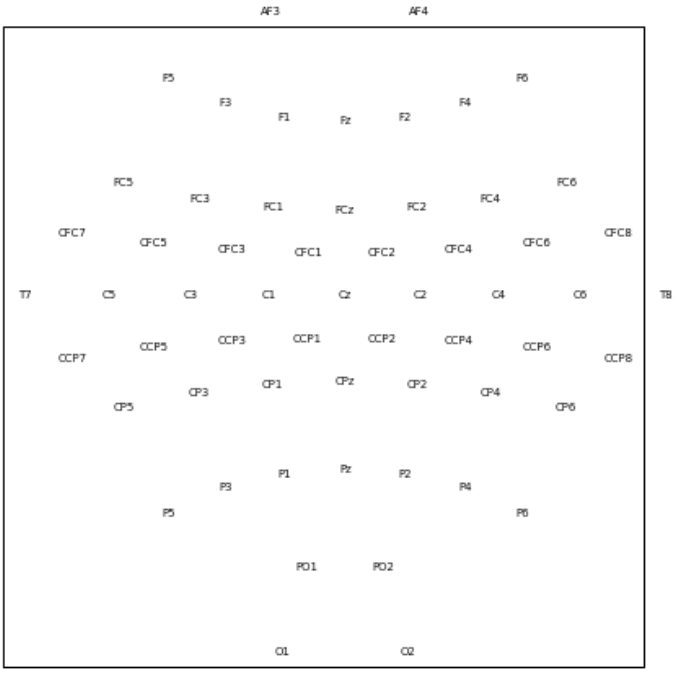


Figure 1 Placement of 59 EEG signal recording electrodes

As we observe from the output, we have 59 channels. Based on the image, we can deduce that the Cz channel exhibits the strongest response for foot movements, while the C3 channel is most responsive to right-hand movements. Similarly, the C4 channel demonstrates the best response for left-hand.

We separate classes and creating two matrices based on binary labels. Then, we filter a 3D matrix (100, 59, 800) in the frequency domain. Given that important signals for hand and foot movements typically fall within the frequency range of 8Hz to 30Hz, you plan to apply a third-order Butterworth filter (an IIR filter) with the following shape:

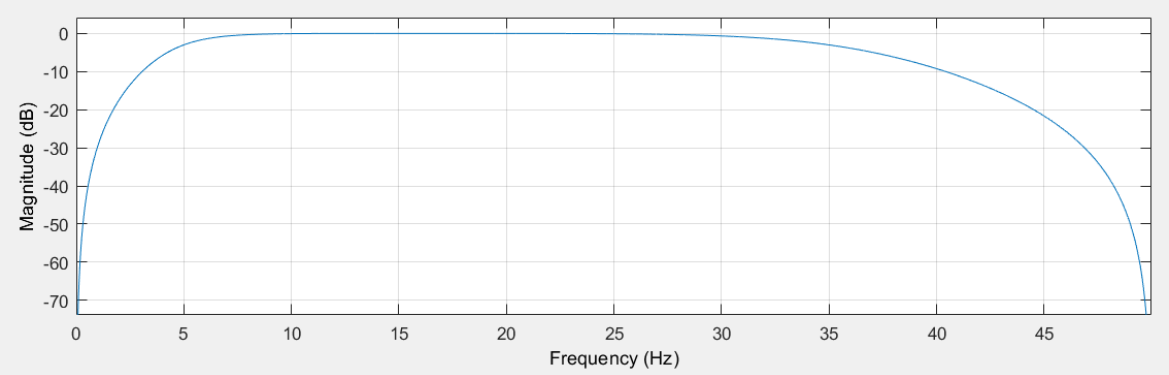


Figure 2 IIR Butterworth filter

The filter is not ideal, and it does not make our gain equal to zero in the frequency range other than 8 to 30 Hz.

Now we select a channel and plot it before and after filtering.

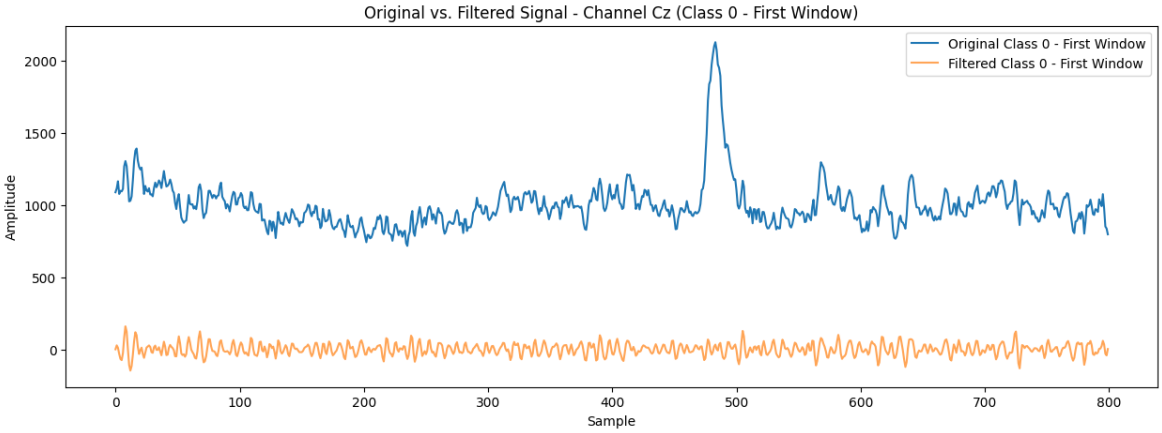


Figure 3 The central signal before and after the application of the bandpass filter per time unit

Note that this plot is only to verify that our filter is working correctly. By using an MSE value of 6764, we observe that it is work correctly. Additionally, by taking the Fourier transform of the signal, we can see that the signal exists at the mentioned frequency.

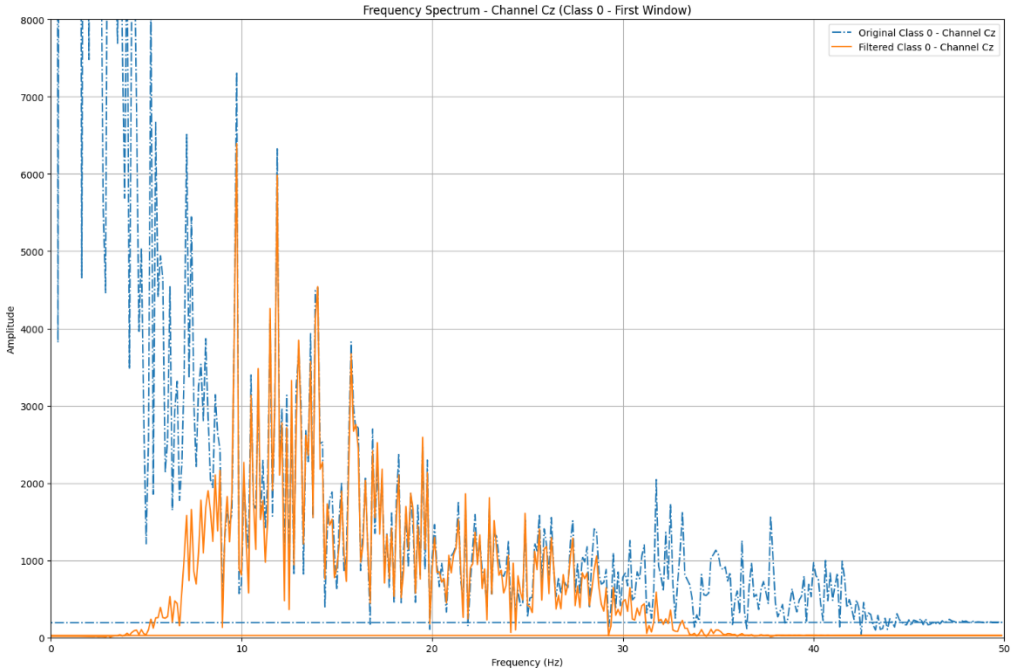


Figure 4 The central signal before and after the application of the bandpass filter per frequency unit

Next, we apply the Laplacian filter. As mentioned in the initial phase, the Laplacian filter is a spatial filter used to increase the signal-to-noise ratio (SNR). It receives EEG signals from various electrode locations using a Laplacian matrix and averages them with respect to brain activity. This process eliminates common noise. The method is based on the potential differences of each neuron.

There are various types of spatial filters. Another one is the Common Average Reference (CAR) filter. This filter subtracts the average of each channel's values to reduce noise.

However, the implementation of this filter does not provide us with sufficiently accurate results. For example, in logistic regression, the accuracy for the CAR filter is 78%, whereas for the Laplacian filter, it is 84%. Based on the literature and the Phase 1 report, the Laplacian filter provides significant accuracy in the spatial location of each channel and can enhance signal strength while reducing noise (similar to an operational amplifier).

We can also use Principal Component Analysis (PCA) and Independent Component Analysis (ICA) for further analysis.

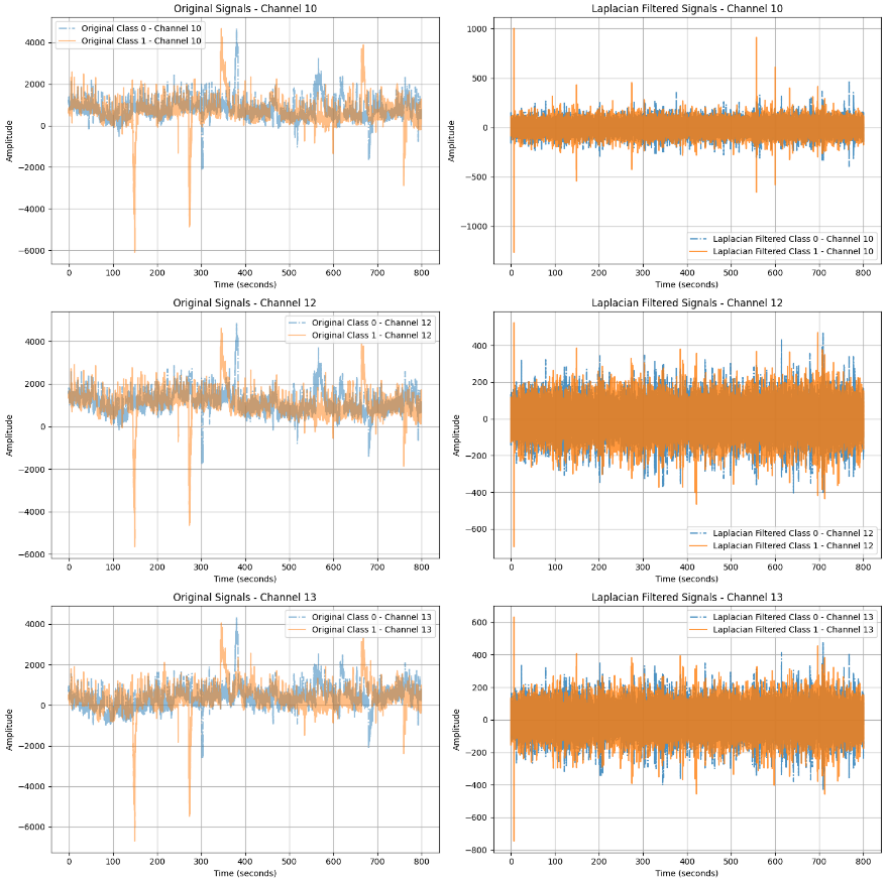


Figure 5 compare original and filtered signals for 3 important channels for Laplacian filter

As we expected, the EEG signal for the channels has become cleaner and more distinguishable after filtering. The sharp features of the signal have also become more pronounced. Given this, we can now proceed with feature extraction.

T-SNE plot is a powerful tool for visualizing high-dimensional data, highlighting clusters, and understanding the local relationships between data points. However, it should be used with an understanding of its limitations, particularly regarding the preservation of global data structure.

Now we plot 2D with t_SNE:

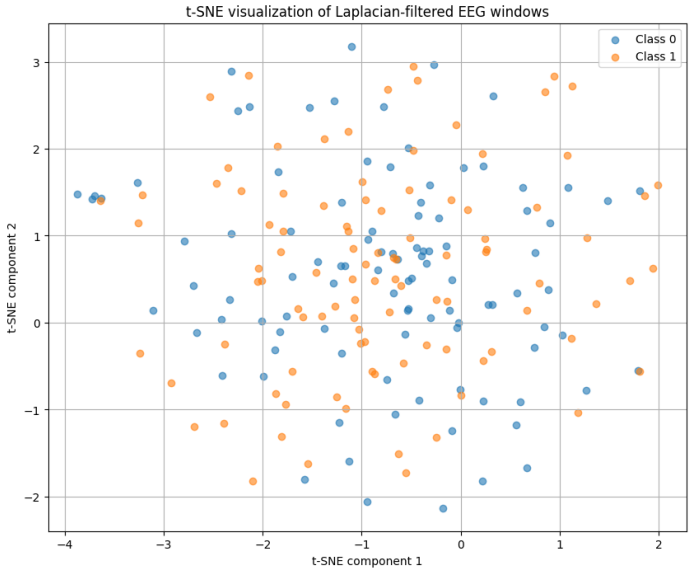


Figure 6 t-SNE plot for Laplacian filter

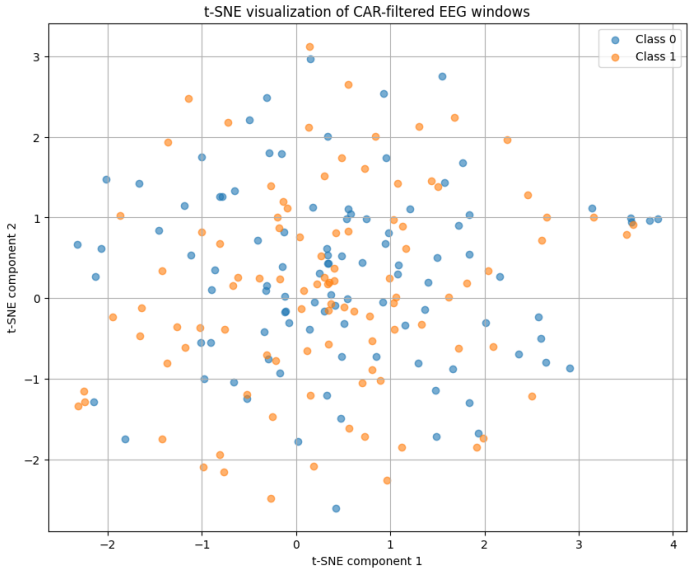


Figure 7 t-SNE plot for CAR filter

This plot indicates that there is a high data scatter, reflecting the differences between signals present in each window. Moreover, there is considerable overlap between the two classes, making it challenging to accurately classify the classes.

To address these challenges, we need to perform feature extraction, which we'll focus on. We can use various methods such as CSP and wavelet transform.

Additionally, we can utilize different classifiers that can understand the complexities of our data and classify with high accuracy.

Feature Extraction for BCICIV_calib_ds1a

We will use the CSP method for feature extraction. This method has been elaborately explained in the first phase. Here, since we have spatial and spatial differences between the two classes, CSP finds filters that maximize the variance differences between the two classes. The variance of EEG signals in the new projected space indicates the level of brain activity that differs between the two classes. These variance features highlight important differences between the two classes, which helps the classification model to separate the data more accurately. Given this, by grouping the features and calculating their maximum variance, we get the best possible features.

CSP is implemented in the MNE (Magnetoencephalography and Electroencephalography) library, which is a comprehensive Python package for analyzing electrophysiological data. We use this package to apply the CSP algorithm to our data.

```
csp = CSP(n_components=num_components, reg=None, log=True,
norm_trace=False)
```

Which n_components specifies the number of spatial filters (or components) to use. Reg is used for regularization to prevent overfitting, regularization can help stabilize the covariance matrix estimation, especially when the number of samples is low compared to the number of channels. In this case that we used; no regularization is applied. Log when sets to True, indicates that a logarithmic transformation will be applied to the features. This can help in normalizing the feature values and improving the performance of the classifier that uses features. Norm_trace when set to False, the traces (sum of diagonal elements) of the covariance matrices are not normalized. If set to True, the covariance matrices are normalized to have a trace equal to one. Normalizing the trace can help in comparing covariance matrices with different scales.

Feature extraction using the wavelet method is a powerful technique often employed in signal processing, particularly for analyzing non-stationary signals like EEG data. Wavelets provide a way to decompose a signal into components at various scales, capturing both frequency and time information.

```
def extract_wavelet_features(data):
    features = []
    for trial in data:
        trial_features = []
        for channel in trial:
            coeffs = pywt.wavedec(channel, 'db4', level=6)
            trial_features.extend([np.sum(np.abs(c) ** 1) for c in coeffs])
            trial_features.extend([np.mean(c) for c in coeffs])
```

```
            trial_features.extend([np.std(c) for c in coeffs])
        features.append(trial_features)
    return np.array(features)
```

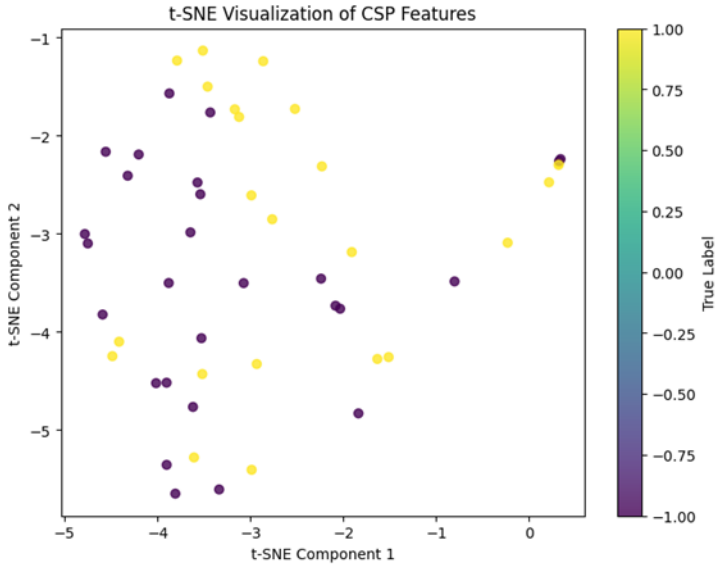


Figure 8 t-SNE visualization of CSP features

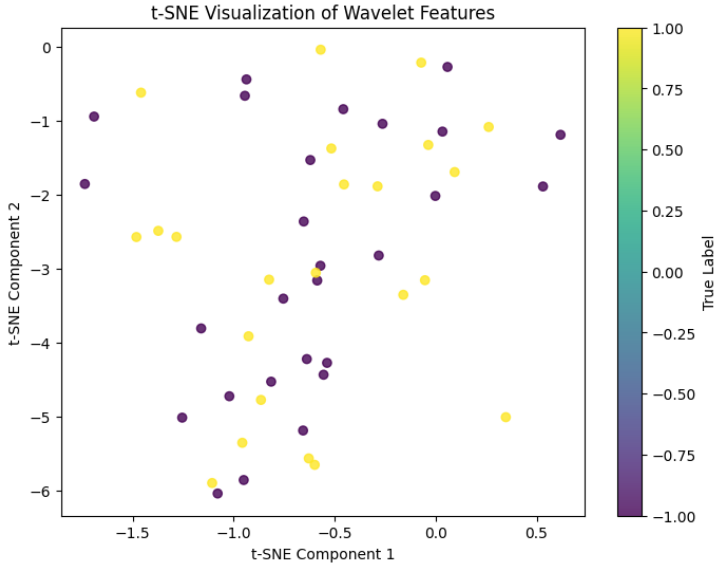


Figure 9 t-SNE visualization of Wavelet features

Due to the overlap and lack of concentration of the labels, it appears that classification using the wavelet feature extraction method does not achieve the required accuracy.

Classification for BCICIV_calib_ds1a

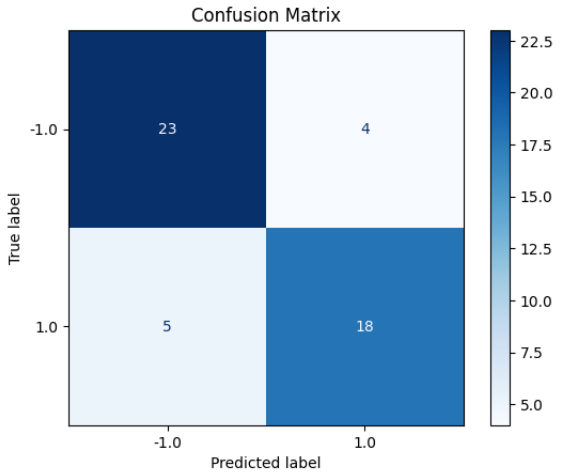
Using three classifiers, Logistic Regression, MLP, and KNN, we will calculate the accuracy, derive the confusion matrix and ROC curve, and also determine the error for each class. We apply all these three methods using two different filter (CAR filter and Laplacian filter).

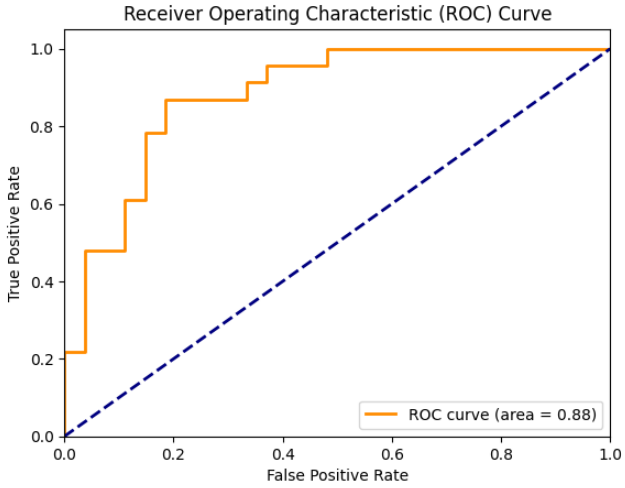
Notice that the choice of num_components affect the dimensionality of the feature space. If you choose too few components, you might lose important discriminative information. If you choose too many, you might include noise and irrelevant information, potentially leading to overfitting. The default value is often the number of EEG channels used, but optimal value depends on the specific dataset and classification task. So, we need to find a trade-off to achieve the best accuracy.

Logistic Regression (Laplacian Filter and CSP):

```
Accuracy on test set: 0.82
```

```
Class -1 error: 0.15
Class 1 error: 0.22
```

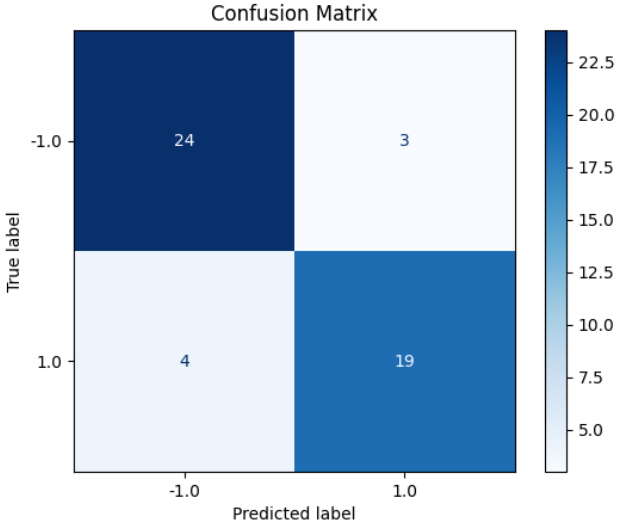


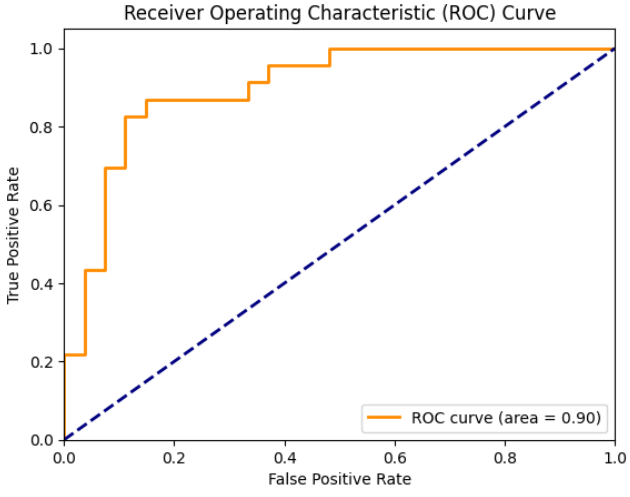


Logistic Regression (CAR Filter and CSP):

```
Accuracy on test set: 0.86
```

```
Class -1 error: 0.11
error: 0.17
```



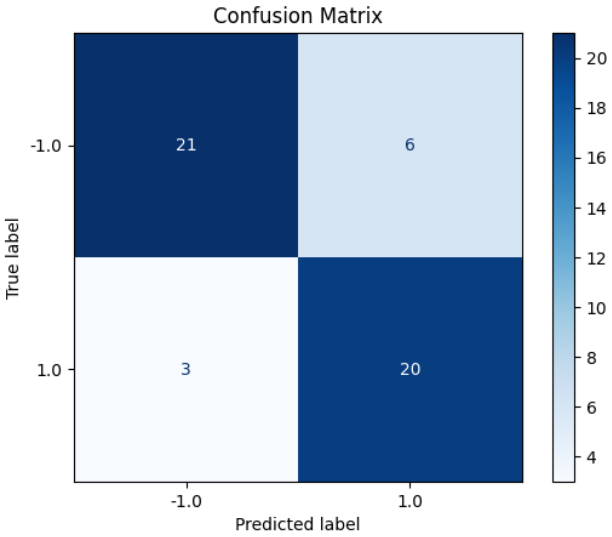


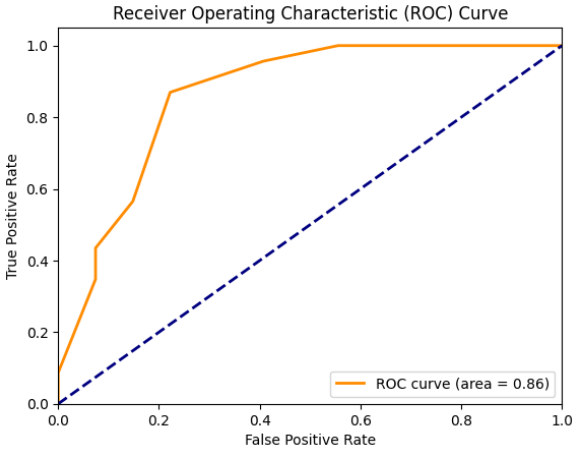
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR).

The ROC curve for the Logistic Regression with the Laplacian filter shows an AUC of 0.88, indicating good discriminative ability. However, the stepped appearance of the ROC curve, with relatively few points of variation, suggests that the model's probability outputs are not highly granular and may be clustered at specific threshold levels. This can be due to the relatively small dataset size or limited variation in the predicted probabilities, causing less smooth and more abrupt changes in the true positive and false positive rates as thresholds vary. Despite this, the high AUC value indicates the model is still effective at distinguishing between the two classes.

KNN (Laplacian Filter and CSP):

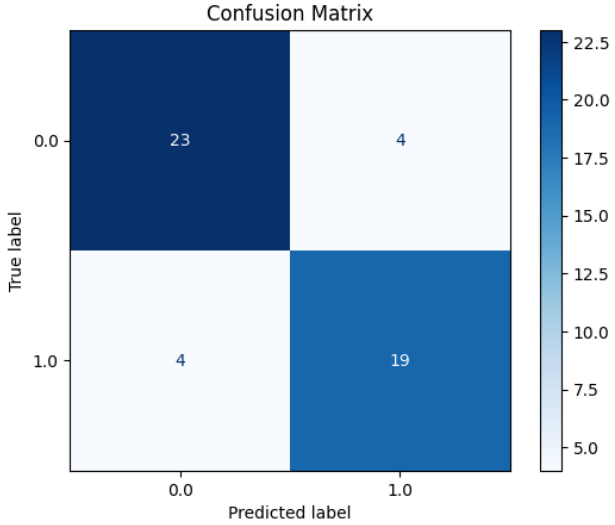
```
Accuracy on test set: 0.82
Class -1 error: 0.22
Class 1 error: 0.13
```

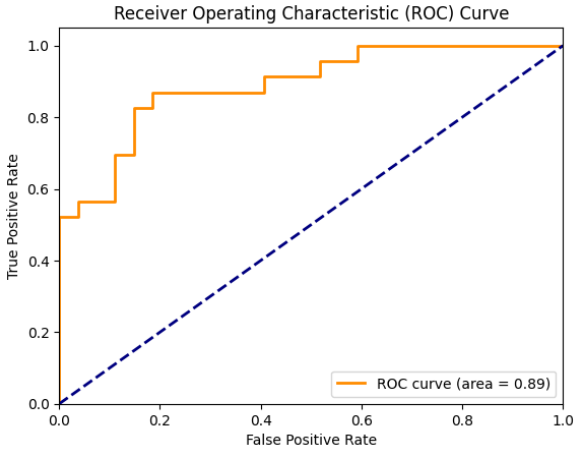




KNN (CAR Filter and CSP):

```
Accuracy on test set: 0.84
Class 0 error: 0.15
Class 1 error: 0.17
```

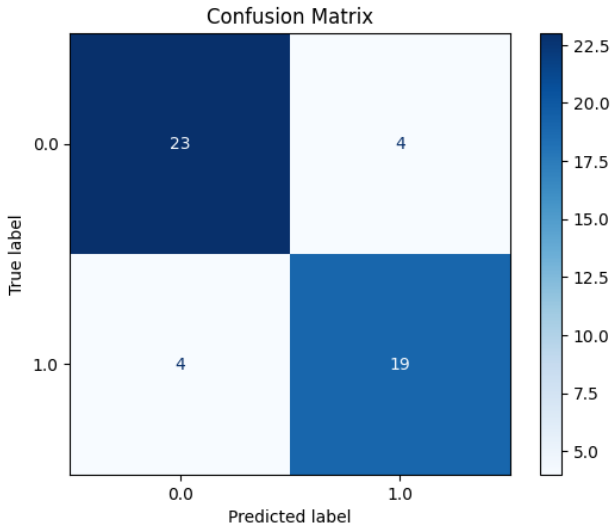


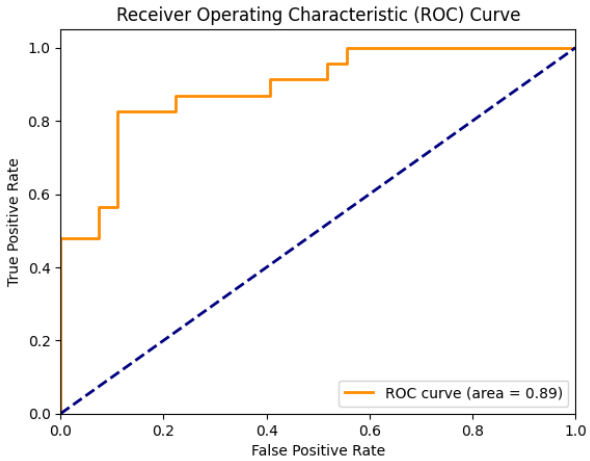


MLP (Laplacian Filter and CSP):

```
Accuracy on test set: 0.84
```

```
Class 0 error: 0.15
Class 1 error: 0.17
```

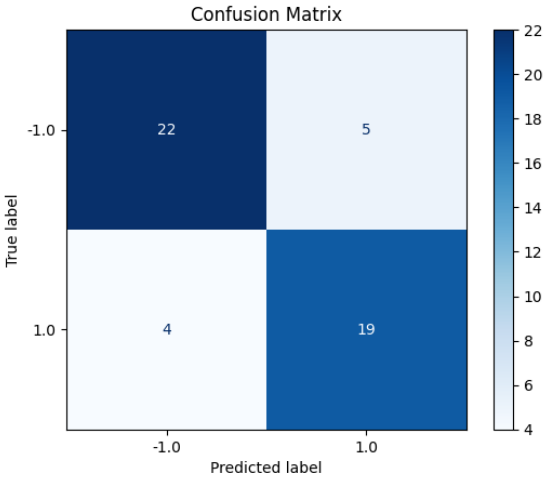


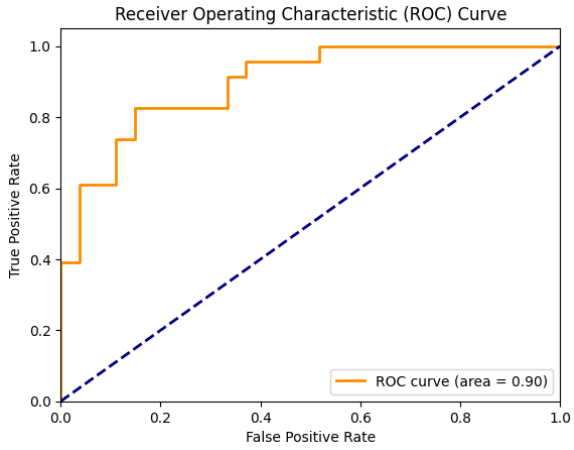


MLP (CAR Filter and CSP):

```
Accuracy on test set: 0.82
```

```
Class -1 error: 0.19
Class 1 error: 0.17
```





The second method for feature extraction is using Wavelet Transform. This method is based on analyzing signals in terms of their frequency and time information, decomposing them, and filtering them. Similar to CSP, it is also based on spatial variations. For feature extraction, it utilizes variance, mean, and even energy of each component.

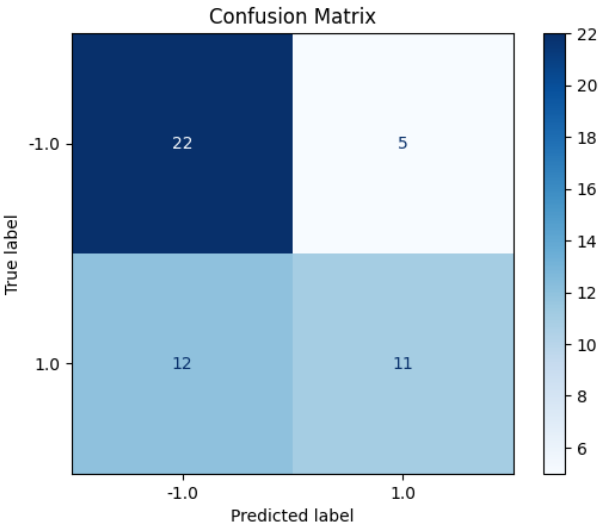
After applying this transformation to the signal, we proceed with classification.

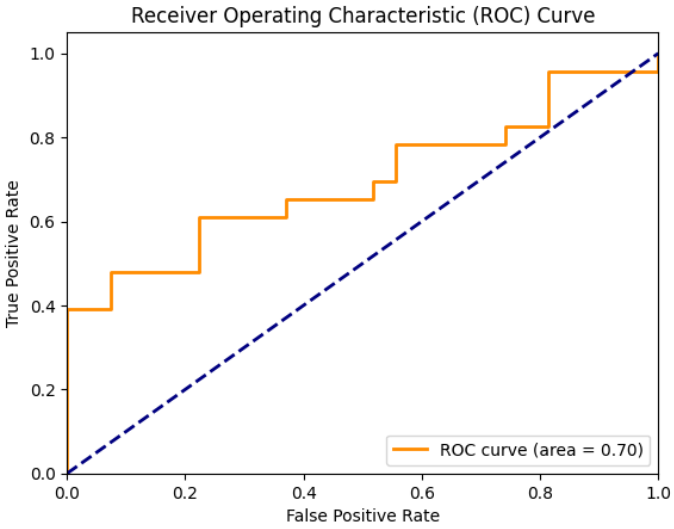
The result of each classification is as follows:

Logistic Regression (Laplacian Filter and Wavelet):

```
Accuracy on test set: 0.66
```

```
Class -1 error: 0.19
Class 1 error: 0.52
```

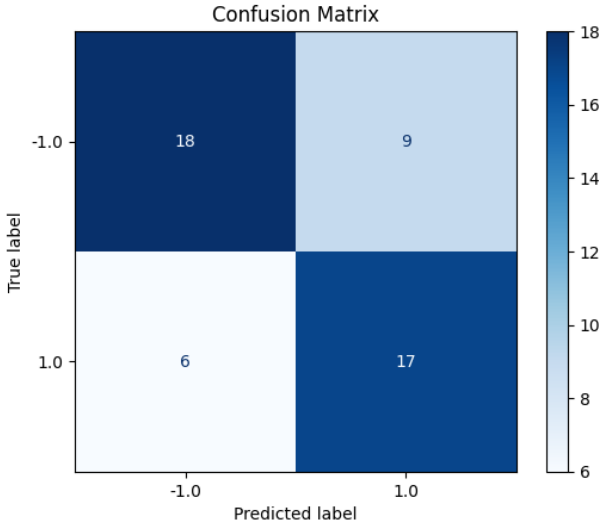


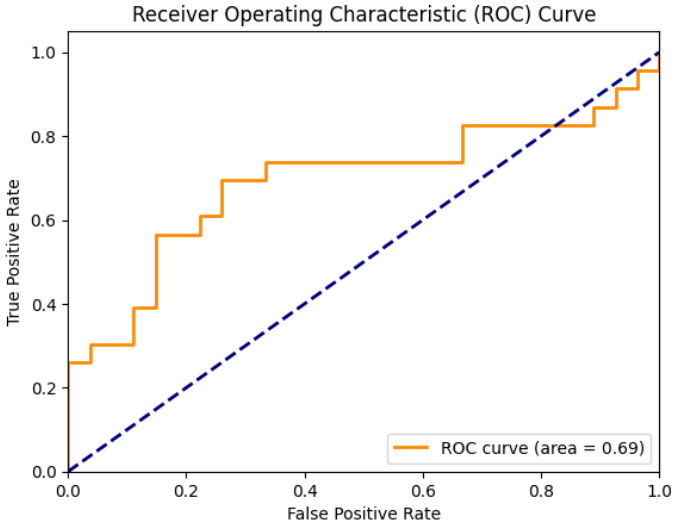


Logistic Regression (CAR Filter and Wavelet):

```
Accuracy on test set: 0.70
```

```
Class -1 error: 0.33
Class 1 error: 0.26
```

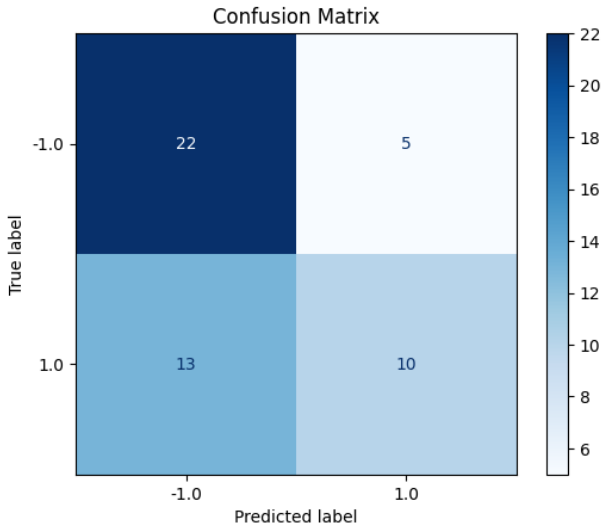


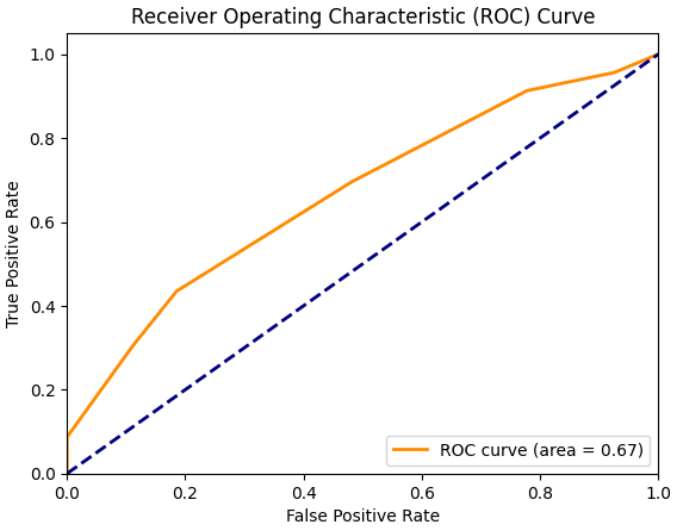


KNN (Laplacian Filter and Wavelet):

```
Accuracy on test set: 0.64
```

```
Class -1 error: 0.19
Class 1 error: 0.57
```

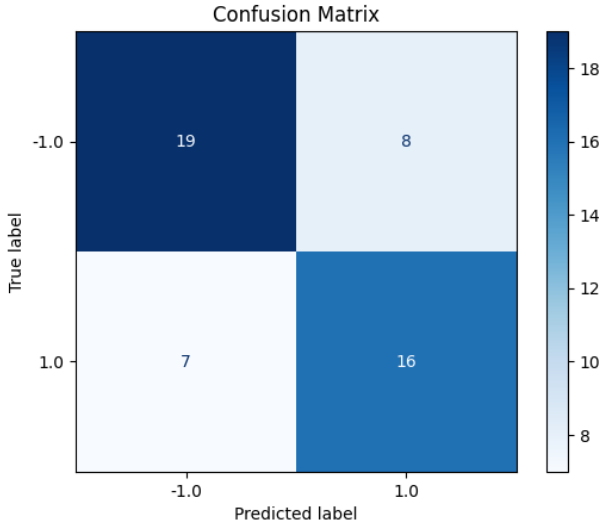


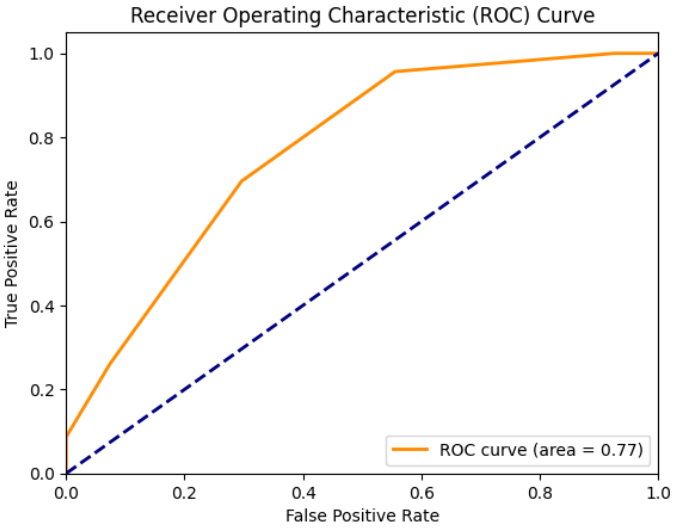


KNN (CAR Filter and Wavelet):

```
Accuracy on test set: 0.70
```

```
Class -1 error: 0.30
Class 1 error: 0.30
```

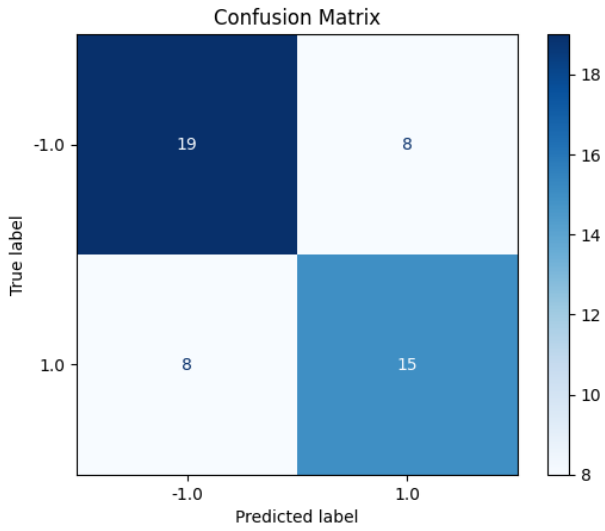


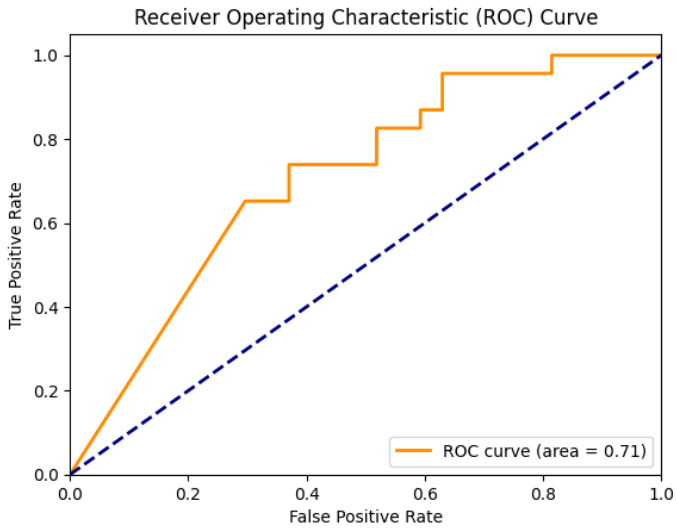


MLP (Laplacian Filter and Wavelet):

```
Accuracy on test set: 0.68
```

```
Class -1 error: 0.30
Class 1 error: 0.35
```

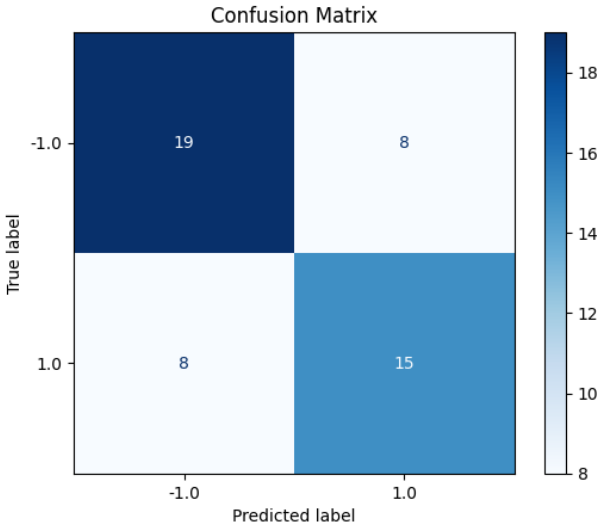


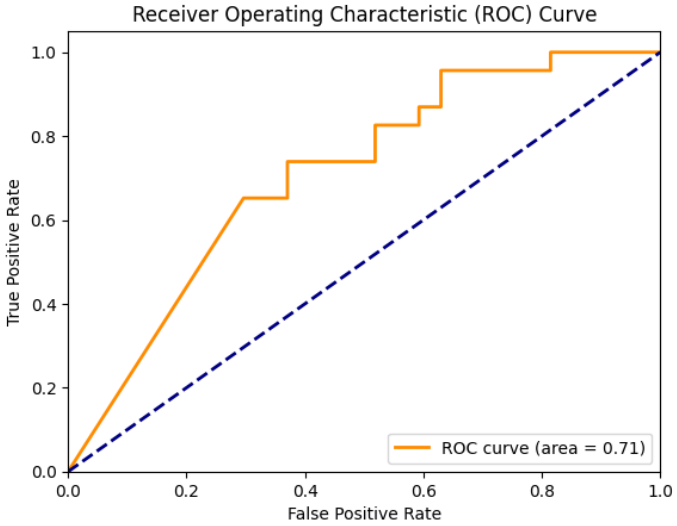


MLP (CAR Filter and Wavelet):

```
Accuracy on test set: 0.68
```

```
Class -1 error: 0.30
Class 1 error: 0.35
```





Accuracy on test set table:

Special filter/Classification method	Logistic regression	KNN	MLP
Laplacian filter & CSP	82%	82%	84%
CAR filter & CSP	86%	84%	82%
Laplacian filter & Wavelet	66%	64%	68%
CAR filter & Wavelet	70%	70%	68%

As we expected, the CSP method has higher accuracy.

Analysis:

Overall Best Performance: The highest accuracy achieved was 86% using Logistic Regression with the CAR filter.

Filter Effectiveness:

- The CAR filter was found to improve the performance of Logistic Regression and KNN classifiers.
- The Laplacian filter appeared to work better with the MLP classifier.

Classifier Selection:

- For the highest accuracy, Logistic Regression with the CAR filter was identified as the best choice.
- When using the Laplacian filter, the MLP provided the highest accuracy.

Feature Extraction: The CSP method was observed to provide better accuracy

Clustering

Clustering is an unsupervised learning technique that involves grouping data points into clusters such that points within the same cluster are more similar to each other than to points in other clusters. This method is widely used in various fields such as data mining, pattern recognition, image analysis, and bioinformatics to discover underlying patterns in data. The objective of clustering is to identify the intrinsic grouping in a set of unlabeled data. Key clustering methods include K-Means, Gaussian Mixture Models (GMM), and Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

Gaussian Mixture Models (GMM):

Gaussian Mixture Models are a probabilistic model for representing normally distributed subpopulations within an overall population. GMMs assume that the data is generated from a mixture of several Gaussian distributions with unknown parameters. The key parameters in GMM are:

```
gmm = GaussianMixture(n_components=n_clusters, random_state=42)
```

- n_components: Number of mixture components (clusters).
- means_: Mean of each Gaussian component.
- covariances_: Covariance of each Gaussian component.
- weights_: Weight of each Gaussian component.

The Expectation-Maximization (EM) algorithm is typically used for parameter estimation in GMM. The EM algorithm iteratively improves the parameter estimates to maximize the likelihood of the data.

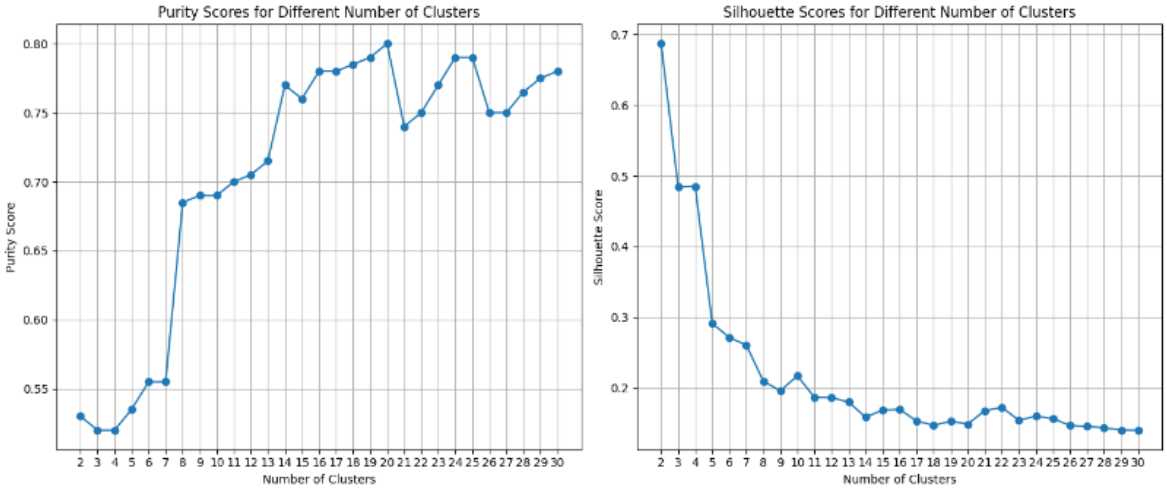


Figure 10 Purity scores and silhouette scores for different number of clusters

As we can see in the above image, we have the best purity score by having 20 clusters.

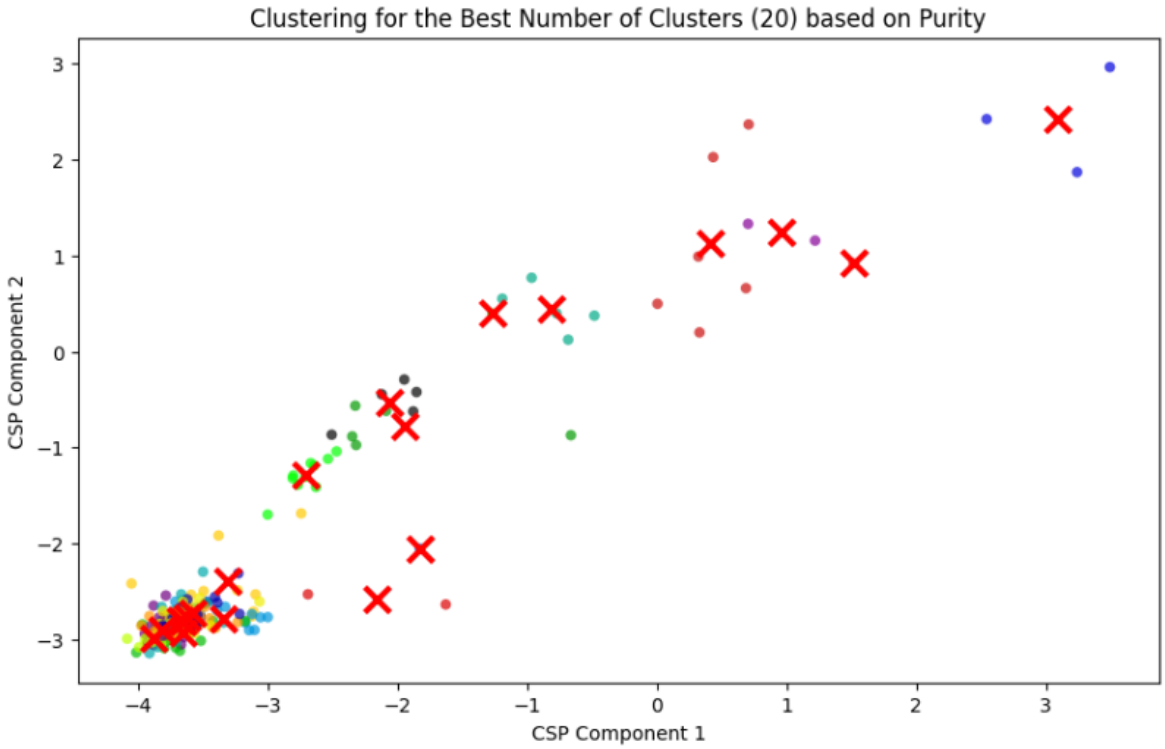


Figure 11 graph of clusters

K-Means:

K-Means is a popular clustering algorithm that partitions data into K distinct clusters based on their features. It aims to minimize the within-cluster sum of squares (inertia). The main steps are:

- Initialize ‘k’ cluster centroids randomly.
- Assign each data point to the nearest centroid.
- Recompute the centroids as the mean of the points assigned to each cluster.
- Repeat steps 2 and 3 until convergence.

K-Means is simple and computationally efficient, but it requires specifying the number of clusters (K) in advance and may not perform well on clusters with varying shapes and sizes.

```
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
```

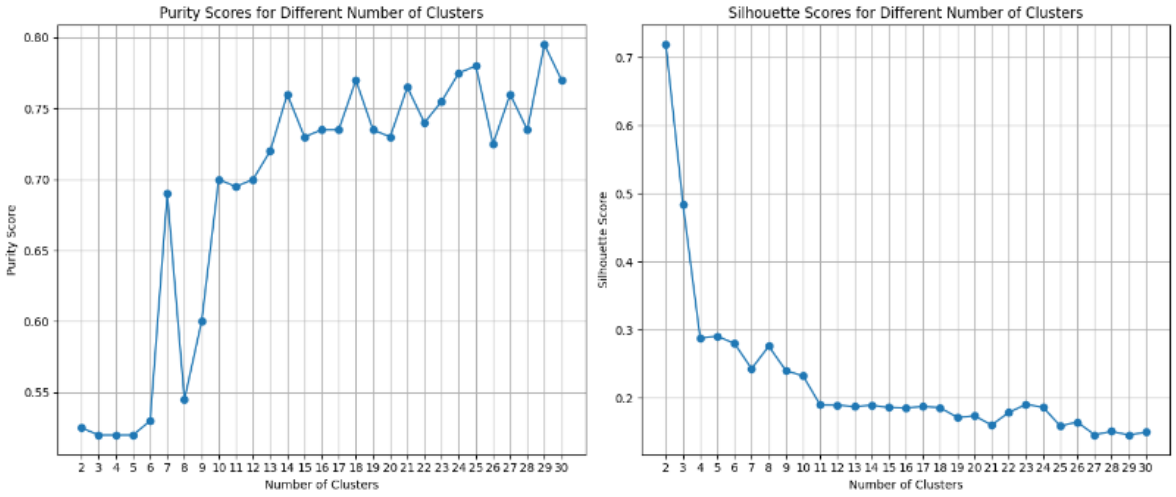


Figure 12 Purity scores and silhouette scores for different number of clusters

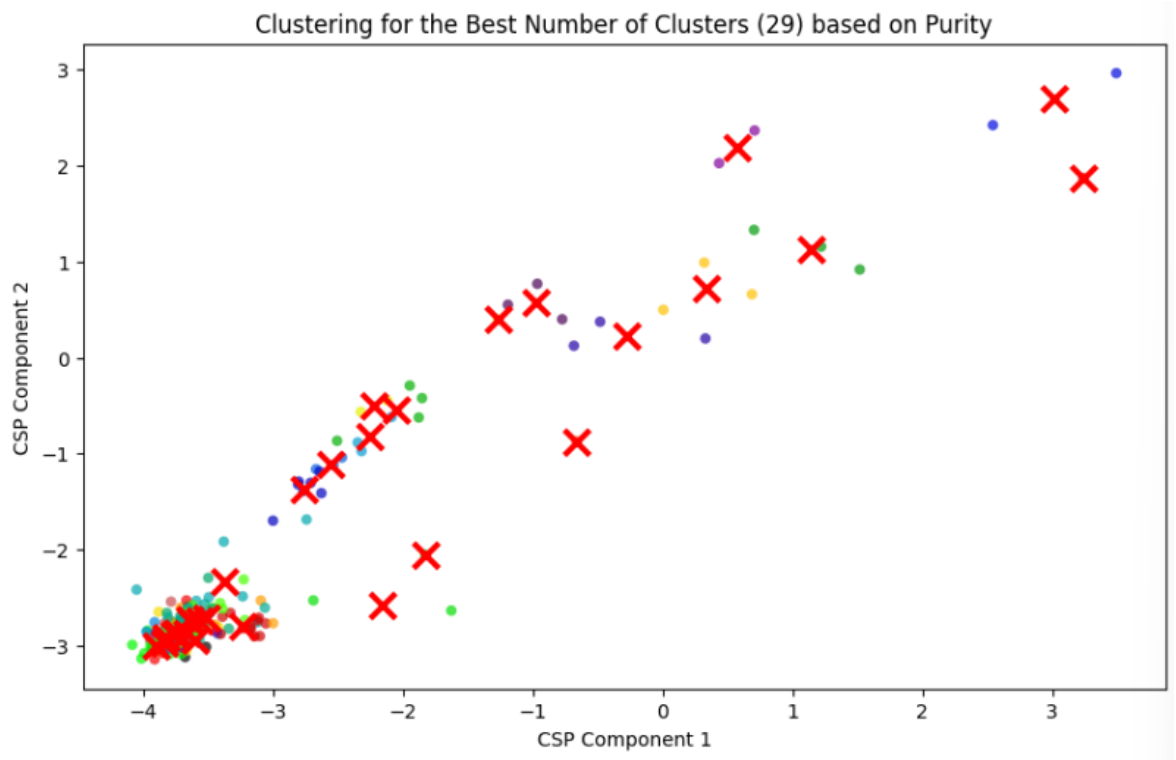


Figure 13 graph of clusters

GMM might be better suited for complex datasets with varying cluster shapes and densities due to probabilistic nature and ability to model covariance. K-Means is efficient and effective for well-separated spherical clusters.

Preprocessing for BCICIV_calib_ds1d

Now, we proceed to analyze the BCICIV_calib_ds1d data. First, we load the data as before and place the important data and signals in a matrix. The names are the same as before, and EEG, event_onset, and event_codes are the same matrices, with only the values inside them changed. Now, for similar reasons, we will window the signals to label the signals that contain useful data from the onset of image display up to 7.5 seconds after. Then, we separate the classes. Here we have a matrix (100,59,750) for each separated class.

We filter them using the shown Butterworth filter and have the following result:

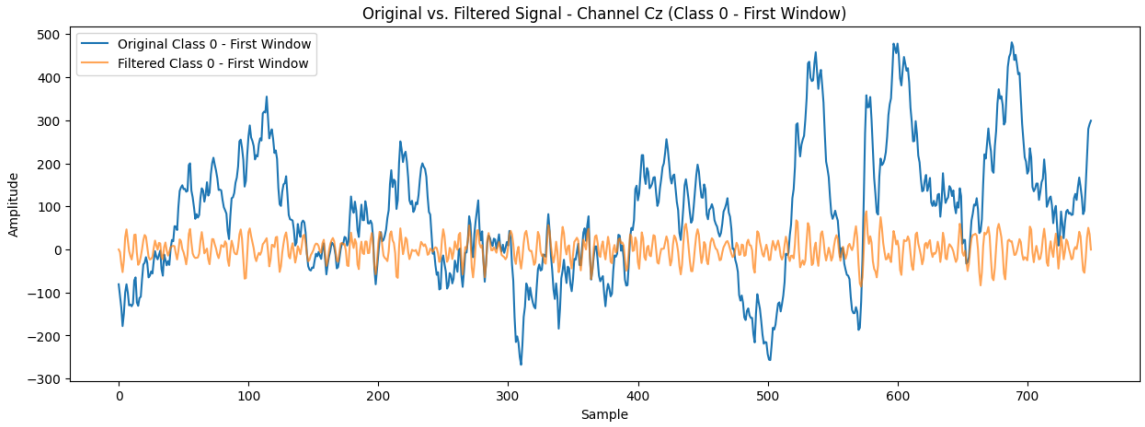


Figure 14 The central signal before and after the application of the bandpass filter per time unit

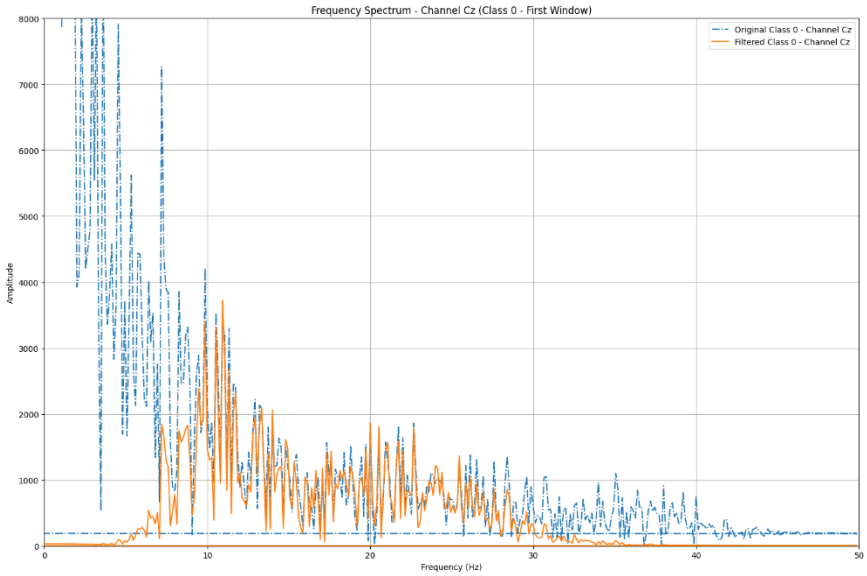


Figure 15 The central signal before and after the application of the bandpass filter per frequency unit

Given the high-frequency range, the filter has been correctly applied.

Now, using the spatial filters we had in the previous section, we filter them. We use the CSP method, and the result after two filters and before that is as follows:

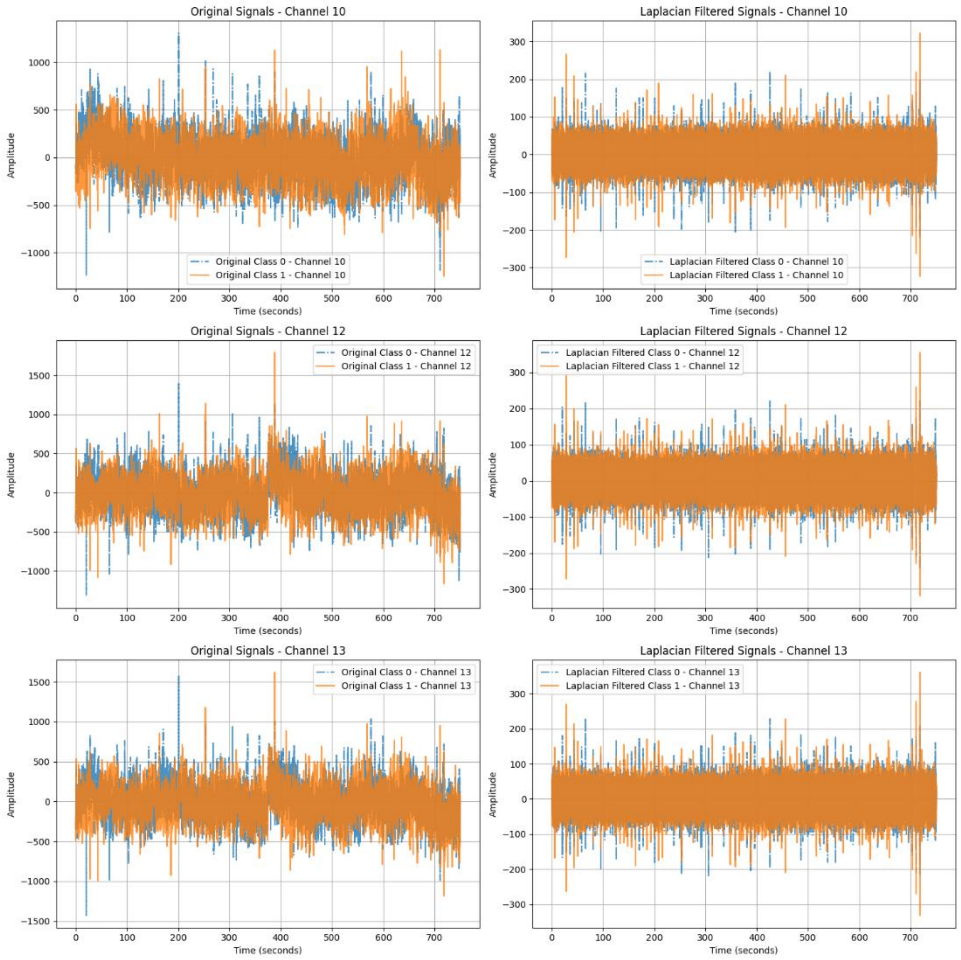


Figure 16 compare original and filtered signals for 3 important channels for Laplacian Filter

As before, given the performance of the Laplacian filter explained in the previous section, the amplitude peaks have been reduced, and after filtering, we have a cleaner signal.

We did one thing in this section. That is, we first applied spatial filters and then frequency filters, but the classification result with a maximum accuracy of 63% was not satisfactory for us. Therefore, like before, we will first apply bandpass and then Laplacian.

We see t-SNE plots for two types of filters.

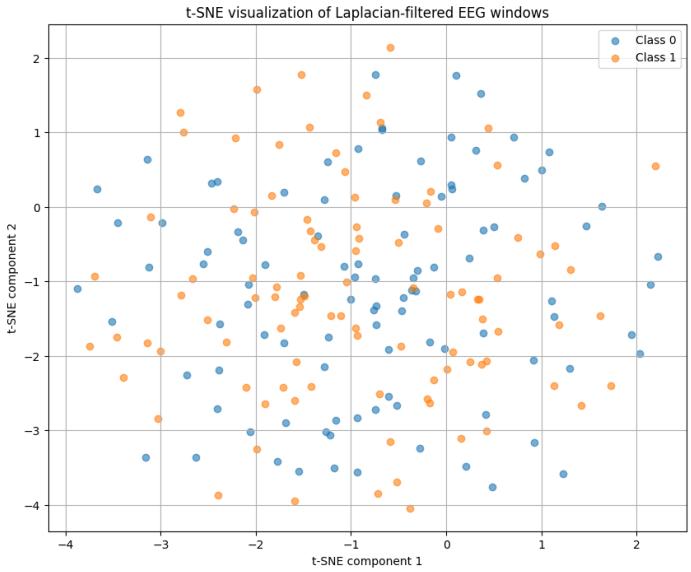


Figure 17 t-SNE plot for Laplacian filter

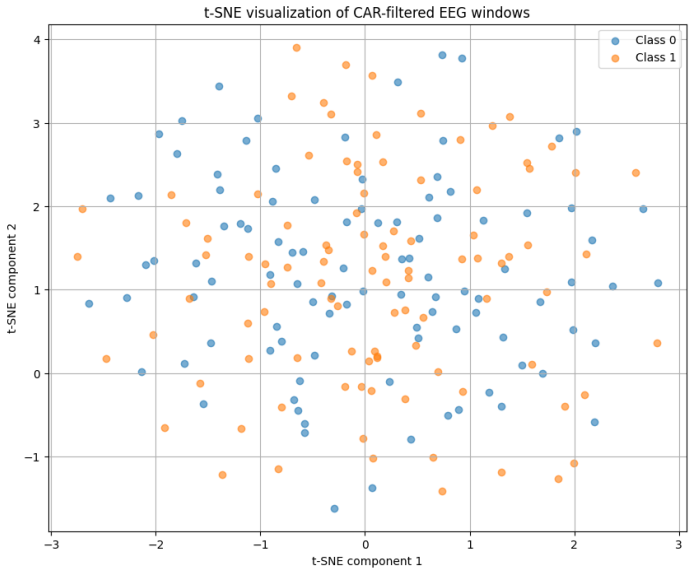


Figure 18 t-SNE plot for CAR filter

Just like before, the issue of high scatter and overlap prevents our classification. Therefore, we should use the methods mentioned earlier that we've used in previous sections for this part as well.

Now let's focus on feature extraction.

Feature Extraction for BCICIV_calib_ds1d

In this section again, we will use two methods for feature extraction: CSP (Common Spatial Patterns) and Wavelet Transform.

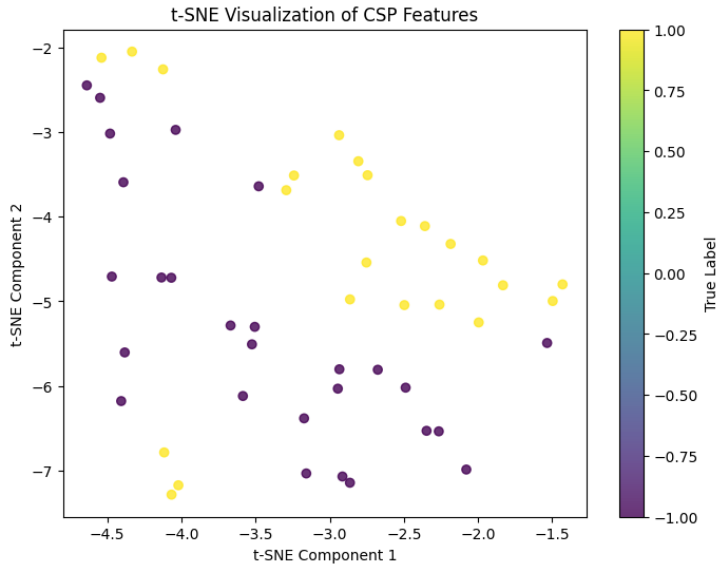


Figure 19 t-SNE visualization for CSP features

The plot above shows us the impact of CSP on feature extraction, and we can clearly observe the difference compared to the previous t-SNE plot.

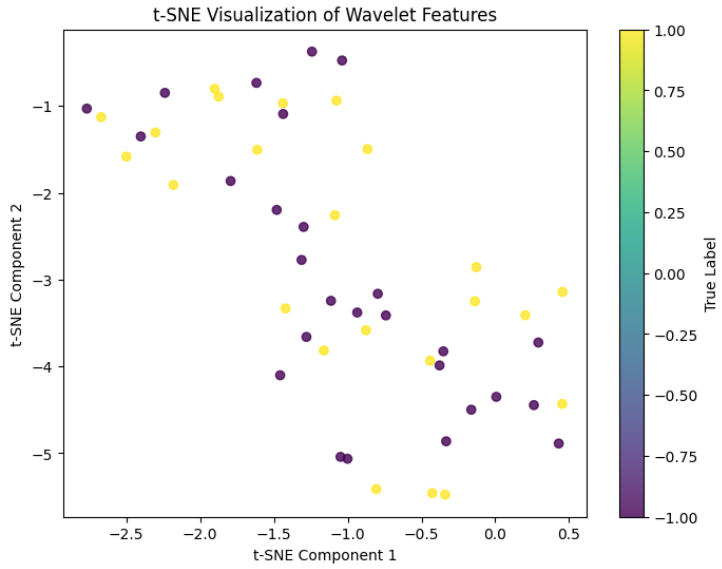


Figure 20 t-SNE visualization for Wavelet features

It is predicted that the classification results using the CSP method will be better.

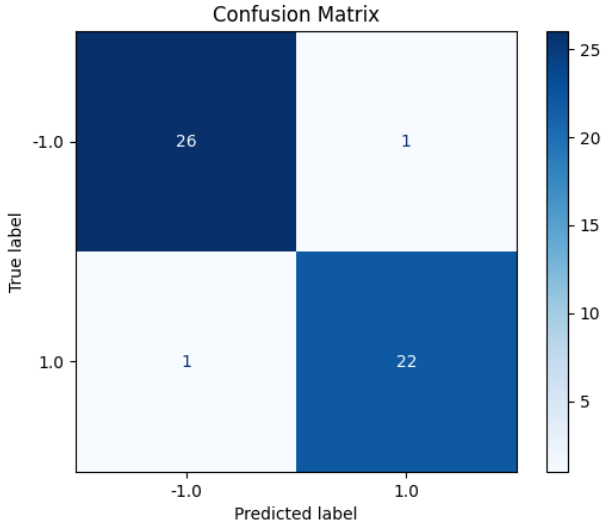
Classification for BCICIV_calib_ds1d

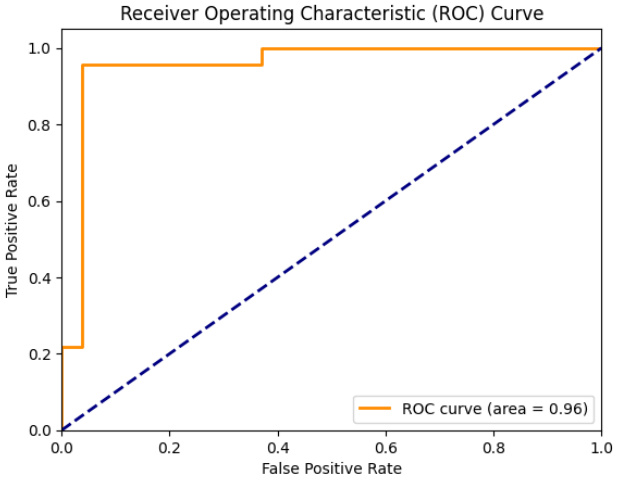
Then, we proceed with data extraction using CSP and classify using the obtained features.

Logistic Regression (Laplacian Filter and CSP):

```
Accuracy on test set: 0.96
```

```
Class -1 error: 0.04
Class 1 error: 0.04
```

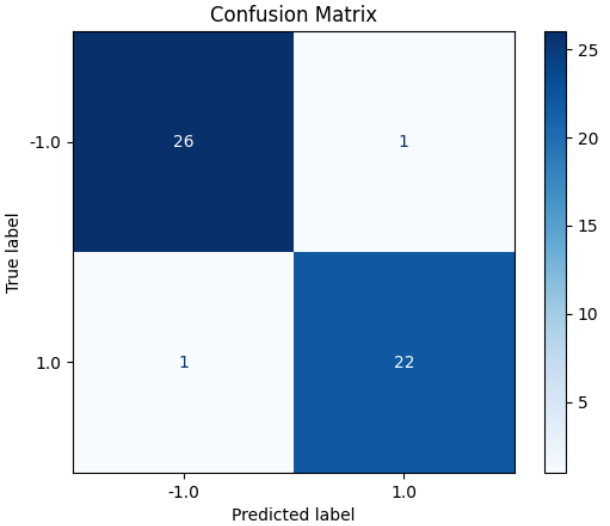


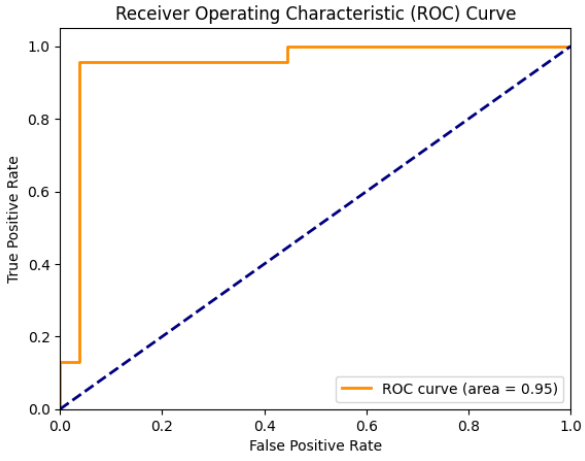


Logistic Regression (CAR Filter and CSP):

```
Accuracy on test set: 0.96
```

```
Class -1 error: 0.04
Class 1 error: 0.04
```

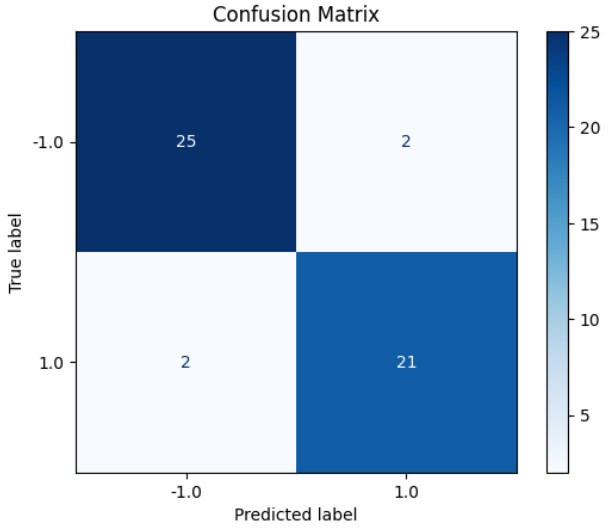


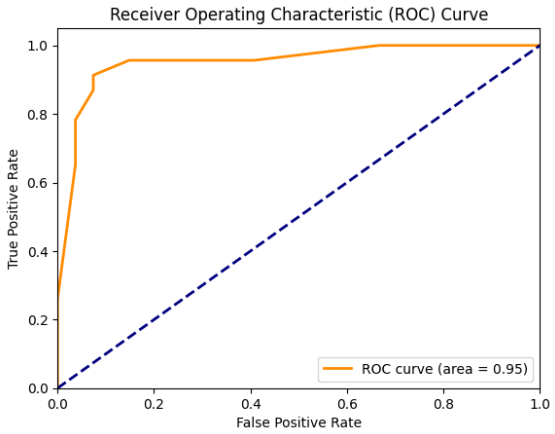


KNN (Laplacian Filter and CSP):

```
Accuracy on test set: 0.92
```

```
Class -1 error: 0.07
Class 1 error: 0.09
```

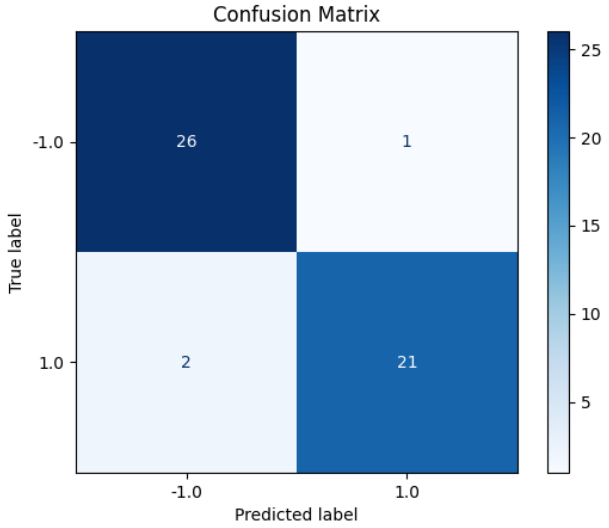


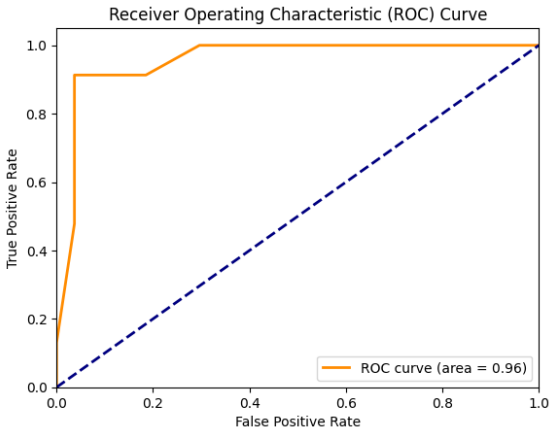


KNN (CAR Filter and CSP):

```
Accuracy on test set: 0.94
```

```
Class -1 error: 0.04
Class 1 error: 0.09
```

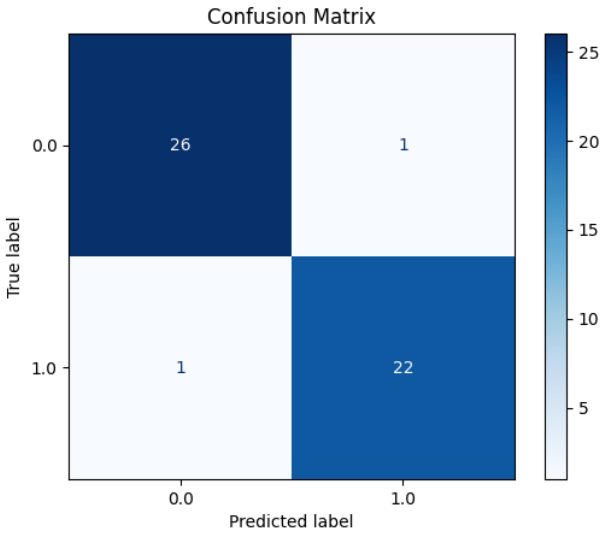


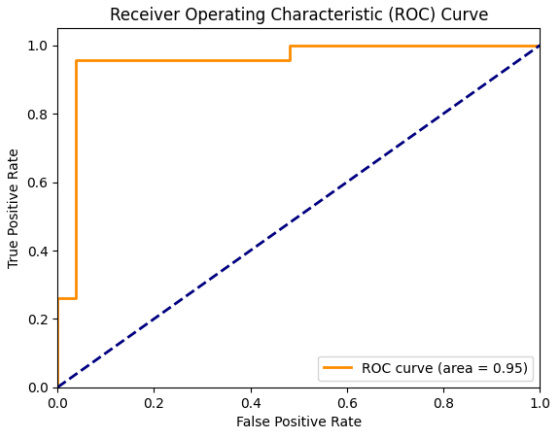


MLP (Laplacian Filter and CSP):

```
Accuracy on test set: 0.96
```

```
Class 0 error: 0.04
Class 1 error: 0.04
```

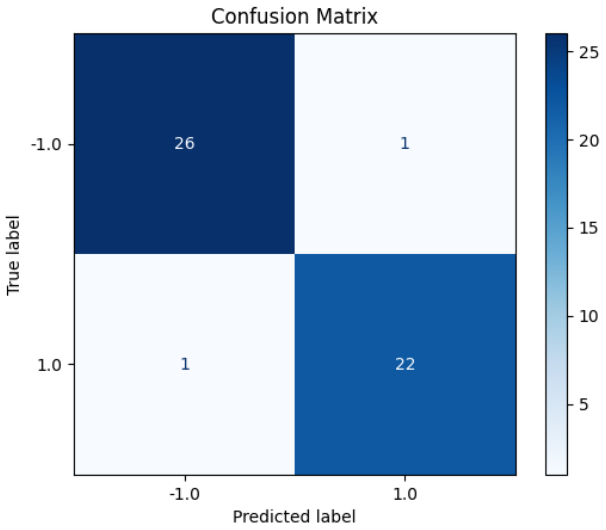


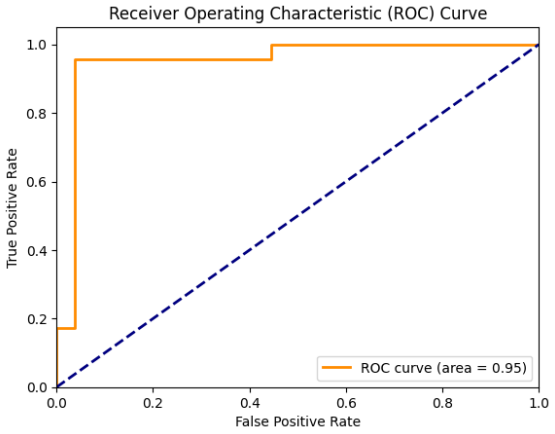


MLP (CAR Filter and CSP):

```
Accuracy on test set: 0.96
```

```
Class -1 error: 0.04
Class 1 error: 0.04
```

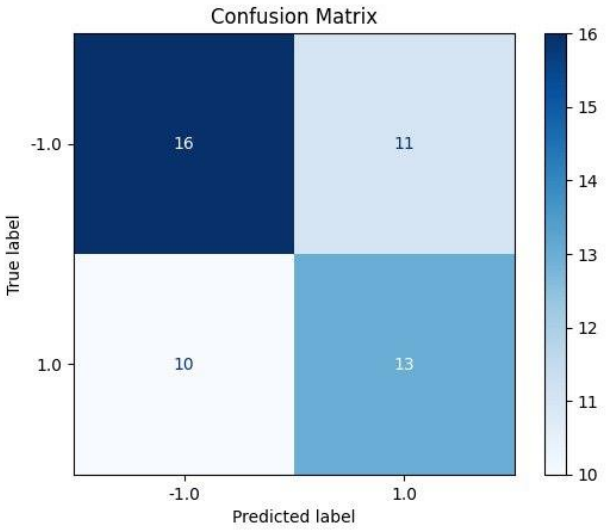


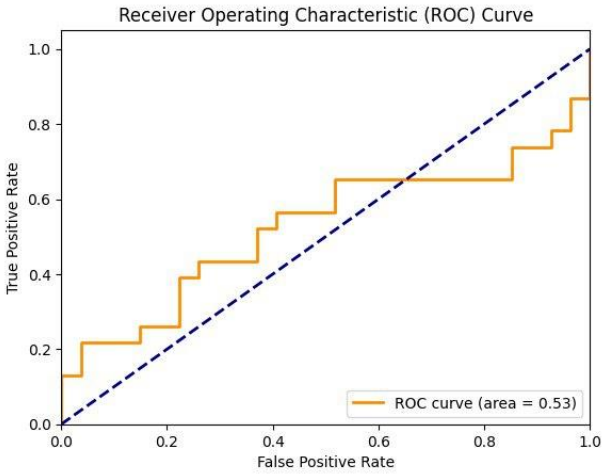


Now, let's examine the wavelet method.

Logistic Regression (Laplacian Filter and Wavelet Transform):

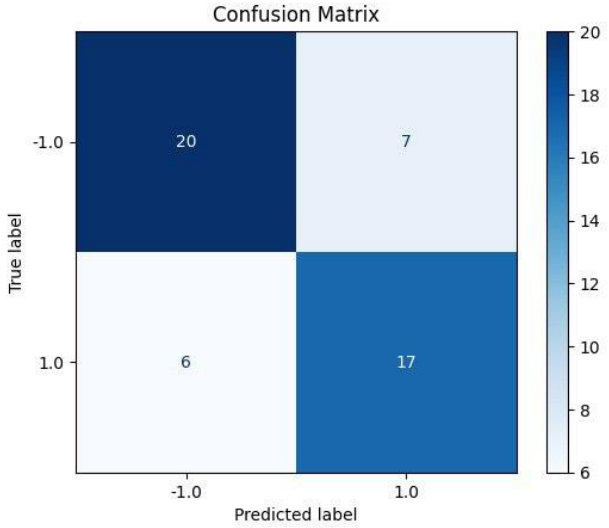
```
Accuracy on test set: 0.58
Class -1 error: 0.41
Class 1 error: 0.43
```

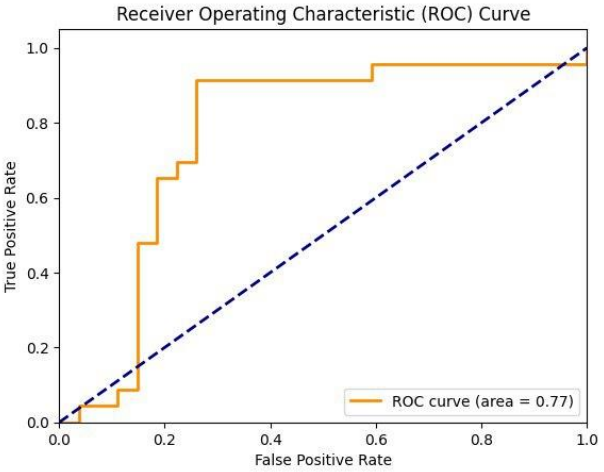




Logistic Regression (CAR Filter and Wavelet Transform):

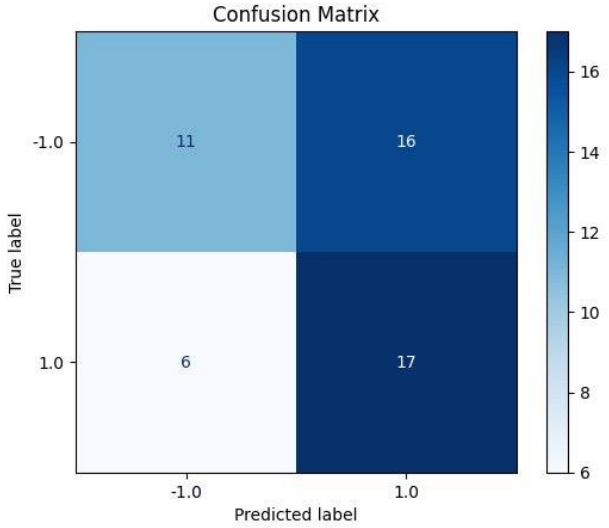
```
Accuracy on test set: 0.74
Class -1 error: 0.26
Class 1 error: 0.26
```

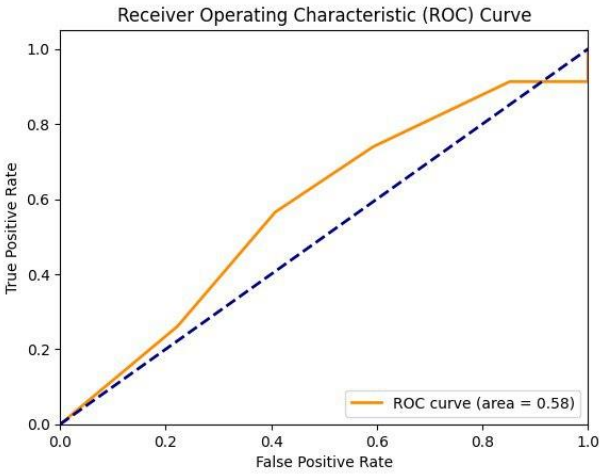




KNN (Laplacian Filter and Wavelet Transform):

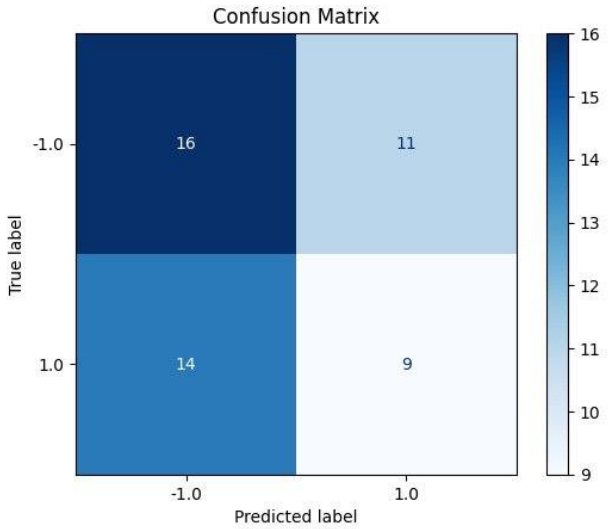
```
Accuracy on test set: 0.56
Class -1 error: 0.59
Class 1 error: 0.26
```

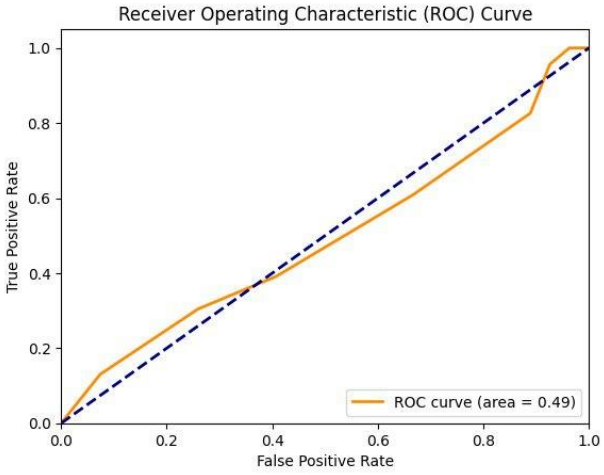




KNN (CAR Filter and Wavelet Transform):

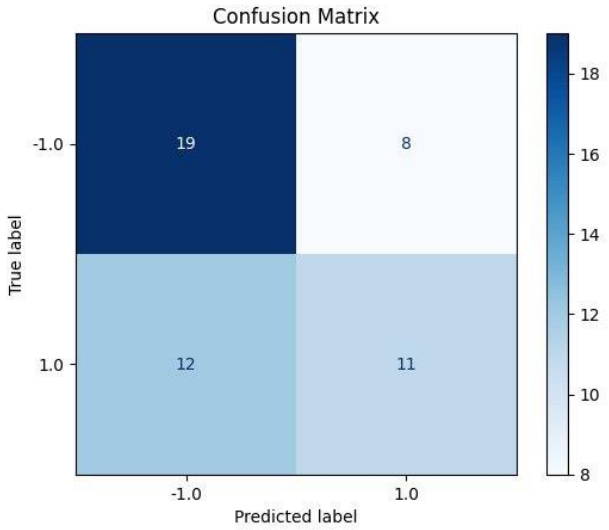
```
Accuracy on test set: 0.50
Class -1 error: 0.41
Class 1 error: 0.61
```

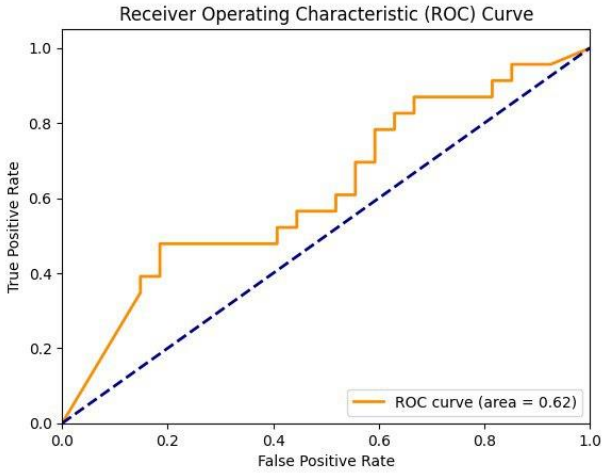




MLP (Laplacian Filter and Wavelet Transform):

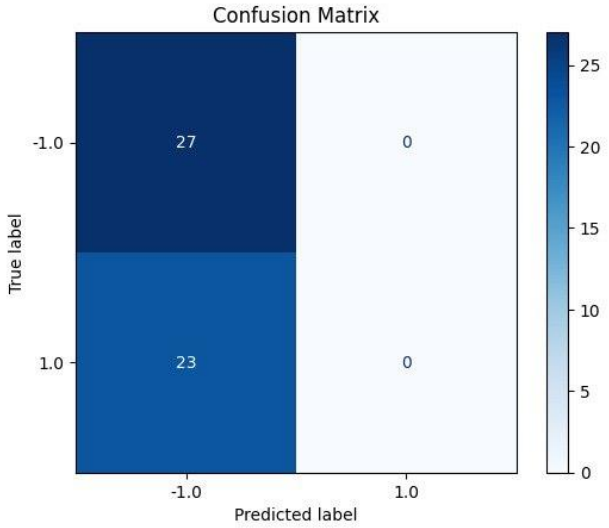
```
Accuracy on test set: 0.60
Class -1 error: 0.30
Class 1 error: 0.52
```

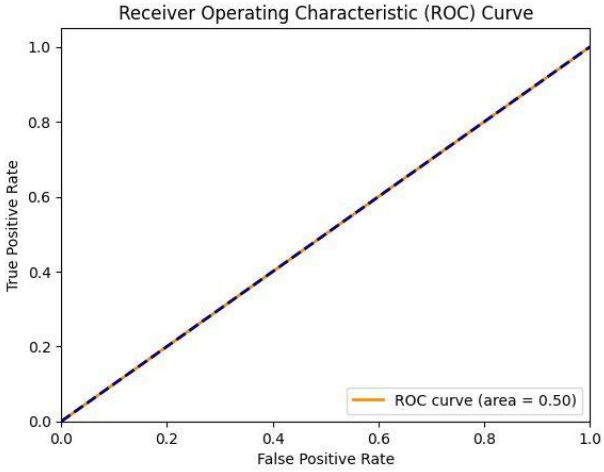




MLP (CAR Filter and Wavelet Transform):

```
Accuracy on test set: 0.54
Class -1 error: 0.00
Class 1 error: 1.00
```





Accuracy on test set table:

Special filter/Classification method	Logistic regression	KNN	MLP
Laplacian filter & CSP	96%	92%	96%
CAR filter & CSP	96%	94%	96%
Laplacian filter & Wavelet	56%	56%	60%
CAR filter & Wavelet	74%	50%	54%

As predicted, the accuracy of the classifiers obtained using the wavelet method is much lower than that of CSP. This was anticipated from their t-SNE plots, where the features overlapped and had significant overlap.

Clustering for BCICIV_calib_ds1d

All the explanations are similar to what was done in the previous dataset, and only the results are given here.

Gaussian Mixture Models (GMM):

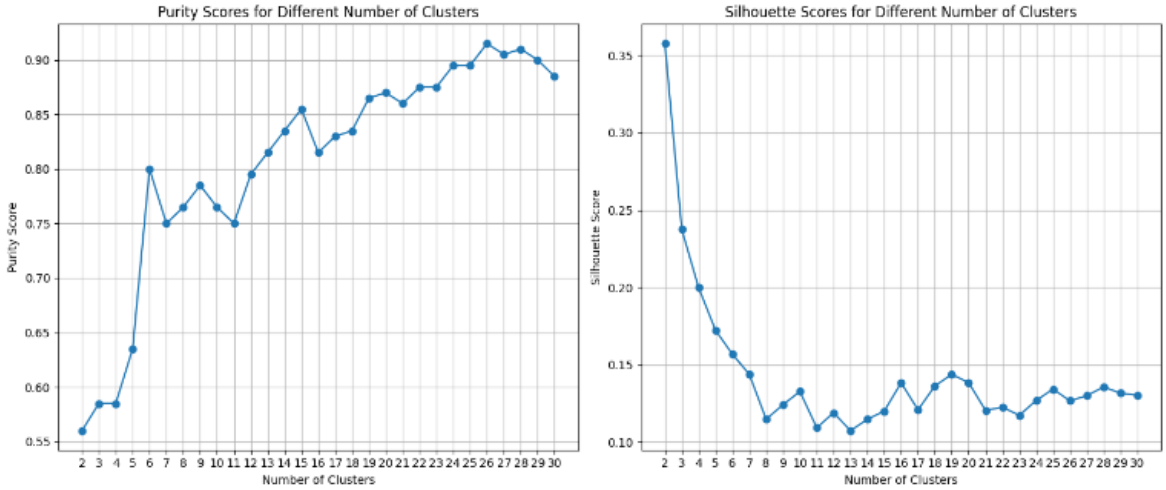


Figure 21 Purity scores and silhouette scores for different number of clusters

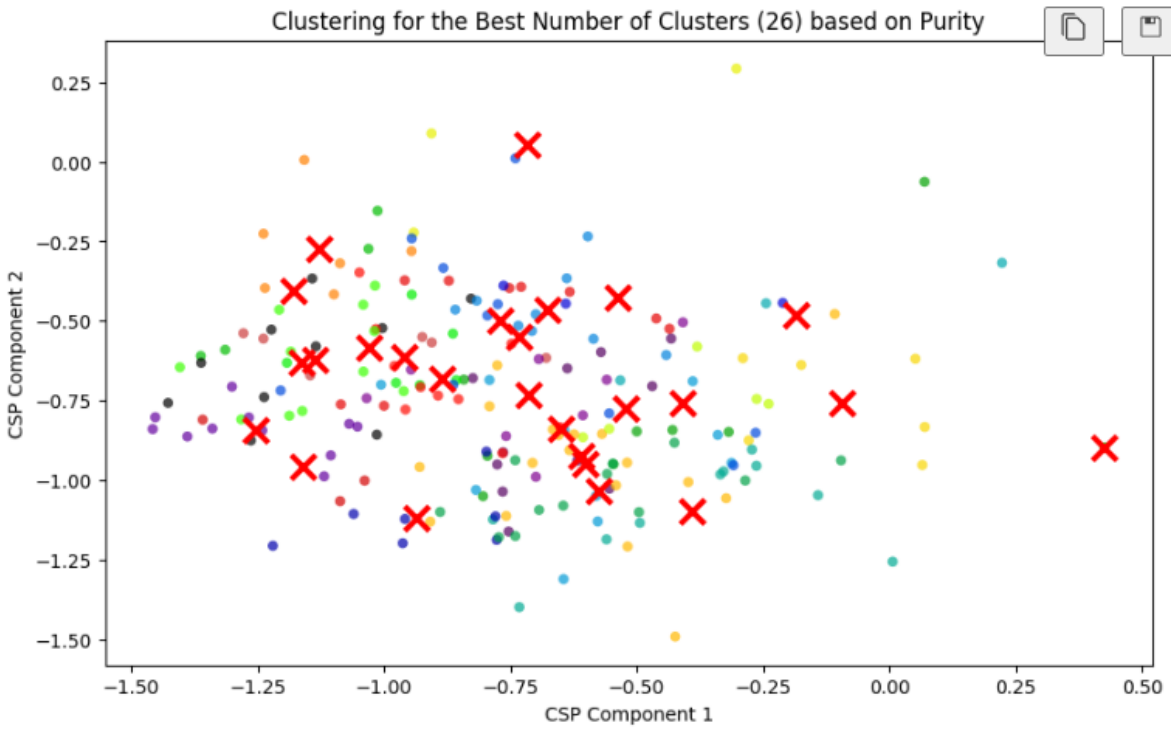


Figure 22 graph of cluster

K-Means:

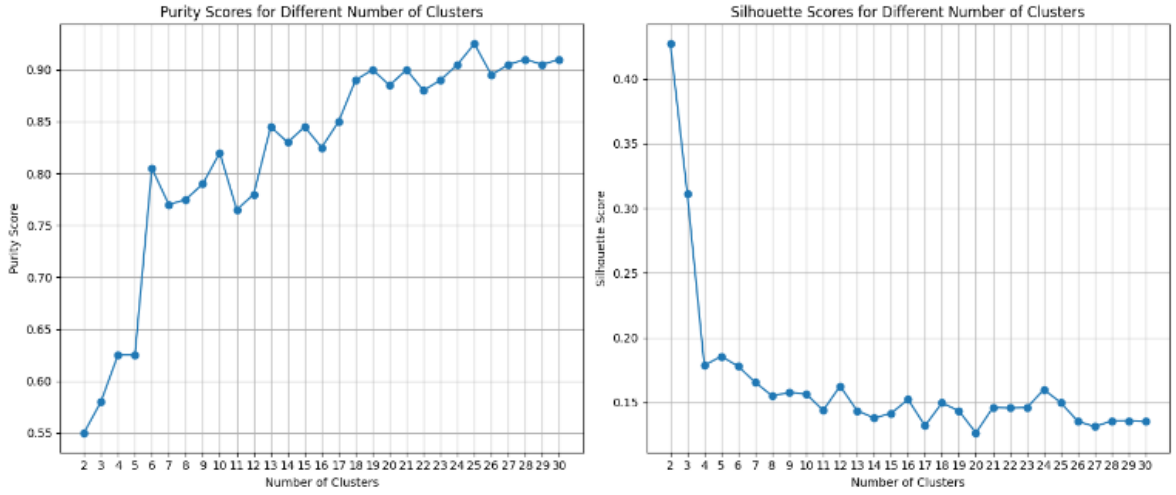


Figure 23 Purity scores and silhouette scores for different number of clusters

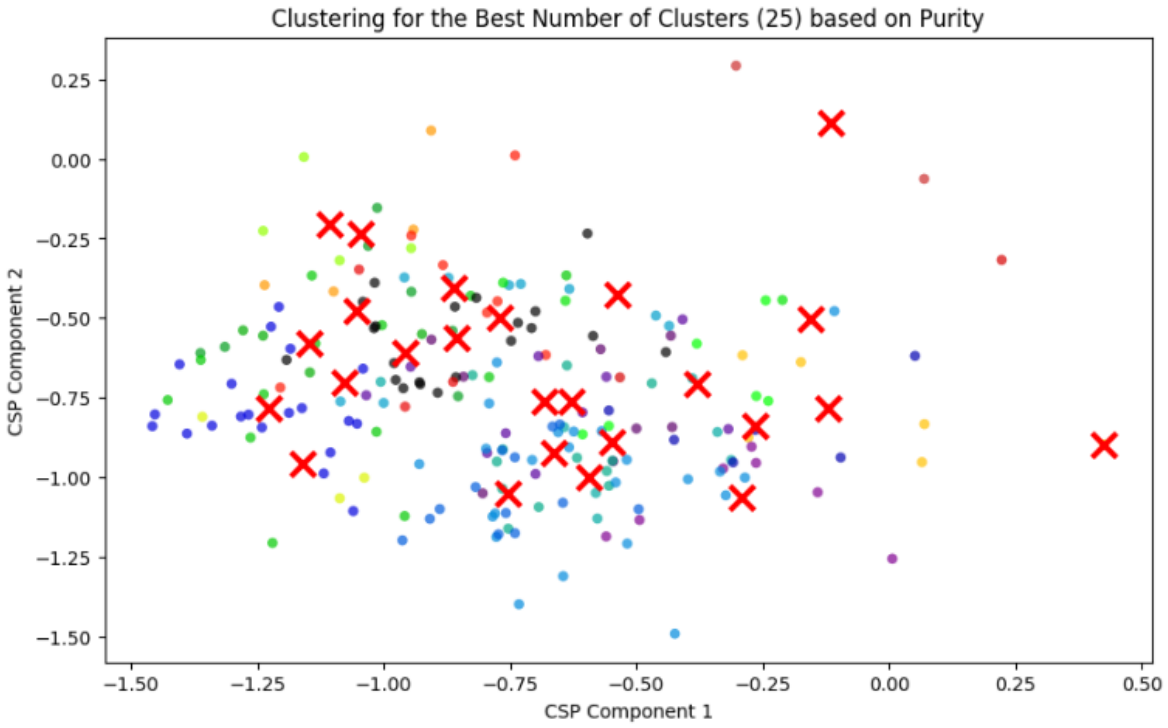


Figure 24 graph of clusters