

بسم الله الرحمن الرحيم



دانشکده مهندسی برق و کامپیوتر
یادگیری ماشین - فاز اول پروژه
سید مهدی رضوی - پونه شب دینی
استاد : آقای دکتر توسلی پور - آقای دکتر ابوالقاسمی

سید مهدی رضوی - پونه شب دینی

تیر ماه ۱۴۰۳



فهرست مطالب

۳	پیش‌پردازش داده‌ها	۱
۴	Common Average Reference Filter	۲
۸	PCA vs Independent Component Analysis	۳
۹	Algorithms for Classifying and Clustering	۴
۹	CSP filter Computation	۵
۹	نتیجه‌گیری	۶

فهرست تصاویر

۳	Importing necessary libraries	۱
۴	Load mat file and nyquist rate	۲
۵	PCA ICA implementation	۳
۶	Importing necessary libraries	۴
۷	t-SNE Visualization of CAR filtered features	۵
۸	Raw EEG Signals for Channels	۶
۹	Filtered Raw EEG Signals for Channels	۷
۹	CAR Filtered Raw EEG Signals for Channels	۸
۱۰	PCA for Channels	۹
۱۰	KNN Algorithms , Precison , Recall	۱۰
۱۱	SVM Algorithms , Precison , Recall	۱۱
۱۲	MLP Algorithms , Precison , Recall	۱۲
۱۳	AdaBoost Algorithms , Precison , Recall	۱۳
۱۴	Silhouette Score for Clustering	۱۴
۱۵	t-SNE KMeans	۱۵
۱۶	یک نمونه از استخراج ویژگی مرتبط به وسیله CSP	۱۶
۱۷	Plot function	۱۷
۱۸	Classifiers and ClusterMethods	۱۸



۱ پیشپردازش داده‌ها

ابتدا کتابخانه‌های مدنظر برای استخراج داده‌ها از این نوع فایل را قرارمی‌دهیم و سپس داده‌های سیگنال‌های EEG را استخراج خواهیم کرد.

```
[ ] from google.colab import drive
drive.mount('/content/drive')
folder_path = '/content/drive/MyDrive/ML_Project/BCICIV_1_mat/'

→ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

▶ import numpy as np
import scipy.io
from scipy.signal import butter, filtfilt
from scipy.linalg import eigh
from sklearn.decomposition import PCA, FastICA
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, roc_curve, auc, classification_report, silhouette_score
from sklearn.cluster import KMeans, AgglomerativeClustering
import matplotlib.pyplot as plt
from numpy.linalg import LinAlgError
```

شکل ۱ : Importing necessary libraries

این کد یک تابع به نام butterbandpass دارد که برای ایجاد فیلتر باندپاس با تروث استفاده می‌شود. این فیلتر برای حذف فرکانس‌های غیرمطلوب در یک سیگنال استفاده می‌شود.

مقادیر ورودی این تابع عبارتند از:
lowcut حد پایین فرکانس باندپاس
highcut حد بالای فرکانس باندپاس
fs نرخ نمونهبرداری سیگنال ورودی
order مرتبه فیلتر باندپاس (مقدار پیشفرض ۳ است)

در این تابع، ابتدا نیکویست (نصف نرخ نمونهبرداری) محاسبه می‌شود. سپس مقادیر low و high بر اساس lowcut و highcut محاسبه می‌شوند. سپس با استفاده از تابع butter از کتابخانه scipy ضرایب b و a برای فیلتر باندپاس محاسبه می‌شوند. سپس تابع دیگری به نام bandpassfilter تعریف شده است که از فیلتر باندپاس با تروث استفاده می‌کند. این تابع سه ورودی دارد data (سیگنال ورودی)، lowcut و highcut (حد پایین و بالای فرکانس باندپاس) (نرخ نمونهبرداری سیگنال ورودی) و order (مقدار پیشفرض ۳ است).

در این تابع، ابتدا با استفاده از تابع butterbandpass ضرایب b و a برای فیلتر باندپاس محاسبه می‌شوند. سپس با استفاده از تابع filtfilt از کتابخانه scipy سیگنال ورودی data با استفاده از ضرایب b و a فیلتر شده و در y ذخیره می‌شود. سپس y به عنوان خروجی تابع برگشت داده می‌شود.



```
import os

mat_file_paths = [f for f in os.listdir(folder_path) if f.endswith('.mat')]

def load_mat_file(file_path):
    mat_contents = scipy.io.loadmat(file_path)
    eeg_data = mat_contents['cnt']
    mrk = mat_contents['mrk']
    nfo = mat_contents['nfo']
    fs = int(nfo['fs'][0, 0])
    return eeg_data, mrk, fs

lowcut = 8.0
highcut = 30.0
order = 3

def butter_bandpass(lowcut, highcut, fs, order=3):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return b, a

def bandpass_filter(data, lowcut, highcut, fs, order=3):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data, axis=0)
    return y
```

شکل ۲: Load mat file and nyquist rate

Common Average Reference Filter ۲

یک تکنیک رایج برای استخراج ویژگی‌ها در تجزیه و تحلیل داده‌های چندکاناله مانند سیگنال‌های EEG است.

فیلتر CAR برای حذف نویز یا اختلال مشترک موجود در همه کانال‌های داده استفاده می‌شود. ایده پشت فیلتر CAR این است که متوسط سیگنال در تمام کانال‌ها را از هر کانال فردی کم کند. این کار به حذف منابع نویز مشترک که بین تمام کانال‌ها به اشتراک گذاشته می‌شوند، مانند نویز محیطی، آرتیفیکت الکتروود یا دیگر منابع مشترک اختلالات کمک می‌کند. فیلتر CAR به صورت ریاضی به شکل زیر نمایش داده می‌شود:

$$y(t) = x(t) - \text{mean}(x)$$

که: $y(t)$ سیگنال فیلتر شده در زمان t است

$x(t)$ سیگنال اصلی در زمان t است

$\text{mean}(x)$ متوسط سیگنال در تمام کانال‌ها است

با کم کردن متوسط سیگنال از هر کانال، فیلتر CAR به طور موثری جزء نویز مشترک را حذف می‌کند و در عین حال اطلاعات خاص هر کانال را حفظ می‌کند. این کار می‌تواند نسبت سیگنال به نویز را بهبود بخشیده و توان تمایزدهی داده را برای وظایف تحلیل یا طبقه‌بندی بعدی افزایش دهد.

مهم است که توجه شود که هرچند فیلتر CAR ممکن است در حذف منابع نویز مشترک مؤثر باشد، اما ممکن است برای همه حالات یا انواع داده مناسب نباشد. در برخی موارد، تکنیک‌های دیگر مانند تکنیک استانداردسازی الکتروود مرجع REST یا فیلترینگ لاپلاسین برای ویژگی‌ها یا اهداف تحقیقات خاص، مناسب‌تر باشند.

```
epoch_start = -1.0
epoch_end = 2.0
epoch_duration = epoch_end - epoch_start

def epoch_data(data, markers, start, end):
    epochs = []
    for marker in markers:
        start_idx = marker + start
        end_idx = marker + end
        if start_idx >= 0 and end_idx <= data.shape[0]:
            epoch = data[start_idx:end_idx, :]
            epochs.append(epoch)
    return np.array(epochs)

def apply_car_filter(epochs):
    car_epochs = epochs - np.mean(epochs, axis=2, keepdims=True)
    return car_epochs

def apply_pca(data, variance=0.95):
    pca = PCA(n_components=variance)
    pca_filtered_data = pca.fit_transform(data.reshape(-1, data.shape[-1]))
    return pca_filtered_data.reshape(data.shape[0], data.shape[1], -1)

def apply_ica(data, max_iter=1000, tol=0.001):
    ica = FastICA(n_components=data.shape[-1], max_iter=max_iter, tol=tol)
    ica_filtered_data = ica.fit_transform(data.reshape(-1, data.shape[-1]))
    return ica_filtered_data.reshape(data.shape[0], data.shape[1], -1)
```

شکل ۳ : PCA ICA implementation

این کد یک سری توابع برای پردازش داده‌های سیگنال‌های زمانی مانند سیگنال‌های EEG است. این توابع شامل تقسیم داده به دوره‌ها epoch اعمال فیلترهای پردازش سیگنال و تبدیل‌های مختلف برای تحلیل سیگنال‌ها می‌باشد.

تابع epochdata یک داده و یک لیست از نشانگرها markers را به عنوان ورودی می‌گیرد و داده را بر اساس نشانگرها به دوره‌ها تقسیم می‌کند. این دوره‌ها در یک لیست قرار داده می‌شوند و به عنوان خروجی برگشت داده می‌شوند.

تابع applyCARfilter یک لیست از دوره‌ها را به عنوان ورودی گرفته و فیلتر کار Reference Average Common را بر روی هر دوره اعمال می‌کند. این فیلتر با کم کردن میانگین کل داده از هر دوره، نویز مشترک را کاهش می‌دهد.

تابع applypca یک داده را به عنوان ورودی گرفته و با استفاده از تحلیل ترکیب اصلی PCA ابعاد آن را کاهش می‌دهد تا درصد مشخصی از واریانس داده را حفظ کند.

تابع applyICA نیز یک داده را به عنوان ورودی گرفته و با استفاده از تحلیل مولفه‌های مستقل ICA سعی می‌کند تا مولفه‌های مستقل اصلی داده را استخراج کند.



```
[ ] def regularize_cov(cov, alpha=1e-5):
    return cov + alpha * np.eye(cov.shape[0])

▶ def compute_csp(epochs, labels, alpha=1e-5):
    class_1_epochs = epochs[labels == 1]
    class_2_epochs = epochs[labels == -1]

    cov_1 = np.mean([np.cov(epoch, rowvar=False) for epoch in class_1_epochs], axis=0)
    cov_2 = np.mean([np.cov(epoch, rowvar=False) for epoch in class_2_epochs], axis=0)

    cov_1 = regularize_cov(cov_1, alpha)
    cov_2 = regularize_cov(cov_2, alpha)

    try:
        evals, evecs = eigh(cov_1, cov_1 + cov_2)
        sorted_indices = np.argsort(evals)[::-1]
        evecs = evecs[:, sorted_indices]
    except LinAlgError as e:
        print("Error computing CSP filters:", e)
        return None

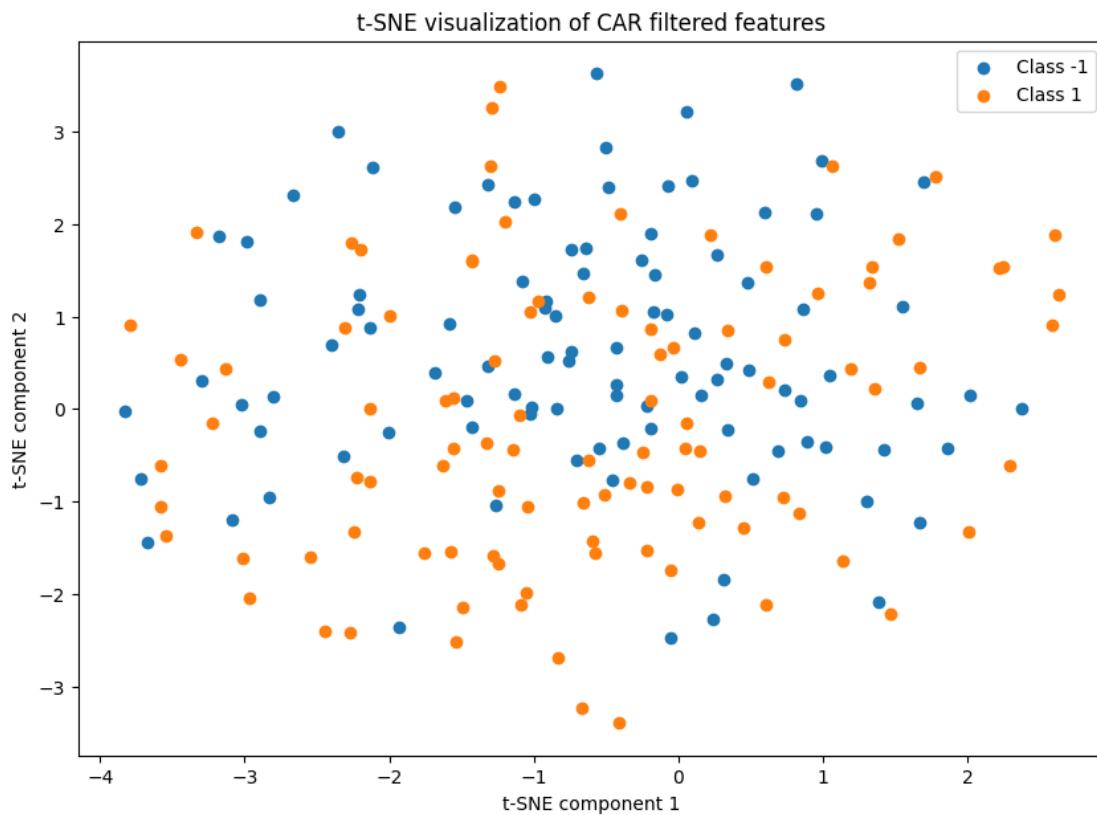
    return evecs
```

شکل ۴ : Importing necessary libraries

این کد یک تابع به نام `computeCSP` را تعریف می‌کند که دو کلاس از داده‌ها را بر اساس برچسب‌ها `labels` محاسبه و فیلتر `CSP` را برای تفکیک این دو کلاس اعمال می‌کند.

در این تابع، ابتدا داده‌ها به دو کلاس مختلف تقسیم می‌شوند. سپس برای هر کلاس، ماتریس کوواریانس محاسبه شده و با استفاده از تابع `regularizeCov` ماتریس کوواریانس منظم‌سازی می‌شود. سپس با استفاده از تحلیل مقدار و بردار ویژه `eigh` ویژه‌ها و بردارهای ویژه ماتریس‌های کوواریانس محاسبه شده و مرتب شده و برگردانده می‌شود.

در صورت بروز خطا در محاسبات، یک پیام خطأ چاپ می‌شود و تابع `None` برگردانده می‌شود. در غیر این صورت، بردارهای ویژه به عنوان خروجی تابع برگشت داده می‌شود.



شکل ۵: t-SNE Visualization of CAR filtered features

این تابع یک مسیر فایل را به عنوان ورودی دریافت می‌کند و سپس داده‌های EEG و MRK را از یک فایل ماتلب بارگذاری می‌کند. در ادامه، داده‌های EEG را به نوع داده اعشاری ۳۲ بیت تبدیل کرده و سپس با استفاده از تابع `bandpassfilter` فیلتر باندهای تفکیک‌کننده را بر روی داده‌های EEG اعمال می‌کند. سپس نشانگرهای رویداد markers event و کلاس‌های رویداد از داده MRK استخراج می‌شوند. سپس مقادیر مورد نیاز برای تقسیم داده به دوره‌ها epochs محاسبه می‌شود.

سپس داده‌های EEG به دوره‌های Reference Average Common CAR تقسیم شده و سپس فیلتر t-SNE بر روی دوره‌های EEG اعمال می‌شود. سپس فیلتر Analysis Component Independent ICA و Analysis Component Principal PCA به دوره‌های فیلتر شده اعمال می‌شود.

سپس ویژگی‌های استخراج شده با استفاده از t-SNE Embedding Neighbor Stochastic t-distributed t-SNE نگاشت می‌شود و نمودار t-SNE برای دیداری ساخته شده و نمایش داده می‌شود. سپس فیلتر Spatial Common CSP Patterns برای تفکیک دو کلاس از دوره‌های فیلتر شده CAR محاسبه می‌شود. در صورت موفقیت، این فیلتر CSP بر روی دوره‌های فیلتر شده CAR اعمال شده و سپس داده‌ها به ویژگی‌های CSP تبدیل شده و آماده برای آموزش مدل می‌شود.

در انتها، چندین مدل آموزش دیده و نتایج آن‌ها چاپ می‌شود. همچنین نمودار ROC Characteristic Operating Receiver برای هر مدل نیز نمایش داده می‌شود.

PCA vs Independent Component Analysis ۳

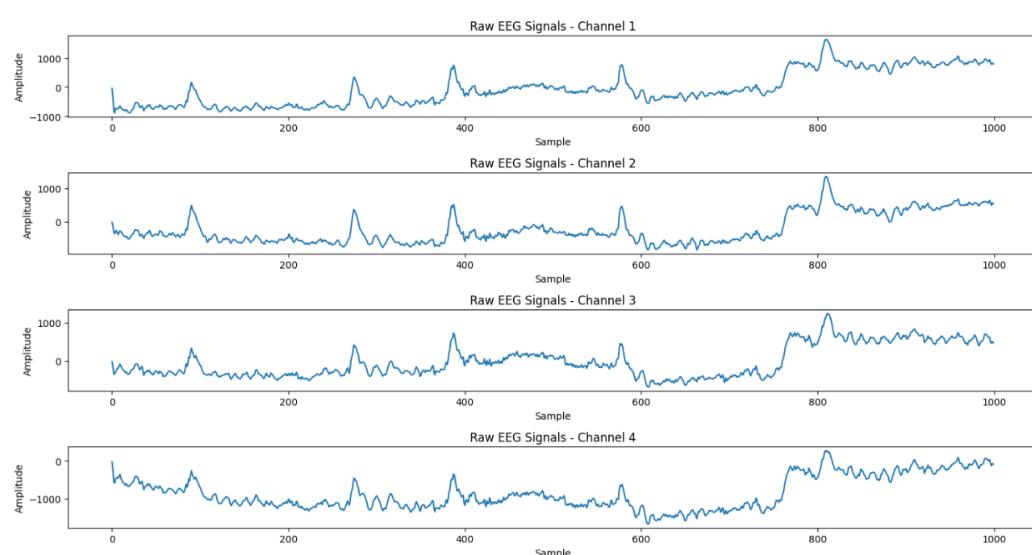
الگوریتم K-Nearest Neighbors (K-Nearest KNN) ، برای پیش‌بینی برچسب یک نمونه جدید، از برچسب نمونه‌های مشابهی که در مجموعه داده وجود دارند استفاده می‌شود. این الگوریتم بر اساس فاصله‌ی نمونه‌ها از همیگر، نزدیک‌ترین K نمونه به نمونه جدید را انتخاب کرده و با توجه به برچسب آن‌ها، برچسب نمونه جدید را پیش‌بینی می‌کند.

الگوریتم Multi-Layer MLP (Multi-Layer MLP Perceptron) یک شبکه عصبی ثرف network neural deep است که شامل چندین لایه از نورون‌ها است. این الگوریتم با استفاده از یک فرآیند آموزش، وزن‌های مختلف بین لایه‌ها را بروزرسانی می‌کند تا بتواند الگوهای پیچیده‌تر را در داده‌ها شناسایی کند.

الگوریتم AdaBoost: Adaptive Boosting یک الگوریتم یادگیری ماشین مجموعه‌ای learning ensemble است که با استفاده از ترکیب چند مدل ضعیف learner weak یک مدل قوی learner strong ایجاد می‌کند. در هر مرحله از فرآیند آموزش، وزن‌های نمونه‌ها تغییر می‌کند تا به نمونه‌های دشوارتر بیشتر توجه شود و مدل بهبود یابد.

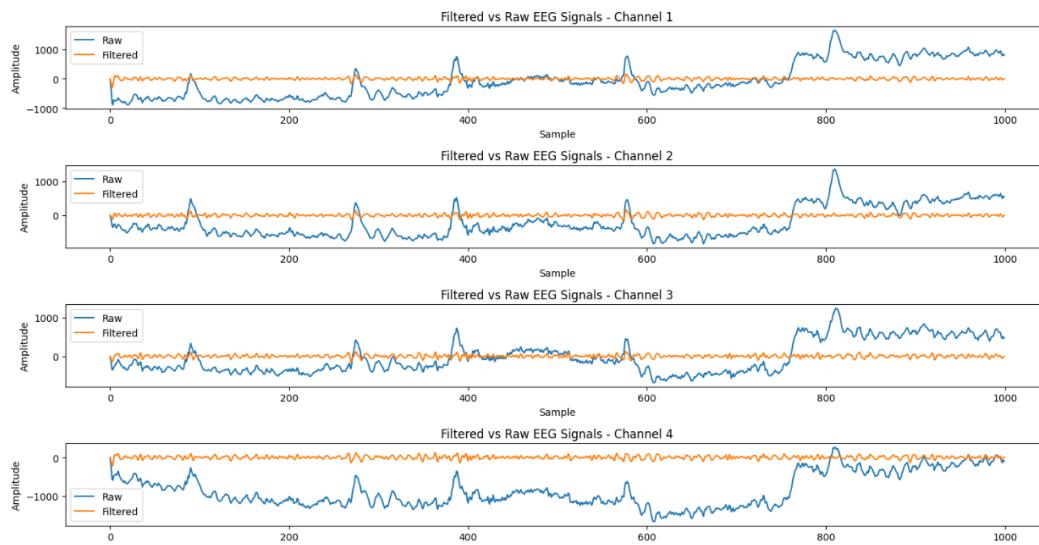
مقایسه: - الگوریتم ساده و قابل فهم است، اما در مقابل داده‌های بزرگ و پیچیده عملکرد خوبی ندارد. - MLP: قابلیت یادگیری الگوهای پیچیده را دارد ولی نیاز به تعداد زیادی پارامتر و داده آموزش دارد. - AdaBoost: قابلیت تطبیق با داده‌های دشوار و تولید مدل‌های قوی را دارد، اما حساس به داده‌های نویزی است.

با توجه به نوع داده و مساله مورد نظر، انتخاب الگوریتم مناسب بستگی به مواردی مانند حساسیت به داده‌های نویزی، اندازه داده، و پیچیدگی مساله دارد.

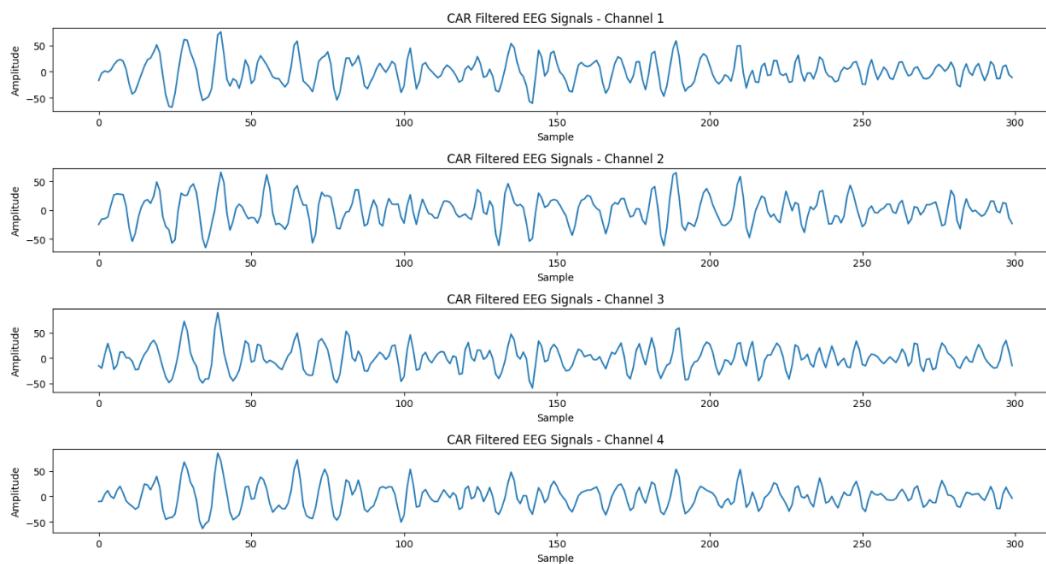


شکل ۶: Raw EEG Signals for Channels

سیگنال‌های خام EEG را در کanal‌های مختلف در تصویر زیر مشاهده می‌فرمایید.



شکل :۷ Filltered Raw EEG Signals for Channels



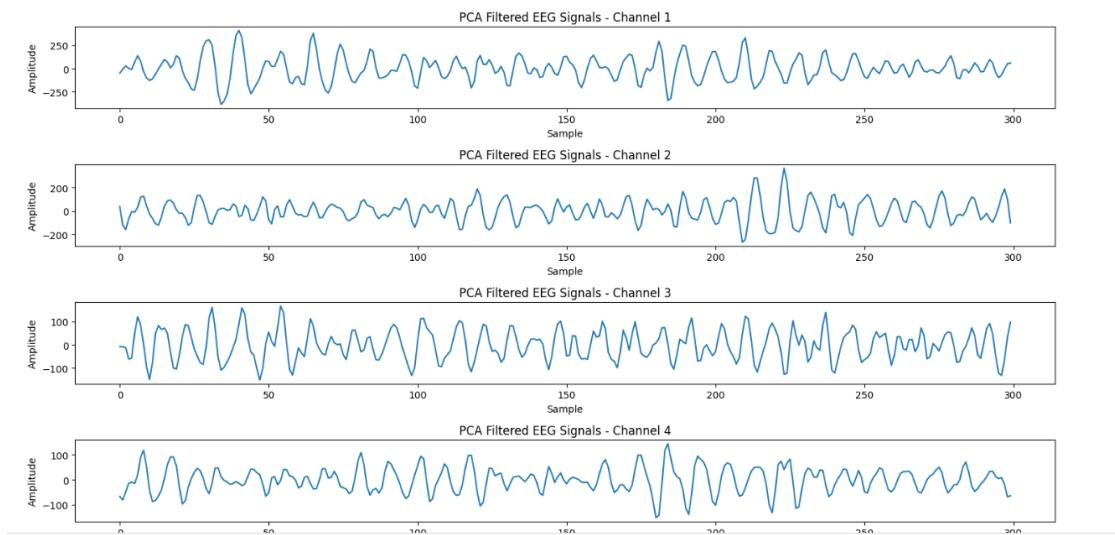
شکل :۸ CAR Filltered Raw EEG Signals for Channels

۴ Algorithms for Classifying and Clustering

۵ CSP filter Computation

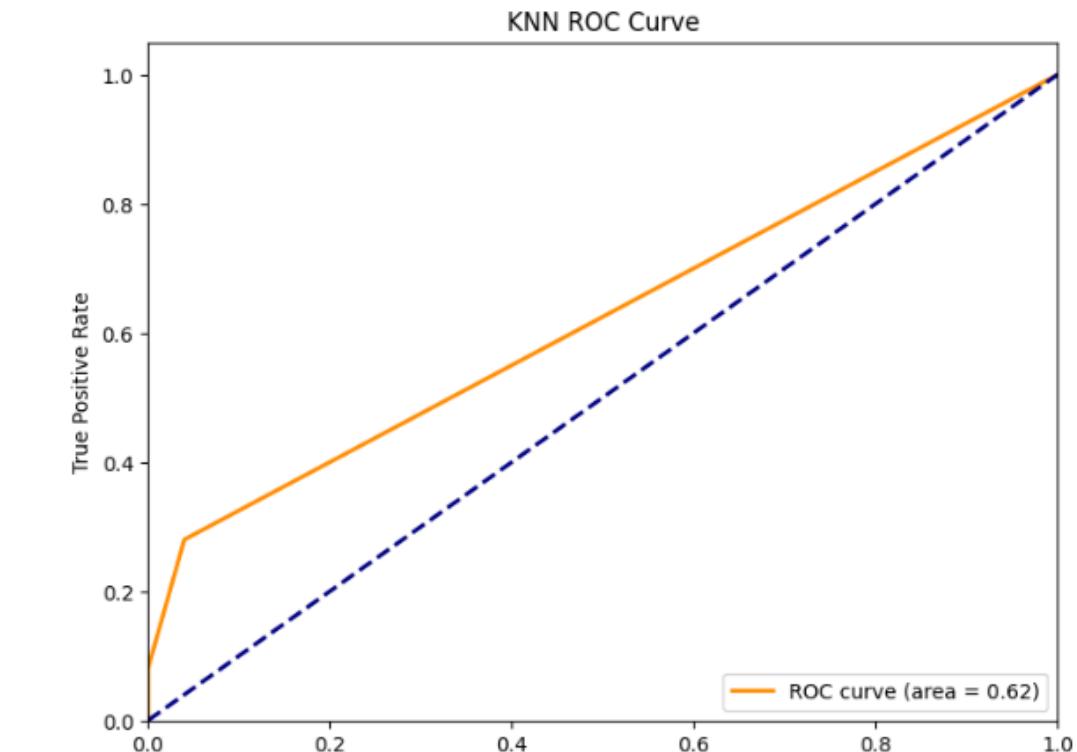
۶ نتیجه‌گیری

در بین الگوریتم های استفاده شده برای طبقه بندی دودویی سیگنال ها ، الگوریتم AdaBoost با اختلاف فاحشی بهتر عمل میکند . شاید این مشکل ناشی از کمبود تعداد داده آموزشی باشد ، اما قدرت الگوریتم AdaBoost در این مسئله میتواند ناشی از استفاده از مفهوم ترکیب چند طبقه بندی ضعیف در یافتن ویژگی مهم و بر جسب زدن داده ها بر اساس آن خواهد بود.



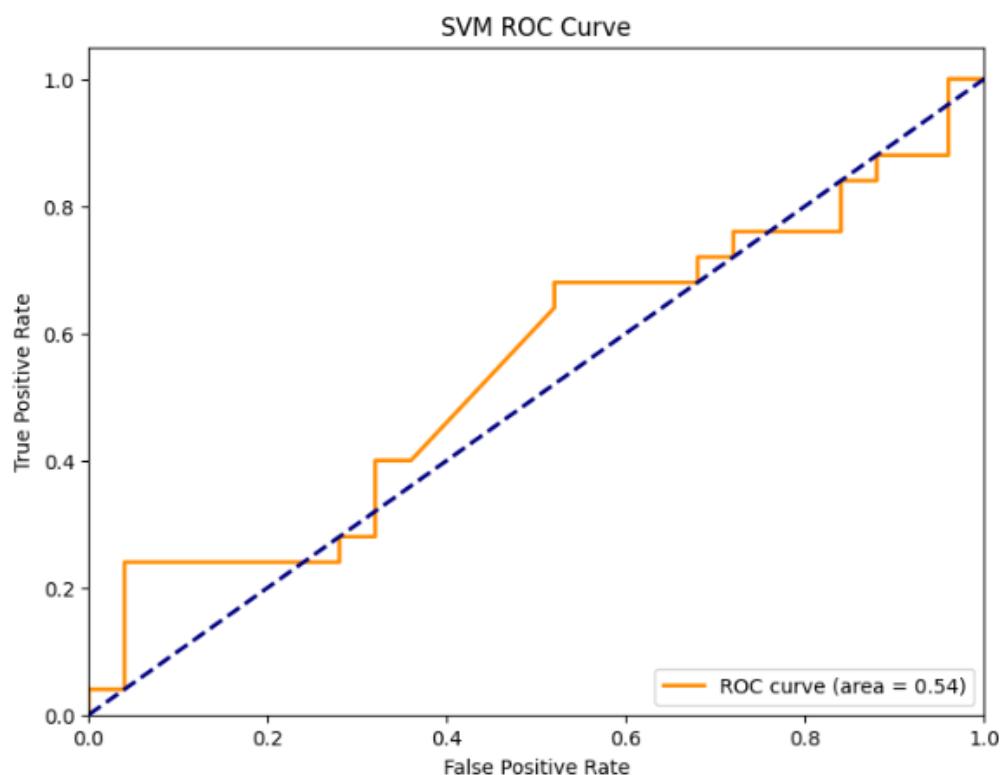
شکل ۹: PCA for Channels

```
KNN Confusion Matrix:  
[[25  0]  
 [25  0]]  
KNN Classification Report:  
          precision    recall   f1-score   support  
  
      -1       0.50     1.00     0.67      25  
       1       0.00     0.00     0.00      25  
  
accuracy                          0.50      50  
macro avg       0.25     0.50     0.33      50  
weighted avg     0.25     0.50     0.33      50  
  
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:  
  _warn_prf(average, modifier, msg_start, len(result))  
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:  
  _warn_prf(average, modifier, msg_start, len(result))  
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:  
  _warn_prf(average, modifier, msg_start, len(result))
```



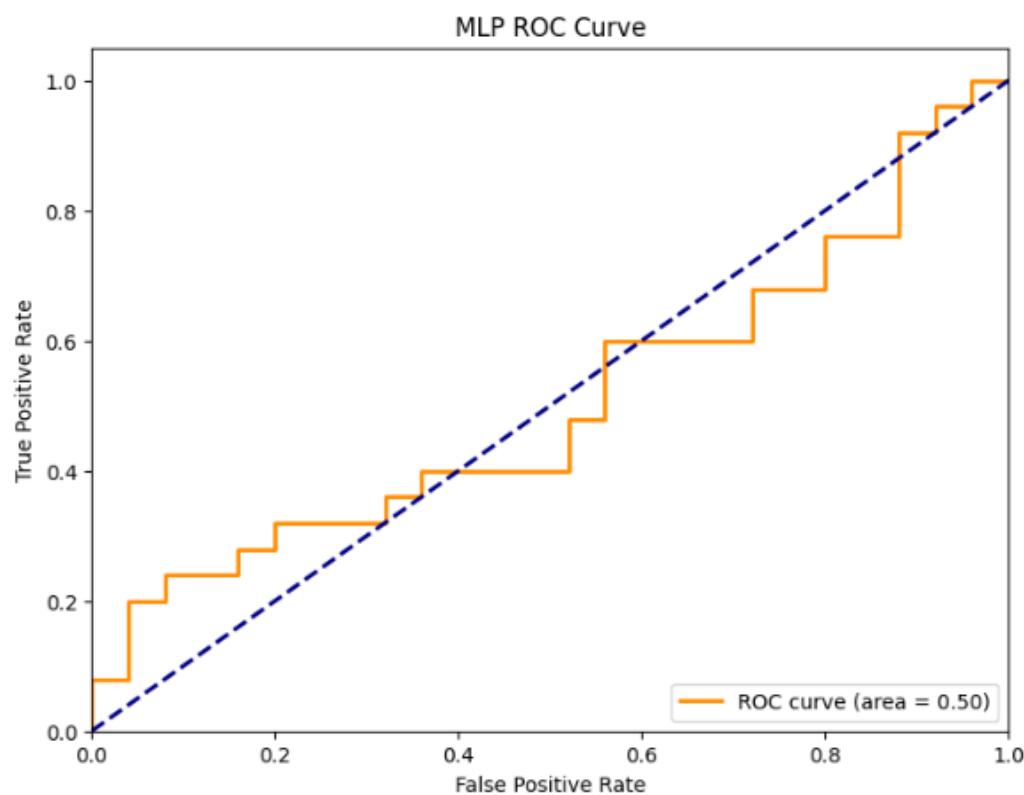
شکل ۱۰: KNN Algorithms , Precison , Recall

```
SVM Confusion Matrix:  
[[15 10]  
 [14 11]]  
SVM Classification Report:  
precision recall f1-score support  
-1 0.52 0.60 0.56 25  
 1 0.52 0.44 0.48 25  
  
accuracy 0.52  
macro avg 0.52 0.52 0.52 50  
weighted avg 0.52 0.52 0.52 50
```



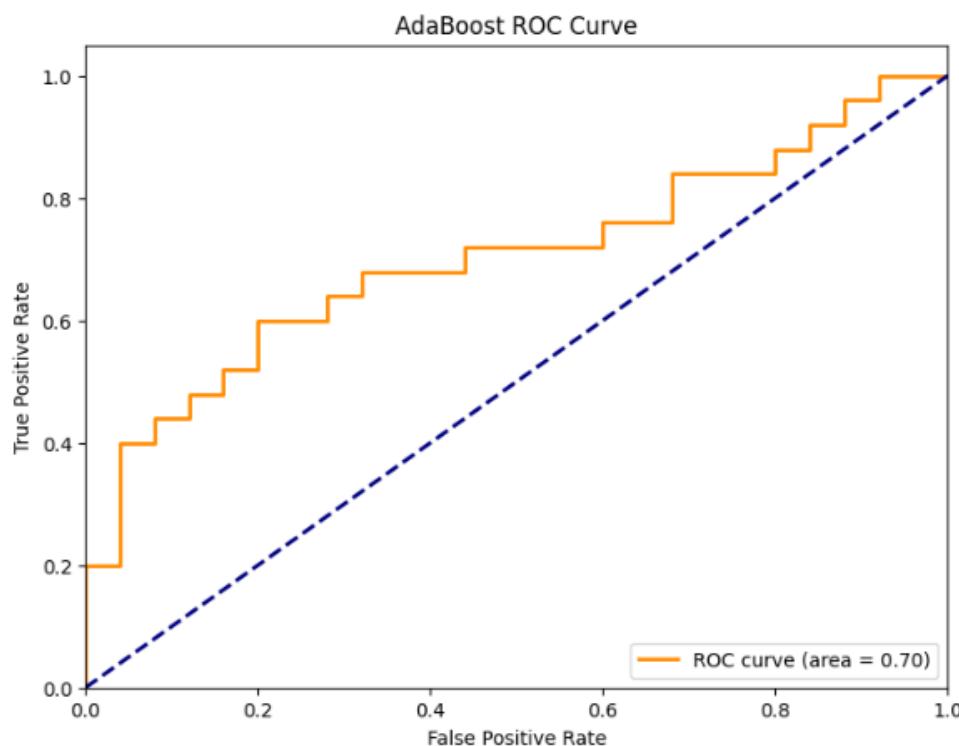
شكل ۱۱ SVM Algorithms , Precison , Recall :۱۱

```
MLP Confusion Matrix:  
[[16  9]  
 [16  9]]  
MLP Classification Report:  
          precision    recall   f1-score  support  
 -1       0.50      0.64      0.56      25  
  1       0.50      0.36      0.42      25  
  
    accuracy           0.50  
   macro avg       0.50      0.50      0.49      50  
weighted avg       0.50      0.50      0.49      50
```



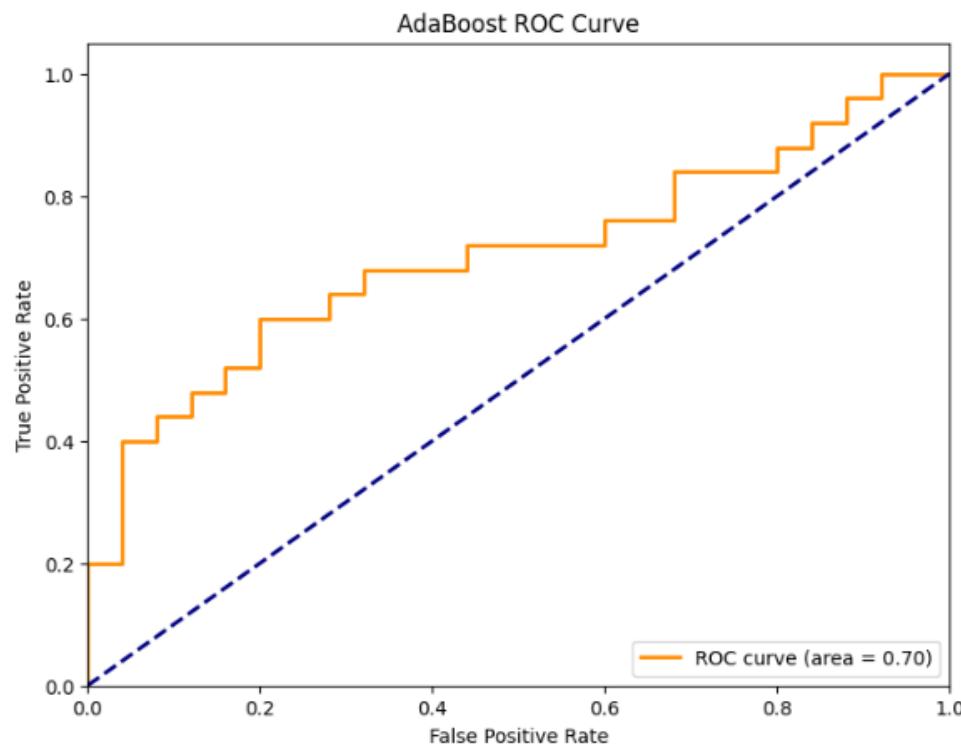
MLP Algorithms , Precison , Recall : ۱۲ شکل

```
AdaBoost Confusion Matrix:  
[[18  7]  
 [10 15]]  
AdaBoost Classification Report:  
 precision    recall    f1-score   support  
  
 -1      0.64      0.72      0.68      25  
 1       0.68      0.60      0.64      25  
  
   accuracy          0.66      50  
   macro avg       0.66      0.66      0.66      50  
 weighted avg     0.66      0.66      0.66      50
```



شكل ۱۳ : Precison ، Recall ، AdaBoost Algorithms

```
AdaBoost Confusion Matrix:  
[[18  7]  
 [10 15]]  
AdaBoost Classification Report:  
          precision    recall   f1-score  support  
 -1       0.64      0.72      0.68     25  
  1       0.68      0.60      0.64     25  
  
   accuracy           0.66      50  
 macro avg       0.66      0.66      0.66     50  
weighted avg     0.66      0.66      0.66     50
```

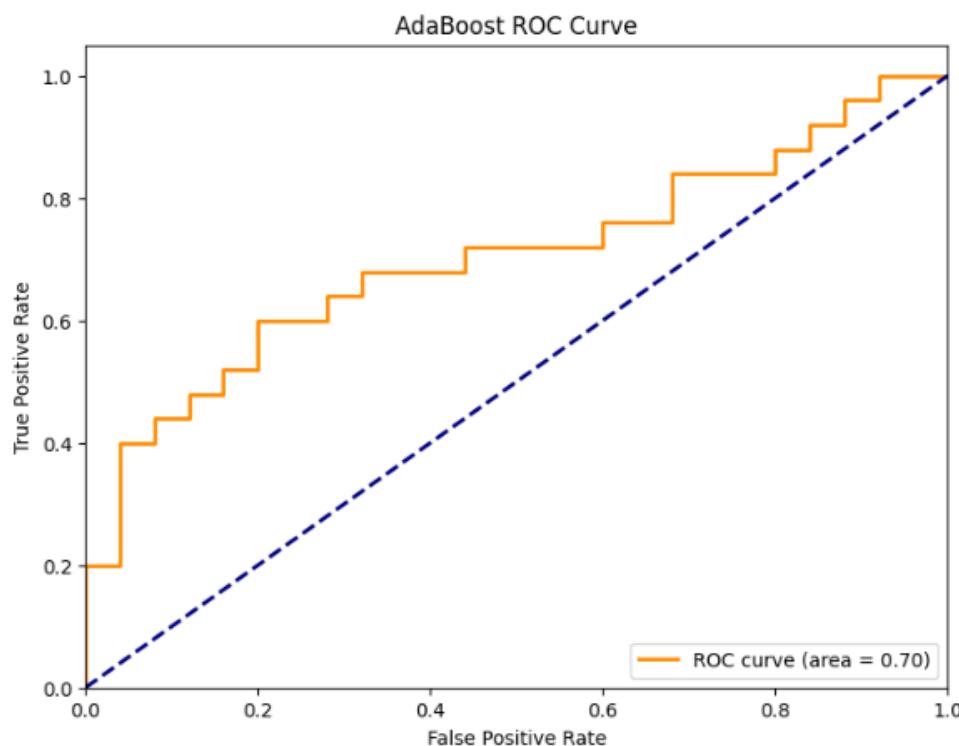


شكل ۱۴ Silhouette Score for Clustering :

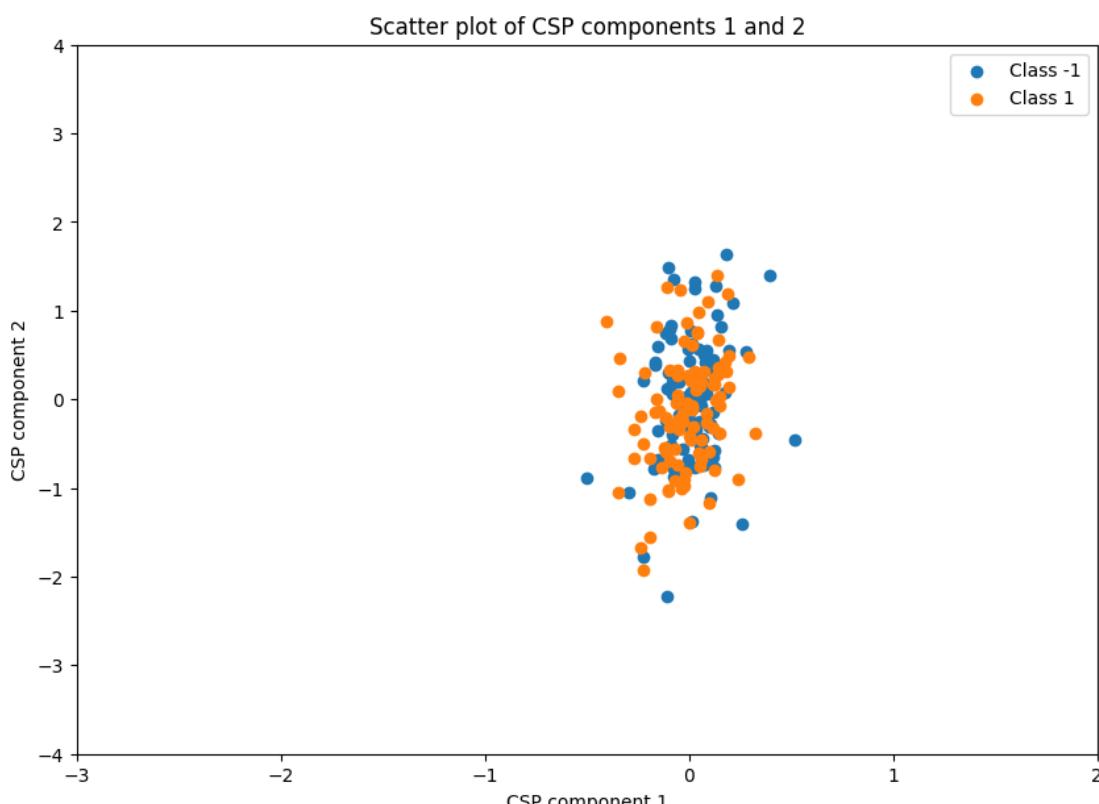
```

AdaBoost Confusion Matrix:
[[18  7]
 [10 15]]
AdaBoost Classification Report:
precision    recall    f1-score   support
 -1       0.64      0.72      0.68      25
  1       0.68      0.60      0.64      25

accuracy                           0.66      50
macro avg       0.66      0.66      0.66      50
weighted avg    0.66      0.66      0.66      50
    
```

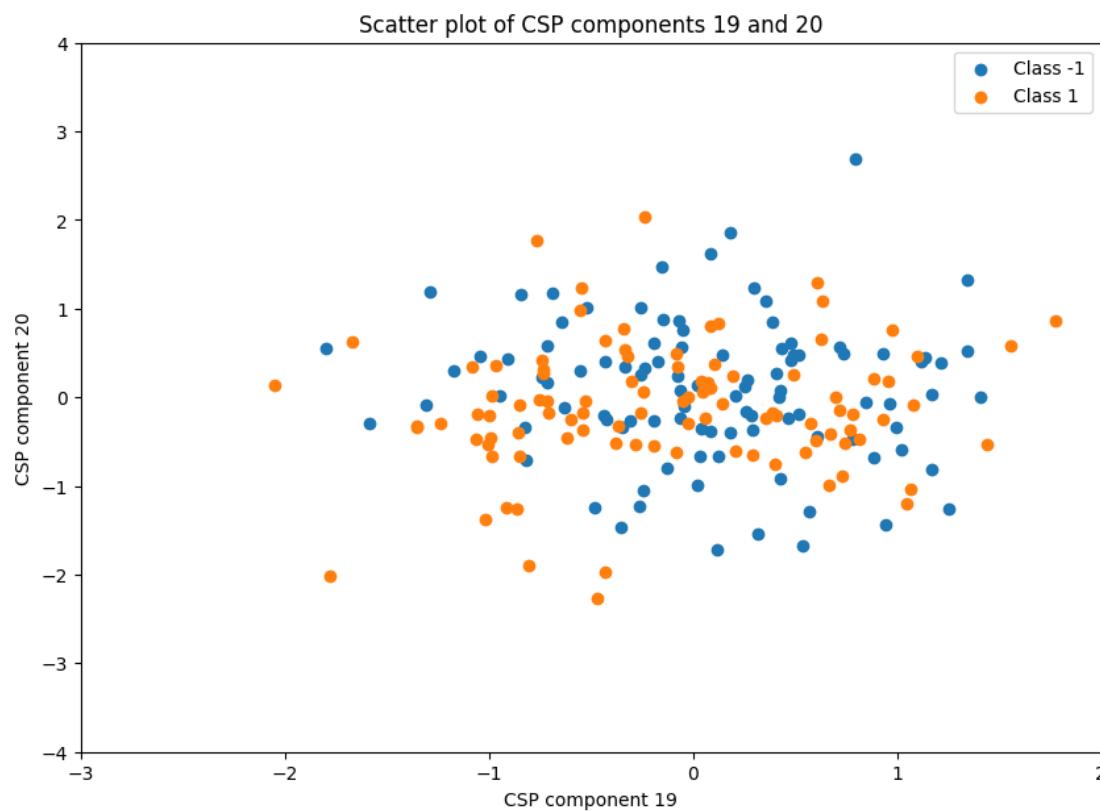


شكل : ۱۵ t-SNE KMeans :

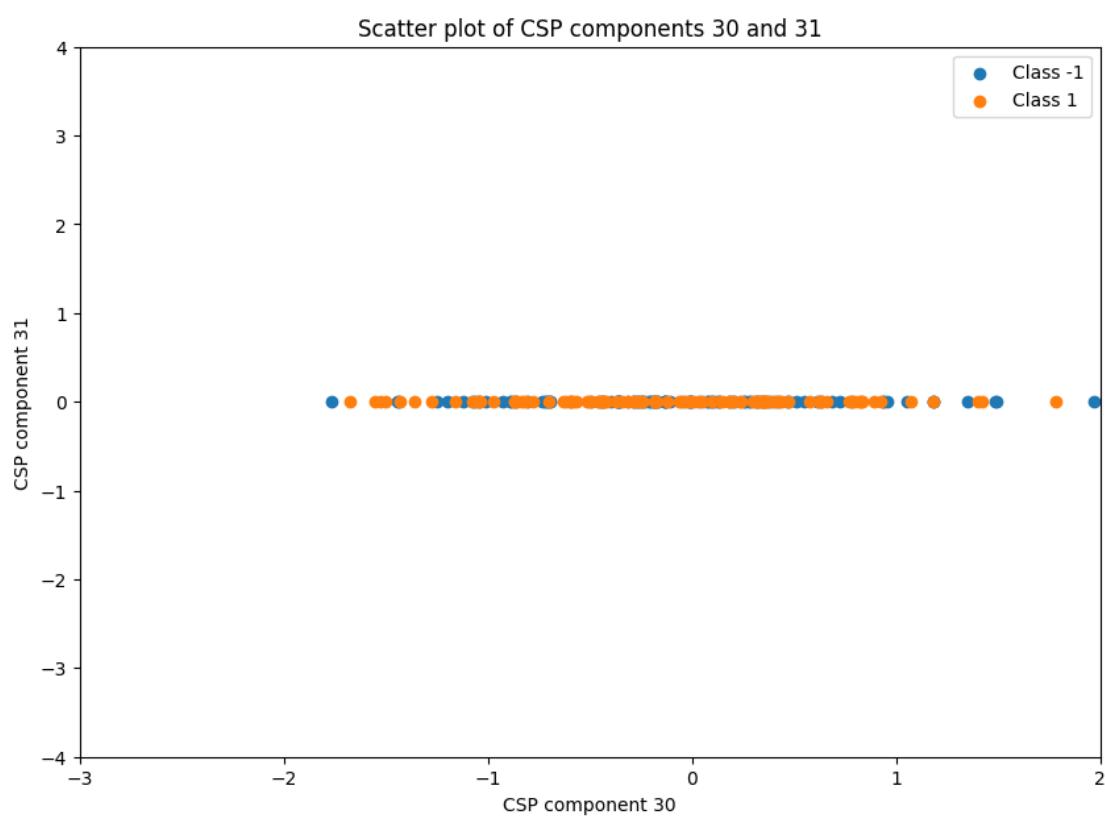


یک نمونه از استخراج

ویژگی مرتبط به وسیله CSP

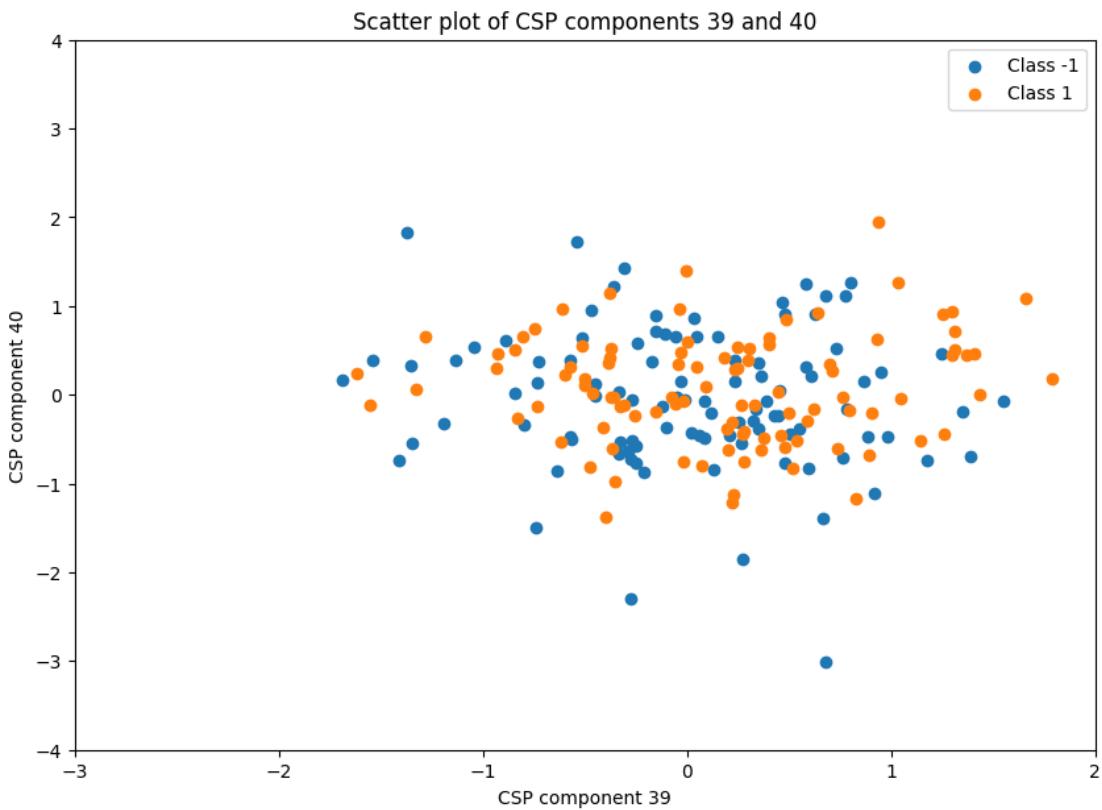


شکل ۱۶: یک نمونه از استخراج ویژگی مرتبط به وسیله CSP



یک نمونه از استخراج

ویژگی مرتبط به وسیله CSP



یک نمونه از استخراج

ویژگی مرتبط به وسیله CSP

```

def plot_eeg_signals(data, title, channels=[0, 1, 2, 3, 4], num_samples=1000):
    plt.figure(figsize=(15, 10))
    for i, channel in enumerate(channels):
        plt.subplot(len(channels), 1, i+1)
        plt.plot(data[:num_samples, channel])
        plt.title(f'{title} - Channel {channel+1}')
        plt.xlabel('Sample')
        plt.ylabel('Amplitude')
    plt.tight_layout()
    plt.show()

plot_eeg_signals(eeg_data_normalized, 'Raw EEG Signals')

def plot_filtered_vs_raw(raw_data, filtered_data, title, channels=[0, 1, 2, 3, 4], num_samples=1000):
    plt.figure(figsize=(15, 10))
    for i, channel in enumerate(channels):
        plt.subplot(len(channels), 1, i+1)
        plt.plot(raw_data[:num_samples, channel], label='Raw')
        plt.plot(filtered_data[:num_samples, channel], label='Filtered')
        plt.title(f'{title} - Channel {channel+1}')
        plt.xlabel('Sample')
        plt.ylabel('Amplitude')
        plt.legend()
    plt.tight_layout()
    plt.show()

plot_filtered_vs_raw(eeg_data_normalized, filtered_eeg_data, 'Filtered vs Raw EEG Signals')

plot_eeg_signals(car_filtered_epochs[0], 'CAR Filtered EEG Signals')

```

Plot function : ۱۷ شکل

```
X_train, X_test, y_train, y_test = train_test_split(flattened_features, event_classes, test_size=0.25, stratify=event_classes, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

classifiers = {
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'SVM': SVC(kernel='linear', probability=True),
    'MLP': MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42),
    'AdaBoost': AdaBoostClassifier(n_estimators=50, random_state=42)
}

for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_prob = clf.predict_proba(X_test)[:, 1] if hasattr(clf, "predict_proba") else None

    cm = confusion_matrix(y_test, y_pred)
    print(f'{name} Confusion Matrix:\n', cm)

    print(f'{name} classification Report:\n', classification_report(y_test, y_pred))

    if y_prob is not None:
        fpr, tpr, _ = roc_curve(y_test, y_prob)
        roc_auc = auc(fpr, tpr)

        plt.figure(figsize=(8, 6))
        plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
        plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
        plt.xlim([0.0, 1.0])
```

Classifiers and ClusterMethods :۱۸ شکل