

به نام خدا



یادگیری ماشین
پروژه پایانی
فاز دوم

۸۱۰۱۹۹۴۰۳	زهرا حجتی
۸۱۰۱۹۹۴۲۵	ایمان رسولی پرتو
۸۱۰۱۹۹۴۴۴	احسان شعفی

فهرست مطالب

۱	چکیده	۲
۲	پیش‌پردازش داده‌ها	۳
۳	مقدمه‌ای در مورد دیتاست	۱.۲
۳	خواندن دیتاست	۲.۲
۶	فیلتر کردن داده‌ها	۳.۲
۶	فیلتر میان‌گذر	۱.۳.۲
۸	فیلترهای فضایی	۲.۳.۲
۱۵	استخراج ویژگی	۳
۱۷	طبقه‌بندی	۴
۱۸	Logistic Regression	۱.۴
۲۰	SVM	۲.۴
۲۱	KNN	۳.۴
۲۲	رسم نمودار ROC و مقایسه طبقه‌بندها	۴.۴
۲۵	خوشه‌بندی	۵
۲۵	پیدا کردن تعداد خوشه بهینه	۱.۵
۲۵	پیدا کردن تعداد خوشه‌های بهینه برای K-means	۱.۱.۵
۲۶	پیدا کردن تعداد خوشه‌های بهینه برای EM	۲.۱.۵
۲۷	Clustering	۲.۵
۲۹	بررسی روند طی شده برای یک دیتاست دیگر	۶
۲۹	پیش‌پردازش	۱.۶
۳۵	استخراج ویژگی	۲.۶
۳۶	طبقه‌بندی	۳.۶
۴۰	خوشه‌بندی	۴.۶
۴۱	مقایسه نتایج با دیتاست قبلی	۵.۶

۱ چکیده

این پژوهش، پیاده سازی کامل طبقه بندی و خوش بندی سیگنال های EEG تولید شده از مغز هنگام تصویرسازی حرکتی می باشد. در ابتدا، به واکاوی کامل دیتابست می پردازیم. سیگنال های ما در طول زمان پیوسته هستند و برای این که ارتباط بین داده ها در طول زمان و لیبل تست های متاظر با آن را برقرار کنیم، نیاز داریم تا داده های اصلی پنجره پنجه یا به اصطلاح epoch کنیم. سیگنال های مغزی، ذاتا دارای نویز بسیار زیادی بوده و آموزش مدل های یادگیری ماشین روی داده های خام آن نتایج مطلوبی ندارد. در نتیجه برای رسیدن به دقت مطلوب، نیاز داریم تا پیش پردازش هایی روی سیگنال های خام انجام دهیم. بدین منظور در گام اول یک فیلتر میان گذر روی داده ها اعمال کرده و تا فرکانس هایی که عدمه تغییرات در آن ها هنگام تصویرسازی حرکتی اتفاق می افتد جدا کنیم. پس از آن، فیلتر لاپلاسین را اعمال کرده تا از نویز داده ها بکاهیم. در گام بعدی تکنیک های PCA و ICA را اعمال کرده تا مولفه های اصلی و مستقل را از هم تفکیک کنیم. پس از انجام مراحل مذکور، الگوریتم CSP را اعمال می کنیم تا داده های خام ما تفکیک پذیری بهتری داشته باشند تا هنگام طبقه بندی به نتایج بهتری برسیم. در نهایت مدل های مختلف طبقه بندی مانند Logistic، KNN، SVM، MLP، XGBoost و Regression استفاده کرده و نتایج را با هم مقایسه می کنیم. برای خوش بندی نیز از دو روش EM و K-Means استفاده کرده و نتایج را مقایسه می کنیم.

۲ پیش‌پردازش داده‌ها

۱.۲ مقدمه‌ای در مورد دیتاست

دیتاست داده شده شامل سیگنال‌های EEG نمونه‌برداری شده توسط الکتروودها در ۵۹ ناحیه از مغز می‌باشد. هر کدام از این نواحی موسوم به یک کانال می‌باشد.
دیتاست شامل سه بخش کلی می‌باشد:

(۱) cnt : سیگنال‌های EEG پیوسته در طول زمان برای همه کانال‌ها

(۲) mrk : این فیلد خود شامل دو بخش برچسب و موقعیت می‌باشد. در واقع در این بخش ۲۰۰ آزمایش انجام شده که مشخص می‌کند هر تست چند ثانیه طول کشیده (توسط متغیر pos) و آن حرکت چه بوده است.

(۳) nfo : این بخش شامل اطلاعات اضافی نظیر نرخ نمونه‌برداری، موقعیت الکتروودها در پروجکشن ۲ بعدی، برچسب کانال‌ها و اسمی کلامس‌ها می‌باشد.

۲.۲ خواندن دیتاست

برای خواندن دیتاست از کتابخانه `scipy` استفاده می‌کنیم. در این بخش اطلاعات مورد نیاز از فایل متلب را ذخیره می‌کنیم. همچنان مقادیر ولتاژ را به μV تبدیل می‌کنیم.

Load Dataset

```
data = scipy.io.loadmat('./BCICIV_calib_ds1a.mat',
                       struct_as_record=True)
sfreq = data['nfo']['fs'][0][0][0]
EEGdataa = 0.1 * np.double(data['cnt'].T)
nchannels, nsamples = EEGdataa.shape

chan_names = [s[0] for s in data['nfo']['clab'][0][0][0]]

event_onsets = data['mrk'][0][0][0]
event_codes = data['mrk'][0][0][1]

labels = np.zeros((1, nsamples), int)
labels[0, event_onsets] = event_codes

cl_lab = [s[0] for s in data['nfo']['classes'][0][0][0]]
c11 = cl_lab[0]
c12 = cl_lab[1]

xpos = data['nfo']['xpos']
ypos = data['nfo']['ypos']
```

در کد فوق ابتدا داده از دیتاست مربوطه خوانده شده و سپس پارامترهای مختلف از آن استخراج می‌شود. برای مثال نرخ نمونهبرداری تحت عنوان sfreq از فیلد اطلاعات اضافی برداشته می‌شود. به همین ترتیب تمامی اطلاعات مورد نیاز از دیتاست جمع آوری می‌شود. برای کار با دیتاست از کتابخانه mne استفاده می‌کنیم. این کتابخانه ابزار قدرتمندی برای تحلیل سیگنال‌های EEG و EMG می‌باشد.

برای جمع آوری و مشخص کردن داده‌ها، همانطور که در بخش ۱.۲ گفته شد، ۲۰۰ تست انجام شده که در هر تست از ۵۹ کانال سیگنال نمونهبرداری شده است. مدت زمان این تست‌ها توسط فیلد pos مشخص می‌شود. با مطالعه دیتاست می‌فهمیم که هر تست به اندازه ۸۰۰ نمونه از هر کانال داده دارد. با توجه به اینکه نرخ نمونهبرداری ۱۰۰ Hz است، می‌توان گفت مدت زمان هر تست ۸ ثانیه می‌باشد. در واقع طول پنجره زمانی که توسط pos مشخص شده، ۸ ثانیه می‌باشد.

در نتیجه نیاز داریم تا پنجره‌های زمانی را مشخص کرده و داده‌ها را با توجه به آن استخراج کنیم. اینکار که در دستور Epoch نام دارد، توسط کتابخانه mne هندل می‌شود. همچنین در میانه برخی تست‌ها عملیات کالیبراسیون انجام می‌شود که جزو داده‌ها نمی‌باشد و باید حذف شود که این مورد نیز توسط کتابخانه رفع می‌شود.

Extract Data

```
dataa = np.concatenate([left, foot])

event_id = dict(left = -1, right = 1)

# Create an event matrix: events with
# alternating event codes

eventLength = Y.shape[0]
ev = [i * sfreq * 3 for i in range(eventLength)]

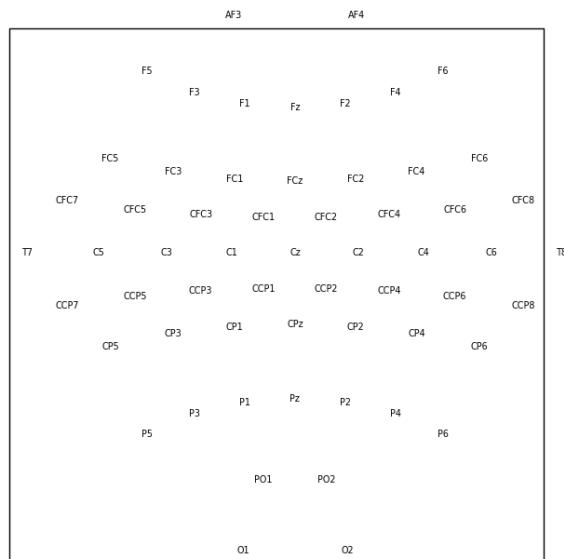
events = np.column_stack((np.array(ev, dtype = int),
                           np.zeros(eventLength, dtype = int),
                           np.array(Y, dtype = int)))

tmin = 0.5

# Create the :class:`mne.EpochsArray` object
epochs = mne.EpochsArray(dataa, info, events, tmin,
                         event_id)
```

در کد بالا ابتدا کلاس‌ها توسط دیکشنری کدگذاری می‌شوند و سپس epoch انجام می‌شود. با خواندن دیتابست توسط کتابخانه mne تمامی event‌ها تشخیص داده می‌شوند. در نهایت ۲۰۰ داده داریم که بعد هر داده 59×800 می‌باشد. با توجه به بعد بالای داده‌ها، نیاز به کاهش ابعاد داریم.

همچنین در این بخش کد با خواندن دیتاست نمایی از موقعیت الکتروودها روی مغز را خواهیم داشت:



شکل ۱.۲: نمایی از موقعیت الکتروودها (کانال‌ها) روی مغز

۳.۲ فیلتر کردن داده‌ها

با توجه به مواردی که درباره اهمیت پاکسازی و حذف نویز سیگنال‌های EEG در فاز نخست مطرح شد، در دو مرحله و با چند رویکرد مختلف داده‌ها را فیلتر می‌کنیم.

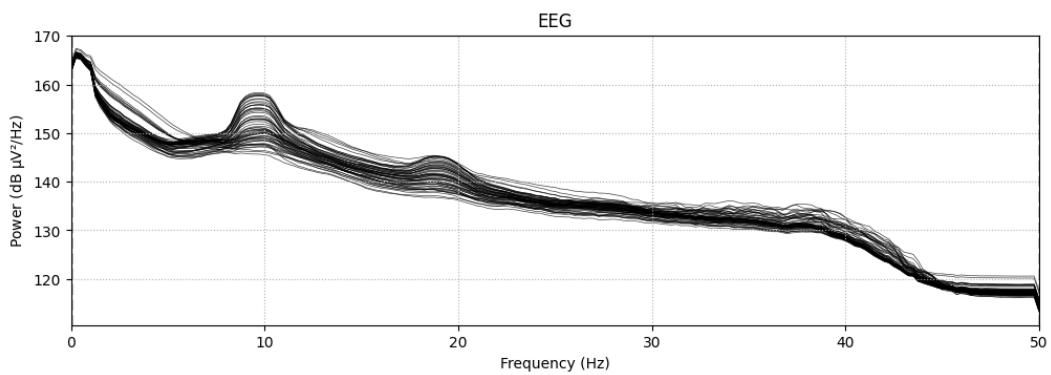
۱.۳.۲ فیلتر میان‌گذر

با توجه به مطالبی که در مورد ناحیه فرکانسی این سیگنال‌های در فاز نخست بحث شد، به کمک یک فیلتر میان‌گذر در بازه فرکانسی $30 - 8\text{ Hz}$ فرکانسی این سیگنال‌ها را فیلتر می‌کنیم.

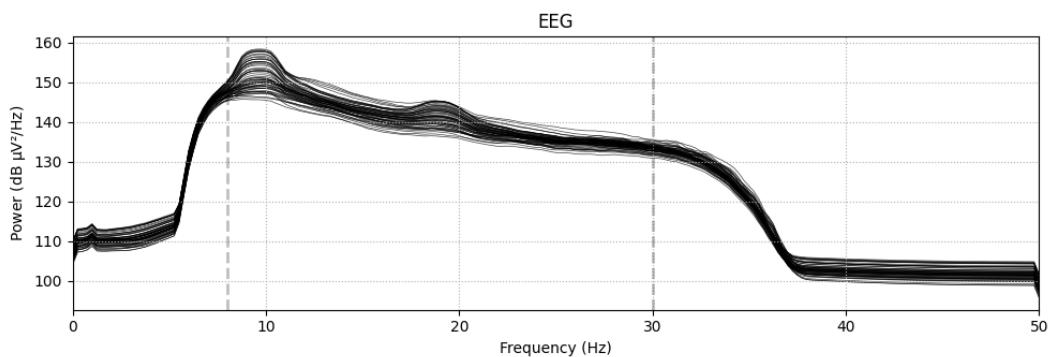
BandPass Filter

```
# Bandpass Filter
epochs.filter(l_freq=8, h_freq=30)
```

پس از epoch کردن، داده‌ها در متغیر epochs ذخیره می‌شود که در کتابخانه mne موجود است. این کتابخانه فیلتر میان‌گذر را درون خود دارد و از آن برای فیلتر کردن سیگنال‌ها استفاده می‌کنیم.



شکل ۲.۲: سیگنال‌ها قبل از اعمال فیلتر میان‌گذر



شکل ۳.۲: سیگنال‌ها پس از اعمال فیلتر میان‌گذر

با توجه به نمودارها مشاهده می‌کنیم که با تقریب خوبی باند فرکانسی $8 - 30\text{ Hz}$ از سیگنال‌ها نگه داشته شده و مابقی فیلتر شده است.

۲.۳.۲ فیلترهای فضایی

فیلترهای فضایی به منظور پاکسازی بیشتر داده‌ها مورد استفاده قرار می‌گیرند. پس از اعمال فیلتر میان‌گذر روی سیگنال‌های فیلتر شده فیلترهای فضایی را اعمال می‌کنیم. این فیلترها عبارت‌اند از:

فیلتر Laplacian

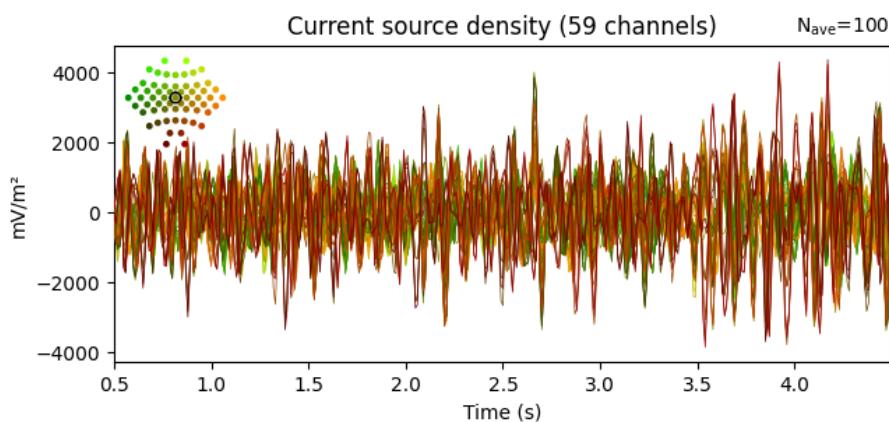
برای اعمال فیلتر لابلسین از قطعه کد زیر استفاده می‌کنیم:

Laplacian Filter

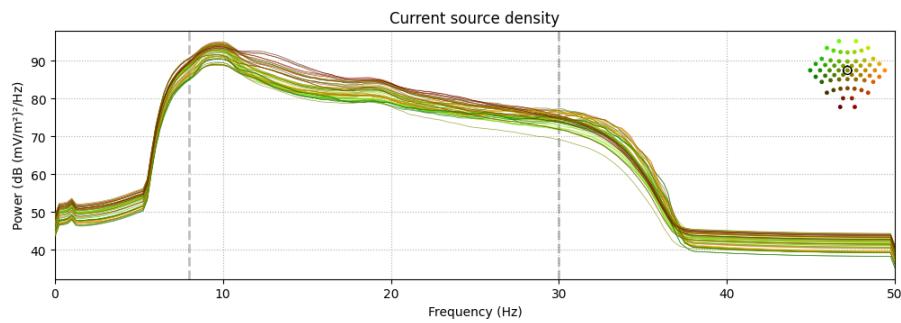
```
zpos = np.zeros_like(xpos) + 0.001
ch_pos = dict(zip(chan_names, np.column_stack([xpos, ypos,
                                              zpos])))
print(ch_pos)
montage = mne.channels.make_dig_montage(ch_pos,
                                          coord_frame='head')
epochs.set_montage(montage)
laplacian_epochs =
    mne.preprocessing.compute_current_source_density(epochs,
                                                    sphere='auto', lambda2=1e-05, stiffness=4, n_legendre_terms=50,
                                                    copy=True)

laplacian_epochs['left'].average().plot()
laplacian_epochs['right'].average().plot()

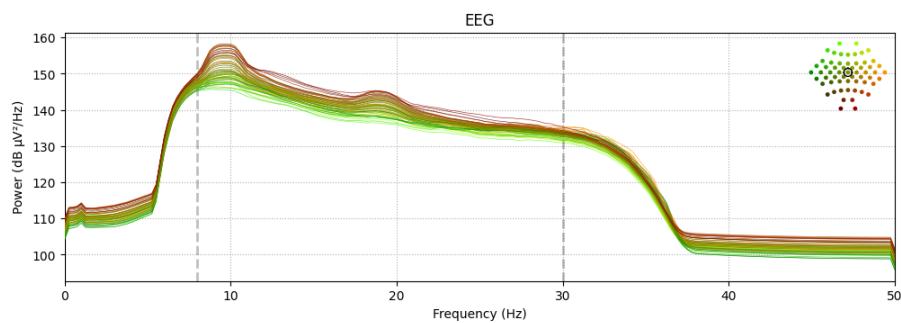
laplacian_epochs.compute_psd().plot()
```



شکل ۴.۲: نمودار سیگنال‌ها پس از اعمال فیلتر لابلسین (برای کلاس left)



شکل ۵.۲: نمودار سیگنال‌ها در طیف فرکانسی پیش از اعمال فیلتر



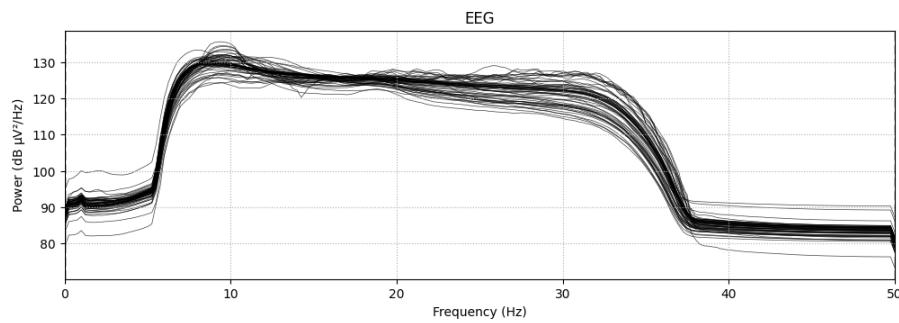
شکل ۶.۲: نمودار سیگنال‌ها در طیف فرکانسی پس از اعمال فیلتر

مشاهده می‌کنیم پس از اعمال فیلتر لاپلاسین نویز سیگنال‌ها در بازه فرکانسی مورد نظر کمتر شده و پیک سیگنال‌ها قابل رویت است.

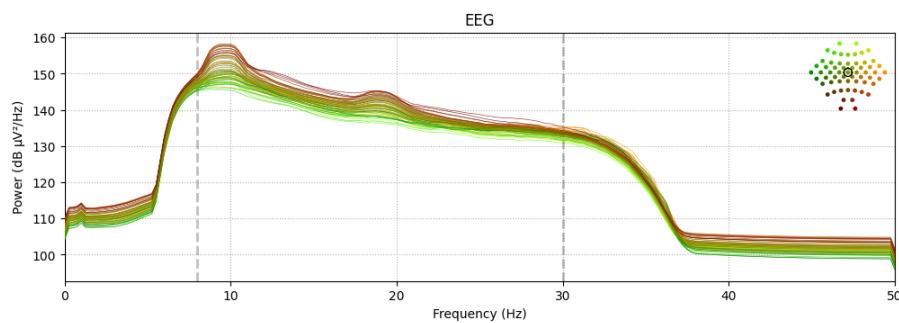
استفاده از ICA

ICA Filtering

```
X = epochs.get_data(copy=False)
ICA = mne.decoding.UnsupervisedSpatialFilter(FastICA(59,
    whiten="unit-variance"), average=False)
ica_data = ICA.fit_transform(X)
epochs_ica = mne.EpochsArray(ica_data, mne.create_info(59,
    epochs.info["sfreq"], ch_types="eeg"), events, tmin, event_id)
```



شکل ۷.۲: نمودار سیگنال‌ها در طیف فرکانسی پیش از اعمال فیلتر



شکل ۸.۲: نمودار سیگنال‌ها در طیف فرکانسی پس از اعمال فیلتر

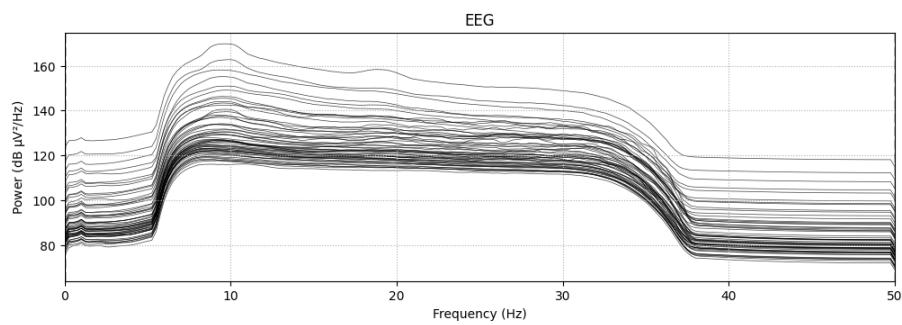
فیلتر ICA نیز نویز سیگنال‌ها را در بازه فرکانسی کاهش داده و باعث می‌شود پیک‌های سیگنال‌ها قابل شناسایی باشند.

استفاده از PCA

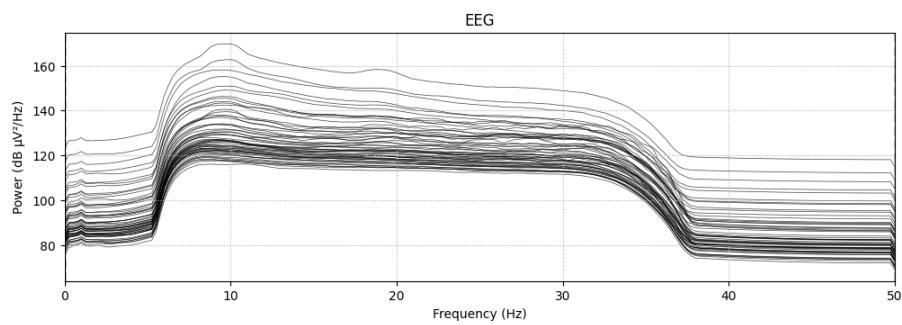
PCA FilteringING

```
pca = mne.decoding.UnsupervisedSpatialFilter(PCA(59),
                                              average=False)
pca_data = pca.fit_transform(X)

epochs = mne.EpochsArray(pca_data, mne.create_info(59,
                                                    epochs.info["sfreq"],
                                                    ch_types="eeg"),
                        events, tmin, event_id)
```



شکل ۹.۲: نمودار سیگنال ها در طیف فرکانسی پیش از اعمال فیلتر



شکل ۱۰.۲: نمودار سیگنال ها در طیف فرکانسی پس از اعمال فیلتر

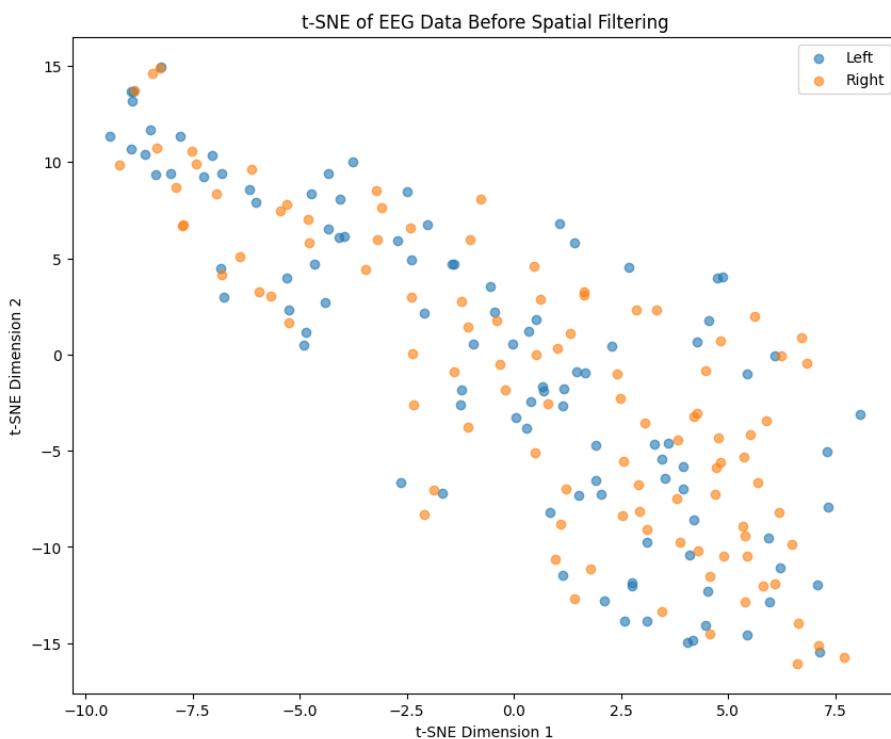
همانطور که مشاهده می کنیم، تکنیک PCA در کاهش نویز سیگنال ها چندان موثر نیست.

تصویرسازی داده‌ها

در این قسمت از کتابخانه t-SNE برای تصویرسازی داده‌ها قبل و بعد از اعمال فیلترهای فضایی استفاده می‌کنیم.

Visualizing raw data

```
n_samples, n_channels, n_times = dataa.shape  
data_2d = dataa.reshape(n_samples, -1)  
  
tsne = TSNE(n_components=2, perplexity=30, random_state=42)  
data_tsne = tsne.fit_transform(data_2d)  
  
plt.figure(figsize=(10, 8))  
plt.scatter(data_tsne[Y == -1, 0], data_tsne[Y == -1, 1],  
            label='Left', alpha=0.6)  
plt.scatter(data_tsne[Y == 1, 0], data_tsne[Y == 1, 1],  
            label='Right', alpha=0.6)
```



شکل ۱۱.۲: داده‌ها قبل از اعمال فیلترهای فضایی و epoching

پس از flat کردن داده‌های خروجی فیلترها آنها را برای تکنیک‌های استفاده شده رسم می‌کنیم.

Visualizing filtered and epoched data

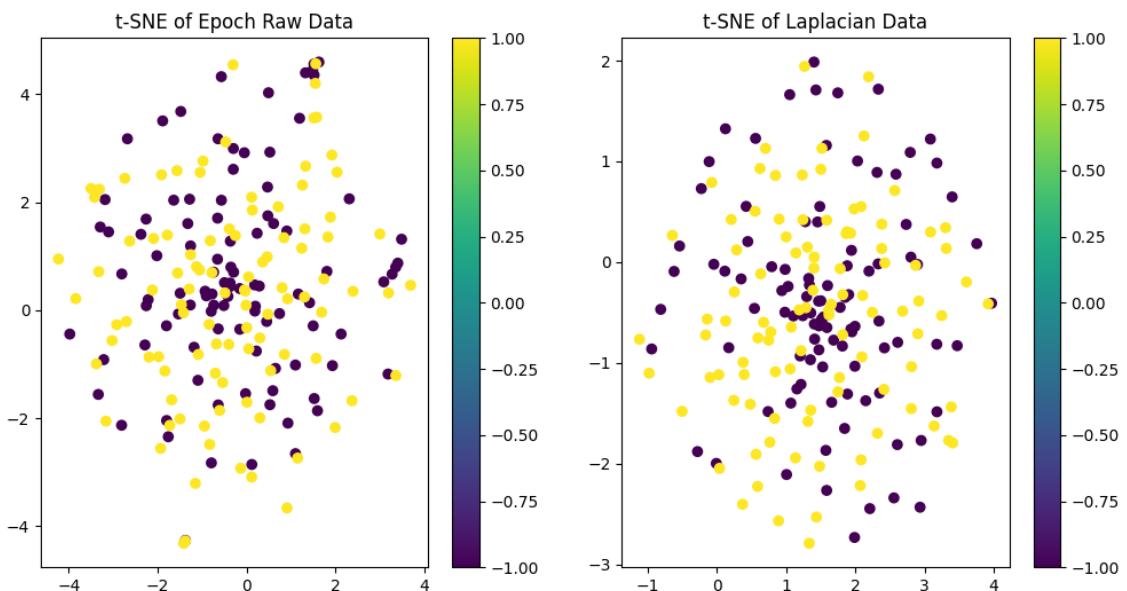
```
tsne = TSNE(n_components=2, random_state=0)

epochs_data = epochs.get_data(copy=False)
epochs_flattened = epochs_data.reshape(epochs_data.shape[0], -1)

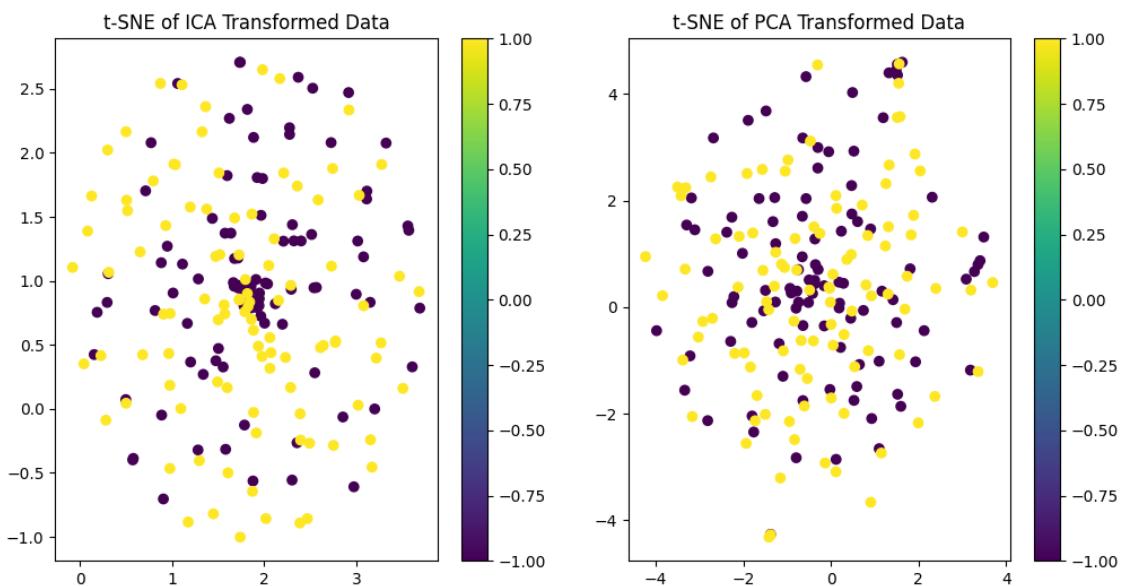
laplacian_data = laplacian_epochs.get_data(copy=False)
laplacian_flattened = laplacian_data.reshape(laplacian_data.shape[0], -1)

ica_flattened = ica_data.reshape(ica_data.shape[0], -1)
pca_flattened = pca_data.reshape(pca_data.shape[0], -1)

ica_tsne = tsne.fit_transform(ica_flattened)
pca_tsne = tsne.fit_transform(pca_flattened)
```



شکل ۱۲.۲: تصویرسازی داده‌ها در حالت فقط epoching و فیلتر لابلائسین



شکل ۱۳.۲: تصویرسازی داده‌ها در حالت تکنیک PCA و ICA

با توجه به شکل فوق پراکندگی داده‌ها در PCA بیشتر است، چرا که تمرکز این تکنیک بر بیشینه‌کردن واریانس داده‌هاست.
مشاهده می‌کنیم که با اعمال فیلتر داده‌ها کمی در هم می‌روند. به کمک تکنیک استخراج ویژگی، این موضوع را می‌توان رفع کرد.

۳ استخراج ویژگی

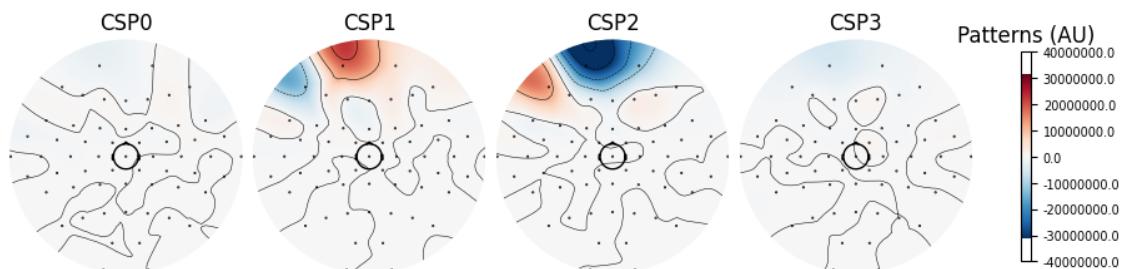
با توجه به بعد بالای داده‌ها برای حل مشکل نحسی ابعاد، از روش‌های کاهش بعد استفاده می‌کنیم. در ابتدا از الگوریتم CSP استفاده می‌کنیم. تمرکز این الگوریتم در بهبود واریانس میان کلاسی می‌باشد. این الگوریتم به طور خودکار کانال‌هایی که تاثیر بیشتری در طبقه‌بندی دارند را استخراج می‌کند.

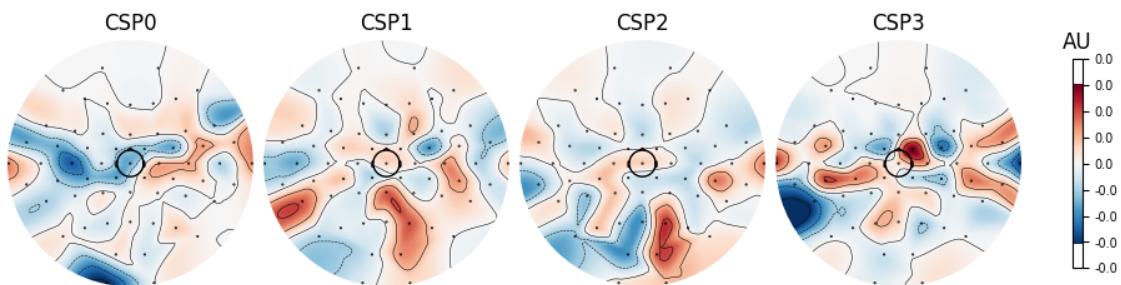
CSP Algorithm

```
csp = mne.decoding.CSP(n_components=4, reg='ledoit_wolf',
log=True, norm_trace=False)

class_balance = np.mean(labels == labels[0])
class_balance = max(class_balance, 1.0 - class_balance)

csp.fit_transform(epochs_data_train, labels)
csp.plot_patterns(epochs.info, ch_type="eeg",
units="Patterns (AU)", size=1.5)
csp.plot_filters(epochs.info, scalings=1e-8, size=1.5)
```





شکل ۱.۳: نمودار کانال‌های فعال برای الگوهای مختلف

با توجه به شکل بالا مشاهده می‌کنیم در هر الگو چه نواحی از مغز در تشخیص کدام حرکت (کلاس‌ها) کاربرد دارند. این موضوع به درک و استخراج ویژگی‌ها کمک می‌کند. در هر الگو کانال‌هایی که تعیین‌کننده نوع حرکت و کلاس هستند مشخص شده‌اند. شدت رنگ‌ها بیانگر تعیین‌کننده بودن کانال‌هاست و بیانگر اینکه این کانال به عنوان یک feature مهم قابل استفاده است. با اعمال الگوریتم CSP به دقت ۷۳.۵٪ می‌رسیم. لازم به ذکر از داده‌های خروجی PCA برای این تکنیک استفاده شده است.

Feature Extraction

```
# Assemble a classifier
lda = LinearDiscriminantAnalysis()
clf = Pipeline([("CSP", csp), ("LDA", lda)])
scores = cross_val_score(clf, epochs_data_train,
                         labels, cv=cv_split, n_jobs=None)
```

در کد بالا روش LDA نیز برای استخراج ویژگی بررسی شده. روش‌های دیگر کاهش بعد مثل PCA در بخش قبل انجام شد. در مقایسه باید گفت الگوریتم CSP یک الگوریتم Supervised است و در کاهش بعد و استخراج ویژگی بهتر عمل می‌کند.

۴ طبقه‌بندی

پس از کاهش بعد و استخراج ویژگی‌ها، به طبقه‌بندی داده‌ها می‌پردازیم.

Partitioning Data

```
montage = mne.channels.make_dig_montage(ch_pos,
    coord_frame='head')
epochs.set_montage(montage)

epochs_train = epochs.copy().crop(tmin=1.0, tmax=2.0)
epochs_data_train = epochs_train.get_data(copy=False)

labels = epochs.events[:, -1]

cv = ShuffleSplit(10, test_size=0.25, random_state=42)
csp = mne.decoding.CSP(n_components=4, reg=None, log=True,
    norm_trace=False)
X_csp = csp.fit_transform(epochs_data_train, labels)

labels_binary = (labels > 0).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X_csp,
    labels_binary, test_size=0.25, random_state=2,
    stratify=labels_binary)
```

ابتدا موقعیت کانال‌های EEG تنظیم می‌شود، سپس داده‌ها به دو دسته آموزش و تست با نسبت 25% و 75% تقسیم می‌شود. به منظور اینکه نسبت دو کلاس در هر دو دسته داده‌ها به یک میزان باشد، فیلد برابر لیل‌هایی که به باینری تبدیل شده‌اند قرار داده می‌شود.

ویژگی‌ها با استفاده از روش CSP از داده‌ها استخراج می‌شوند.

برای طبقه‌بندی تابعی به نام evaluate model تعریف می‌کنیم. این تابع مدل را روی داده‌های آموزش، آموزش می‌دهد و با داده‌های تست آزمایش می‌کند، سپس ماتریس آشفتگی هر مدل را رسم کرده و دقت طبقه‌بند را محاسبه می‌کند. در انتهای FPR و TPR محاسبه می‌شود و منحنی ROC مدل به همراه معیار AUC بدست می‌آید.

Model Evaluate

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title(f'{model_name} - Confusion Matrix')
plt.show()

fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)
roc_curves.append((fpr, tpr, roc_auc, model_name))

accuracy = np.mean(y_pred == y_test)
```

پس از تعاریف فوق و تعریف مدل‌ها طبقه‌بندها روی داده‌ها آموزش داده شده و تست می‌شوند.

Logistic Regression ۱.۴

Logistic Regression

```
clf = LogisticRegression(solver="liblinear")
scaler = StandardScaler()

eeg_data = epochs.get_data(copy=False).reshape(len(labels), -1)

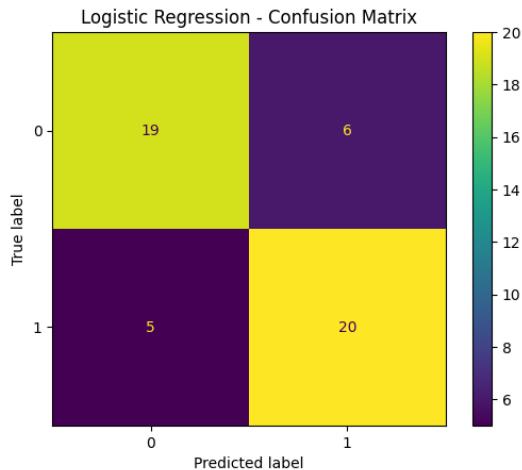
model = mne.decoding.LinearModel(clf)
X = scaler.fit_transform(eeg_data)
model.fit(X, labels)

for name, coef in ((patterns_, model.patterns_),
                    (filters_, model.filters_)):
    coef = scaler.inverse_transform([coef])[0]
    coef = coef.reshape(len(epochs.ch_names), -1)

    evoked = mne.EvokedArray(coef, epochs.info, tmin=epochs.tmin)
    fig = evoked.plot_topomap(size=1.5)
    fig.suptitle(f"EEG {name}")
```

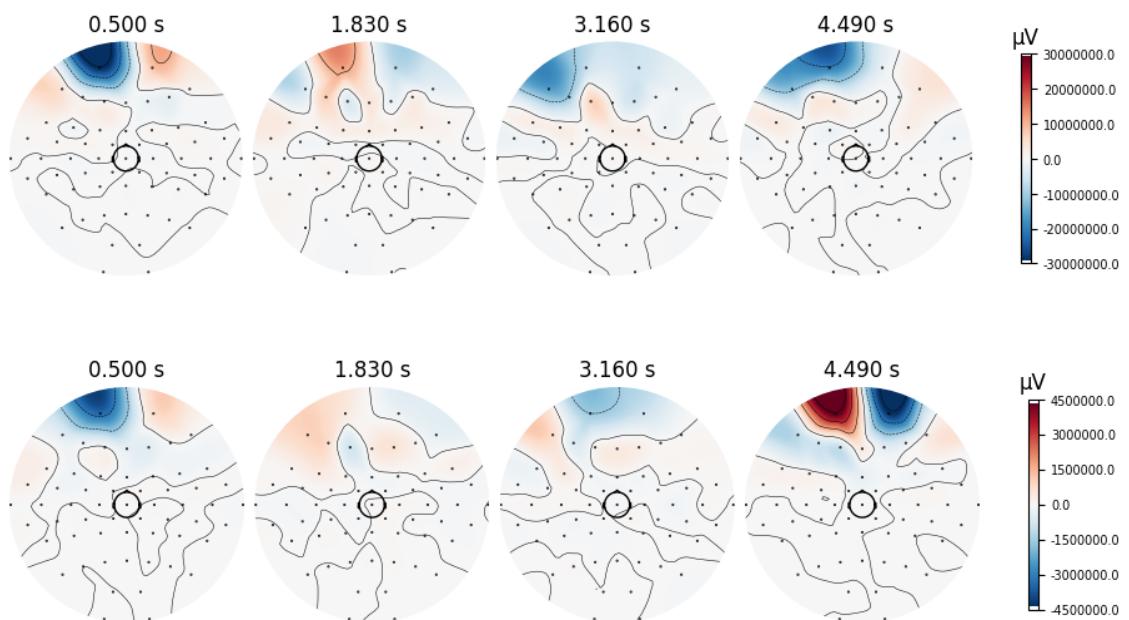
پس از تعریف مدل و اسکیل کردن داده‌ها، الگوهای مختلف برای این مدل رسم می‌شود.

با اعمال این طبقه‌بند ماتریس آشتفتگی بصورت زیر بدست می‌آید:



شکل ۱.۴: ماتریس آشتفتگی برای Logistic Regression

دقت این طبقه‌بند در حدود 78% بدست می‌آید که دقت قابل قبولی می‌باشد. همچنین با اعمال این طبقه‌بند می‌توان کانال‌های انتخاب شده و نقش آنها در طبقه‌بندی را مشاهده کرد.



شکل ۲.۴: نمودار نواحی مغز در تست‌های مختلف و بررسی نقش کانال‌های انتخاب شده

مشاهده می‌کنیم نواحی جدایی پذیر هستند و کانال‌های نزدیک بهم حرکت واحدی را پیش‌بینی می‌کنند. این امر نتیجه انتخاب بهینه از فیلترها می‌باشد.

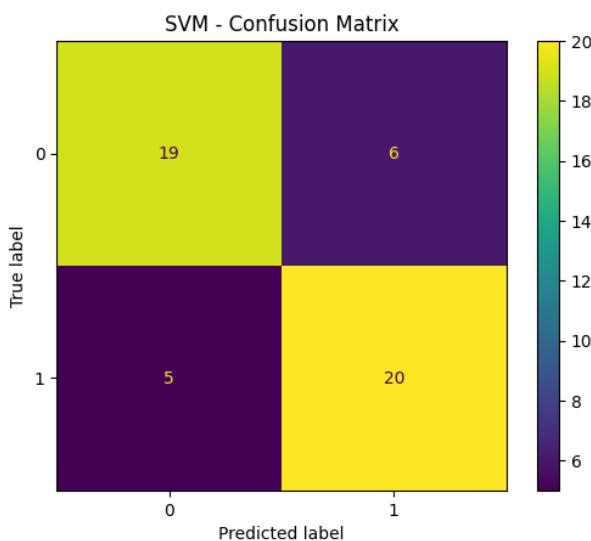
SVM ۲.۴

SVM

```
clf = make_pipeline(mne.decoding.Scaler(epochs.info),
    mne.decoding.Vectorizer(),
    mne.decoding.LinearModel(LinearSVC(C=0.1)))
clf.fit(epochs.get_data(), labels)

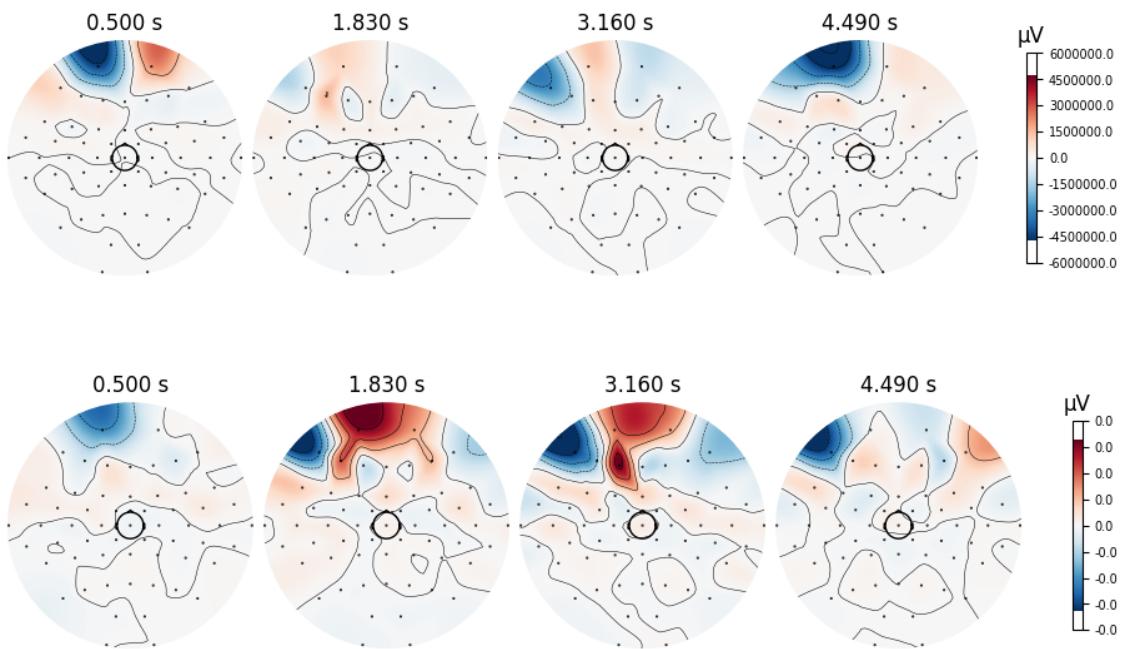
patterns = mne.decoding.get_coef(clf, 'patterns_',
    inverse_transform=True)
evoked_patterns = mne.EvokedArray(patterns, epochs.info,
    tmin=epochs.tmin)
fig_patterns = evoked_patterns.plot_topomap(size=1.5)
fig_patterns.suptitle("EEG Patterns")

filters = mne.decoding.get_coef(clf, 'filters_',
    inverse_transform=True)
evoked_filters = mne.EvokedArray(filters, epochs.info,
    tmin=epochs.tmin)
fig_filters = evoked_filters.plot_topomap(size=1.5)
fig_filters.suptitle("EEG Filters")
```



شکل ۴.۳: ماتریس آشفتگی برای SVM

دقت این طبقه‌بند در حدود 78% بdst می‌آید.



شکل ۴.۴: نمودار نواحی مغز در تست‌های مختلف و بررسی نقش کانال‌های انتخاب شده

مشابه Logistic Regression در این طبقه‌بندی هم کانال‌های نزدیک بهم حرکت یکسانی را پیش‌بینی می‌کنند که برای طبقه‌بندی مناسب است.

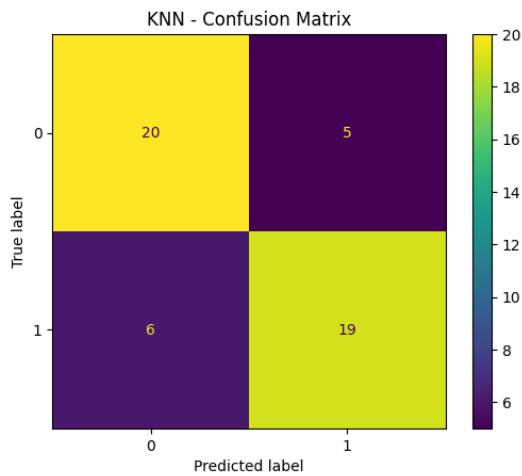
KNN ۳.۴

KNN

```
clf = make_pipeline(mne.decoding.Scaler(epochs.info),
                     mne.decoding.Vectorizer(),
                     KNeighborsClassifier(n_neighbors=5))
clf.fit(epochs.get_data(), labels)

predictions = clf.predict(epochs.get_data())
```

در این بخش یک مدل KNN تعریف می‌شود، مدل با استفاده از داده‌های EEG آموزش داده می‌شود و سپس داده‌های تست را پیش‌بینی می‌کند.

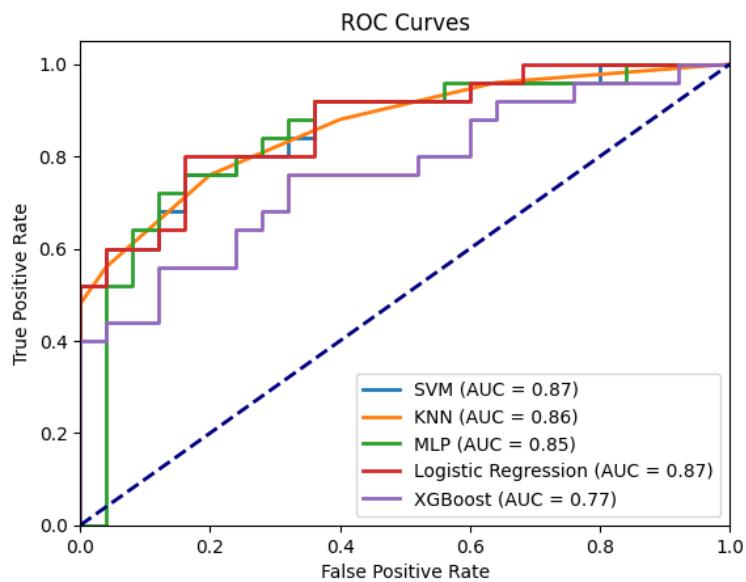


شکل ۵.۴: ماتریس آشتفتگی برای KNN

دقت این طبقه‌بند 78% بدست می‌آید.

۴.۴ رسم نمودار ROC و مقایسه طبقه‌بندها

با طبقه‌بندی‌های مختلف و اضافه کردن یک شبکه MLP و طبقه‌بند XG Boost برای مقایسه ملموس‌تر نمودار ROC به شکل زیر خواهد بود:



شکل ۶.۴: نمودار ROC

مشاهده می کنیم طبقه بندهای SVM و Logistic Regression عملکرد بهتری نسبت به باقی طبقه بندها دارد. همینطور معیار AUC برای این طبقه بندهای نسبت به باقی بیشتر است.

ROC Curve

```
models = [
    (SVC(kernel='linear', probability=True), "SVM"),
    (KNeighborsClassifier(n_neighbors=5), "KNN"),
    (MLPClassifier(hidden_layer_sizes=(100,), max_iter=300,
        random_state=1), "MLP"),
    (LogisticRegression(solver='lbfgs', max_iter=1000),
        "Logistic Regression"),
    (XGBClassifier(use_label_encoder=False,
        eval_metric='logloss'), "XGBoost")
]

roc_curves = []

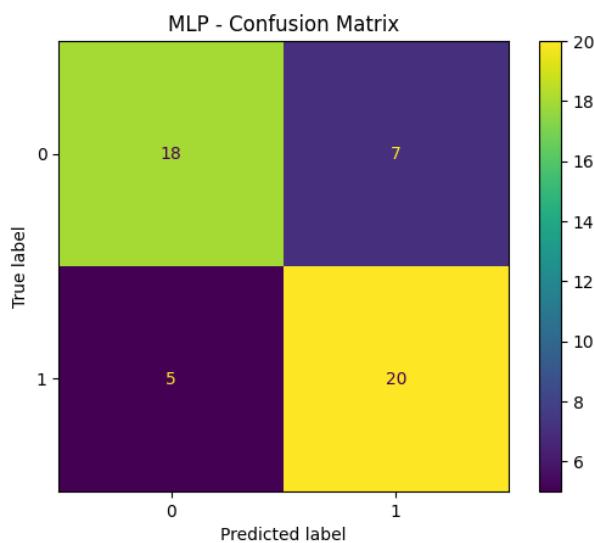
for model, model_name in models:
    evaluate_model(model, model_name, roc_curves)

plt.figure()
for fpr, tpr, roc_auc, model_name in roc_curves:
    plt.plot(fpr, tpr, lw=2, label=f'{model_name} (AUC = {roc_auc:.2f})')

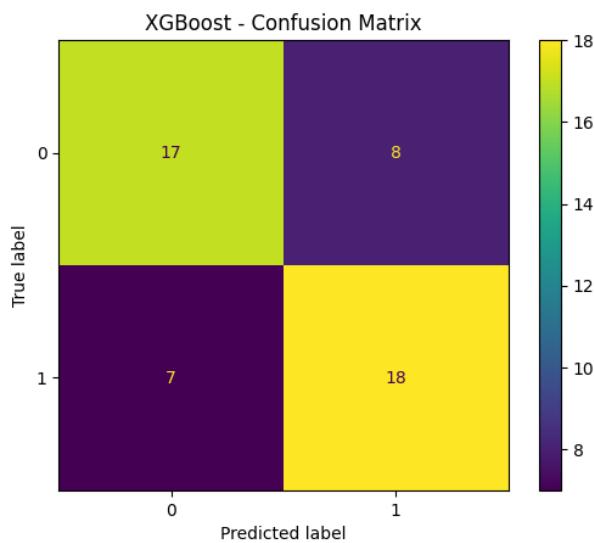
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curves')
    plt.legend(loc="lower right")
    plt.show()
```

در ابتدا مدل ها تعریف می شود، سپس هر مدل توسط تابع evaluate آموزش داده شده و پارامترهای مهم آن اندازه گیری می شود و منحنی ROC برای آن مدل رسم می شود.

همچنین ماتریس آشتفتگی برای دو طبقه‌بند MLP و XG Boost به شکل زیر خواهد بود:



شکل ۷.۴: ماتریس آشتفتگی برای MLP، دقت ۷۶٪



شکل ۸.۴: ماتریس آشتفتگی برای XG Boost، دقت ۷۰٪

۵ خوشبندی

۱.۵ پیداکردن تعداد خوشبندی بهینه

برای خوشبندی ابتدا باید تعداد خوشبندی‌های بهینه را بیابیم. برای اینکار از معیار silhouette score استفاده می‌کنیم. همچنین به شکل شهودی مشخص است چون دو کلاس داریم، دو خوشبندی باید داشته باشیم. در نتیجه بررسی این معیار باید مطابق این شهود باشد.

۱.۱.۵ پیداکردن تعداد خوشبندی‌های بهینه برای K-means

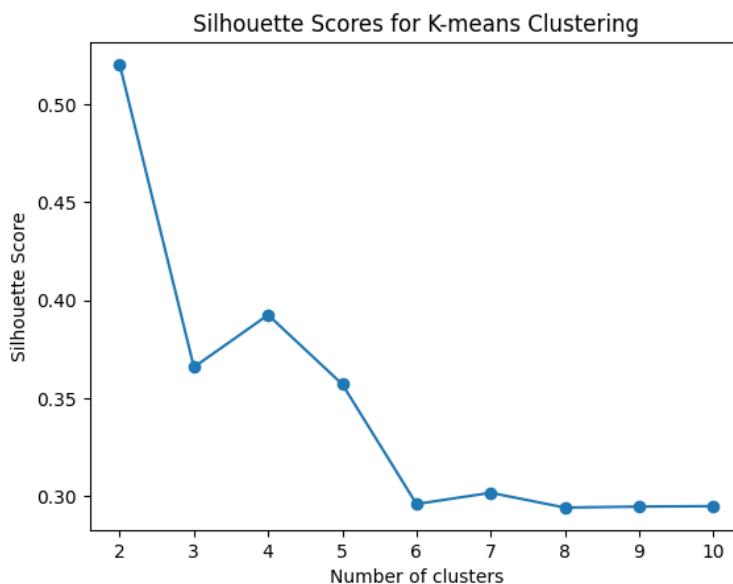
K-means

```
scaler = StandardScaler()
X_csp_scaled = scaler.fit_transform(X_csp)

def optimal_clusters_Kmeans(data, max_k=10):
    scores = []
    for k in range(2, max_k + 1):
        kmeans = KMeans(n_clusters=k, random_state=42, n_init = 10)
        kmeans.fit(data)
        score = silhouette_score(data, kmeans.labels_)
        scores.append(score)
    return scores

# Calculate silhouette scores for K-means
kmeans_scores = optimal_clusters_Kmeans(X_csp_scaled)
optimal_k = np.argmax(kmeans_scores) + 2
# +2 because we start from k=2
```

به ازای هر k معیار silhouette score محاسبه شده و در نهایت k ای که این معیار را مаксیمم کند به عنوان تعداد خوشبندی‌ها تعیین می‌شود.



شکل ۱.۵: نمودار silhouette score برای K-means به ازای تعداد خوشه‌های مختلف

مشاهده می‌کنیم این معیار در تعداد خوشه ۲ بالای ۰.۵ است و باقی موارد زیر این مقدار هستند. در نتیجه و مطابق انتظار تعداد خوشه‌های بهینه برای K-means دو خوشه بدبست می‌آید.

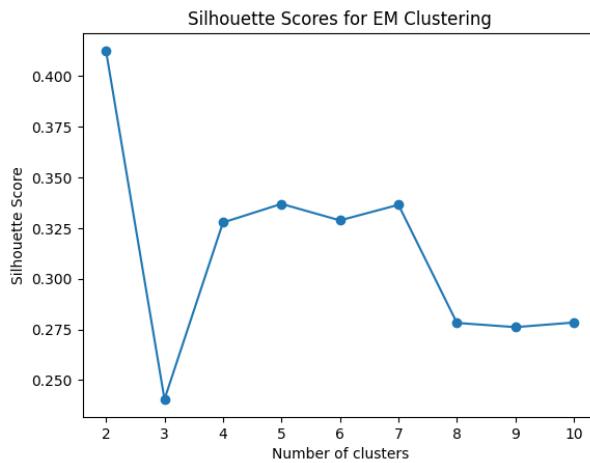
۲.۱.۵ پیدا کردن تعداد خوشه‌های بهینه برای EM

EM

```
def optimal_clusters_EM(data, max_k=10):
    scores = []
    for k in range(2, max_k + 1):
        gmm = GaussianMixture(n_components=k, random_state=42)
        gmm.fit(data)
        labels = gmm.predict(data)
        score = silhouette_score(data, labels)
        scores.append(score)
    return scores

# Calculate silhouette scores for EM
gmm_scores = optimal_clusters_EM(X_csp_scaled)
optimal_k = np.argmax(gmm_scores) + 2
# +2 because we start from k=2
```

برای پیاده‌سازی مدل EM از GMM استفاده می‌شود. مشابه روندی که برای K-Means طی شد، تعداد خوشه‌ها بدبست می‌آید.



شکل ۲.۵: نمودار silhouette score برای EM به ازای تعداد خوش‌های مختلف

در این نمودار هم مشاهده می‌کنیم که برای دو خوش‌های این معیار بیشترین مقدار خودش را دارد هرچند که مقدار آن زیر ۰.۵ است.

Clustering ۲.۵

مشابه روندی که در طبقه‌بندی طی کردیم، یکتابع برای خوش‌بندی تعریف می‌کنیم که علاوه بر رسم نمودار، معیارهای Davies-Bouldin و silhouette score را گزارش می‌کند.

Clustering

```
def calculate_cluster_metrics(data, labels, method_name):
    silhouette_avg = silhouette_score(data, labels)
    db_index = davies_bouldin_score(data, labels)
    return silhouette_avg, db_index

kmeans = KMeans(n_clusters=optimal_k, random_state=42,
                 n_init = 10)
kmeans_labels = kmeans.fit_predict(X_csp_scaled)

gmm = GaussianMixture(n_components=optimal_k, random_state=42)
gmm_labels = gmm.fit_predict(X_csp_scaled)

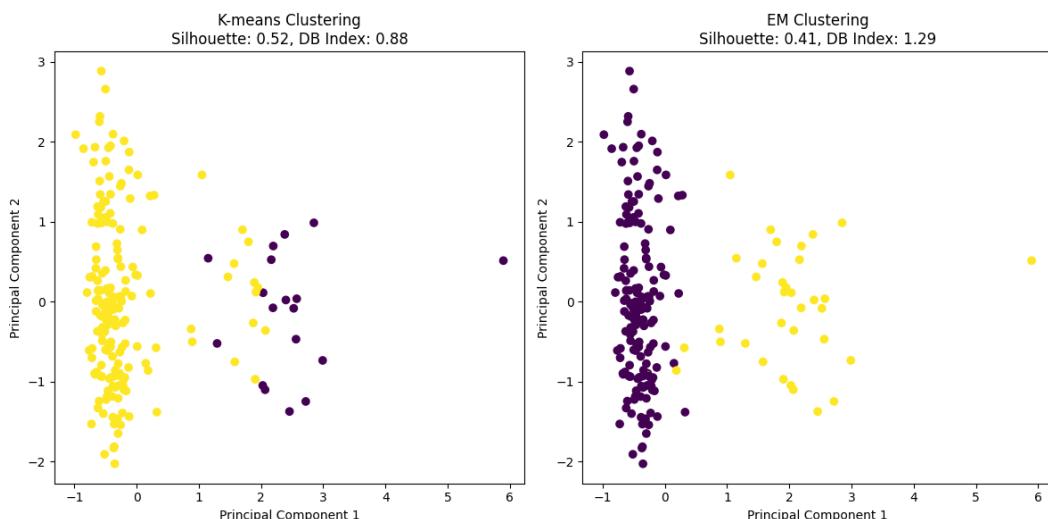
kmeans_silhouette, kmeans_db =
    calculate_cluster_metrics(X_csp_scaled, kmeans_labels,
                               "K-means")
gmm_silhouette, gmm_db =
    calculate_cluster_metrics(X_csp_scaled, gmm_labels, "GMM")
```

ابتدا دو مدل خوشبندی تعریف می‌شود و سپس خوشبندی انجام شده و نتایج ترسیم می‌گردد.

```
K-means - Silhouette Score: 0.5203320760827558
K-means - Davies-Bouldin Index: 0.8782176303053176
GMM - Silhouette Score: 0.4124451661501325
GMM - Davies-Bouldin Index: 1.2919397283425582
```

شکل ۳.۵: معیارهای گزارش شده برای خوشبندها

مشاهده می‌کنیم که روش K-means بالاتر و silhouette score ، Davies-Bouldin کمتری نسبت به EM دارد که در کل بیانگر این است که این روش کارآمدتری برای خوشبندی می‌باشد.



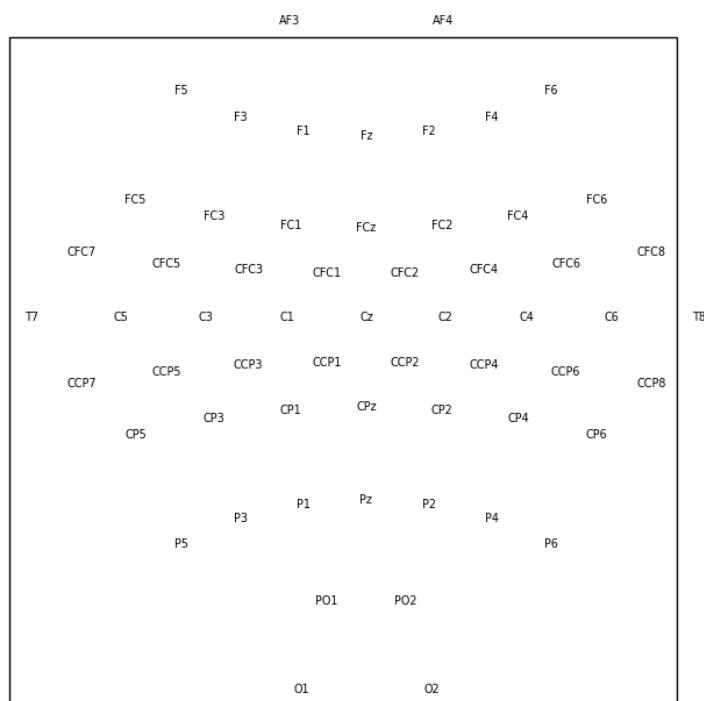
شکل ۴.۵: خوشبندی داده‌ها

با توجه به شکل‌ها مشاهده می‌کنیم که خوشبندی با دقت مناسبی انجام شده است.

۶ بررسی روند طی شده برای یک دیتاست دیگر

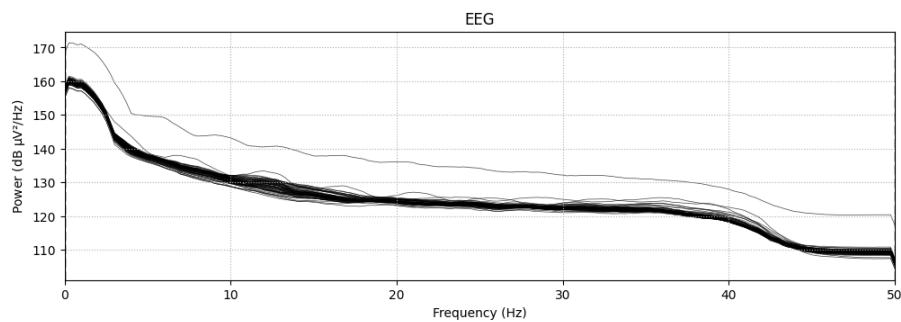
در روندی که طی کردیم دیتاست BCICIV-calib-ds1a را بررسی کردیم. در این بخش دیتاست BCICIV-calib-ds1e را بررسی می‌کنیم. کدها و توضیحات یکسان است و صرفا نتیجه را گزارش می‌کنیم.

۱.۶ پیش‌پردازش

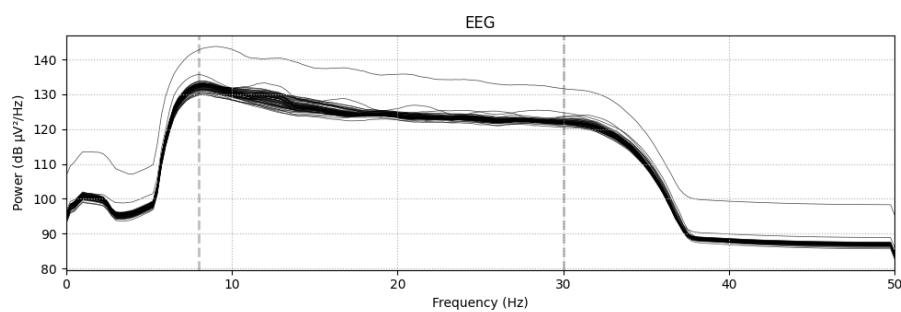


شکل ۱.۶: نمایی از موقعیت الکترودها روی مغز

Bandpass Filter

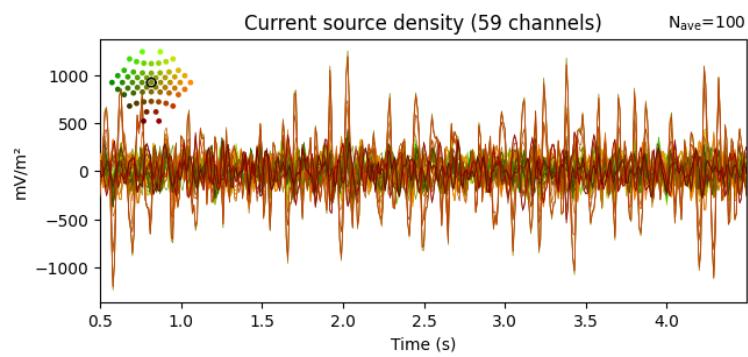


شکل ۲.۶: سیگنال‌ها قبل از اعمال فیلتر میان‌گذر

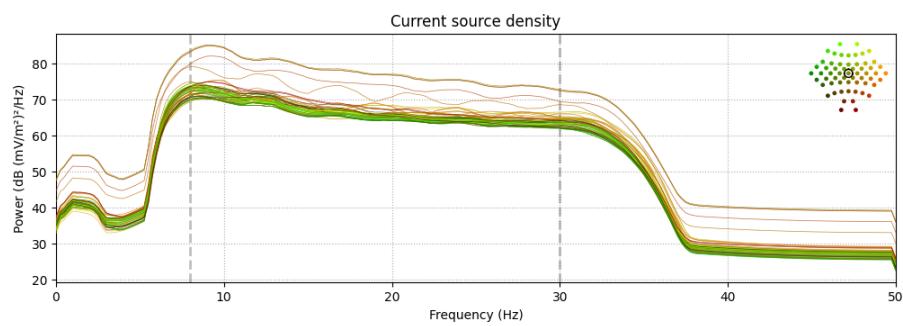


شکل ۳.۶: سیگنال‌ها بعد از اعمال فیلتر میان‌گذر

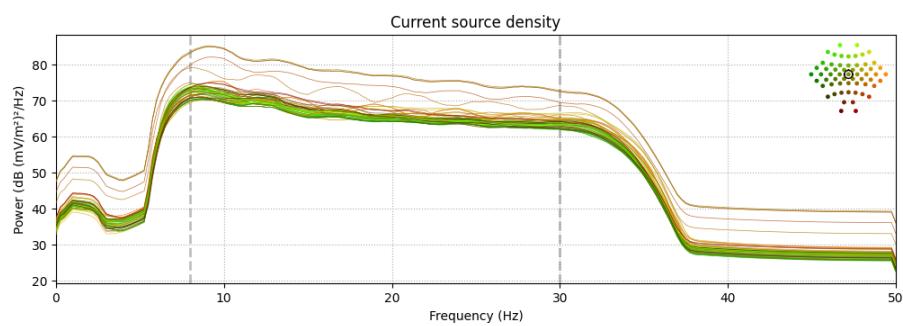
Laplacian Filter



شکل ۴.۶: نمودار سیگنال‌ها پس از اعمال فیلتر لاپلاسین (برای کلاس left)

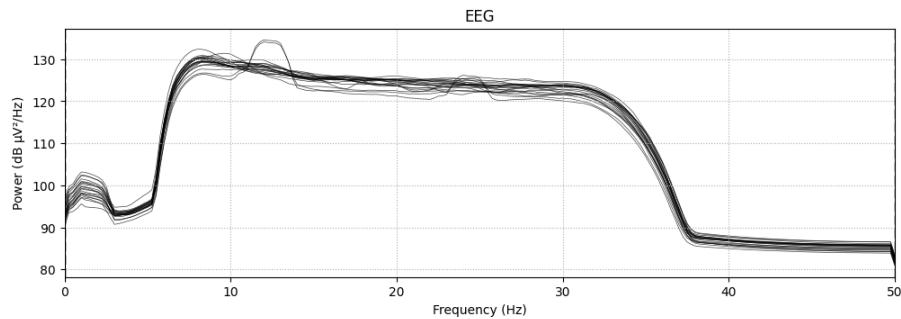


شکل ۵.۶: نمودار سیگنال‌ها در طیف فرکانسی پیش از اعمال فیلتر لاپلاسین

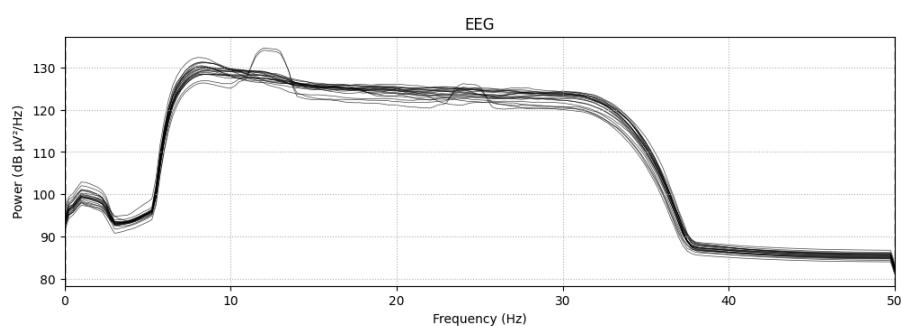


شکل ۶.۶: نمودار سیگنال‌ها در طیف فرکانسی پس از اعمال فیلتر لاپلاسین

ICA Filtering

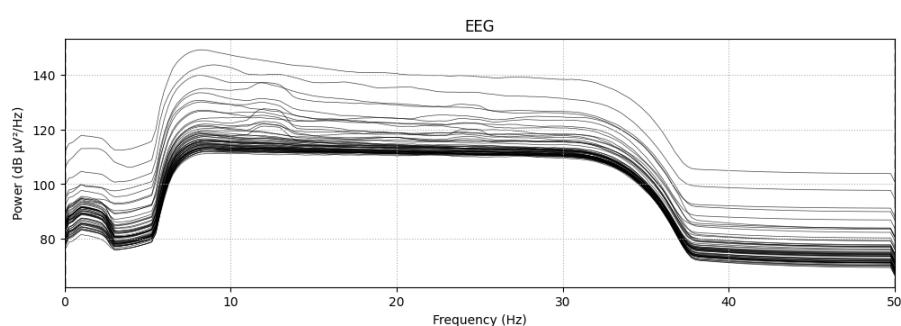


شکل ۷.۶: نمودار سیگنال‌ها در طیف فرکانسی پیش از اعمال فیلتر ICA

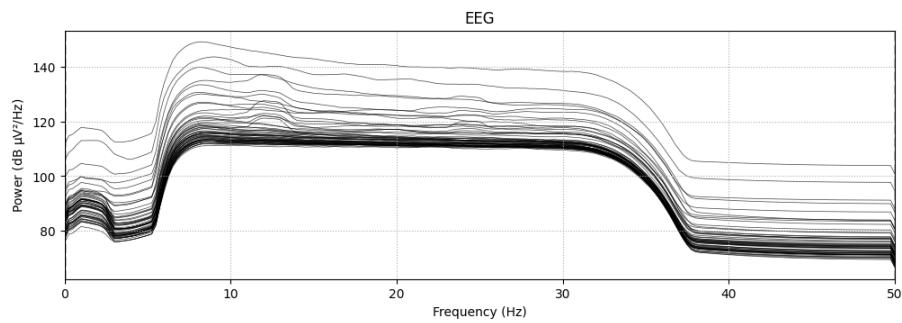


شکل ۸.۶: نمودار سیگنال‌ها در طیف فرکانسی پیش از اعمال فیلتر ICA

PCA FilteringING



شکل ۹.۶: نمودار سیگنال‌ها در طیف فرکانسی پیش از اعمال فیلتر PCA

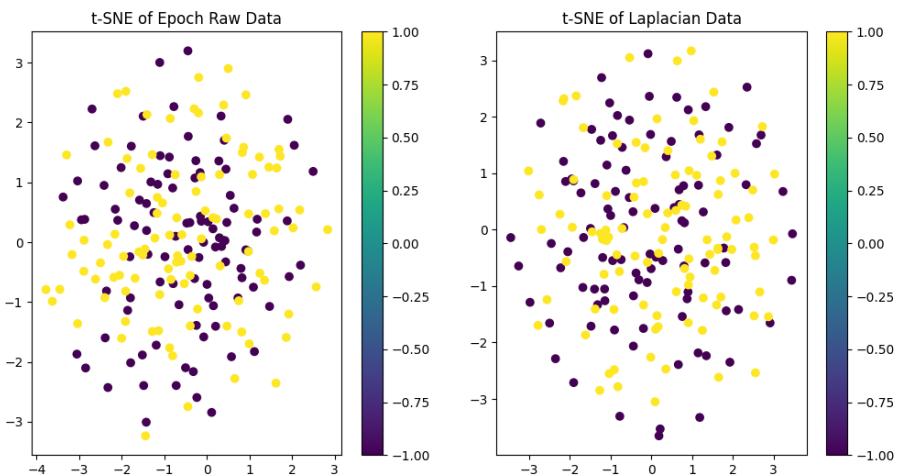


شکل ۱۰.۶: نمودار سیگنال‌ها در طیف فرکانسی پس از اعمال فیلتر PCA

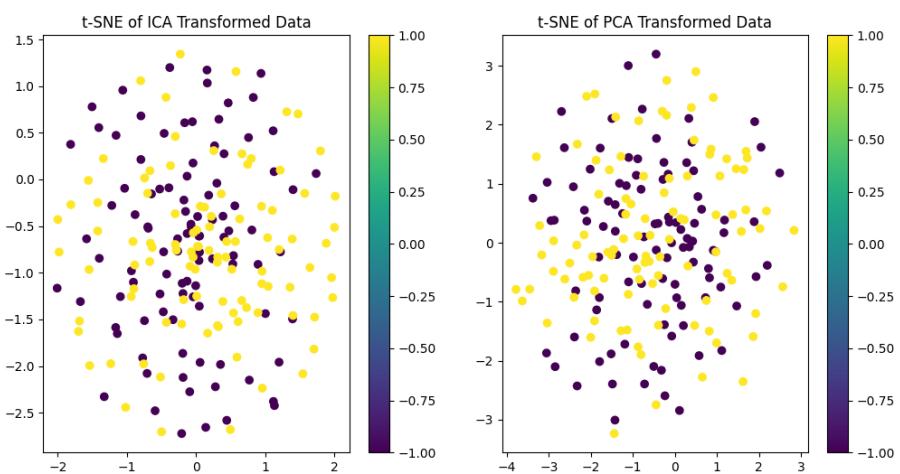
تصویرسازی داده‌ها



شکل ۱۱.۶: داده‌ها قبل از اعمال فیلترهای فضایی و epoching

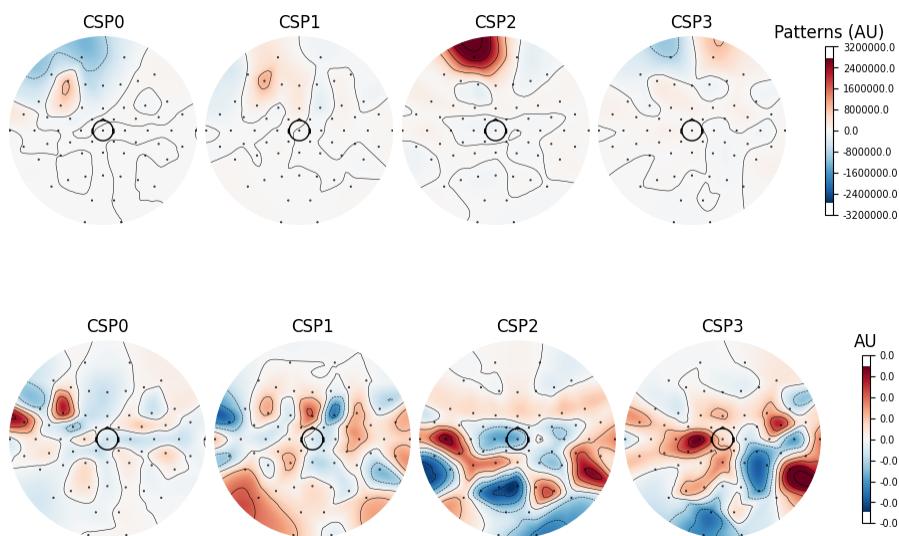


شکل ۱۲.۶: داده‌ها در حالت فقط epoching و فیلتر لاپلاسین



شکل ۱۳.۶: تصویرسازی داده‌ها برای دو تکنیک ICA و PCA

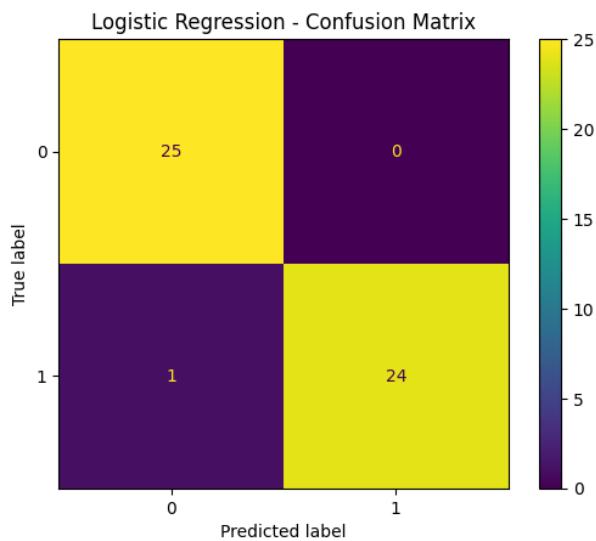
۲.۶ استخراج ویژگی



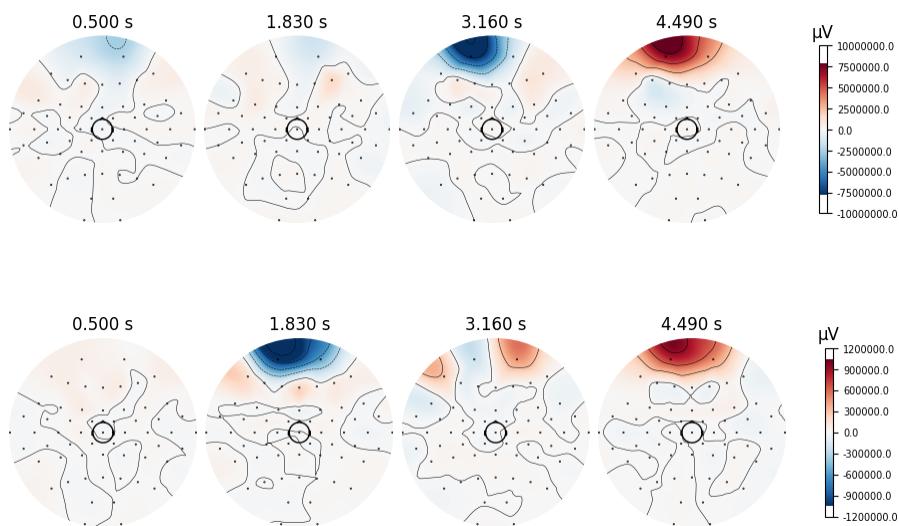
شکل ۱۴.۶: نمودار کانال‌های فعال برای الگوهای مختلف CSP

دقت الگوریتم CSP در حدود 94.5% می‌باشد. در این دیتاست انتخاب درست کانال‌ها می‌تواند به دقتش بالایی منجر شود.

۳.۶ طبقه‌بندی

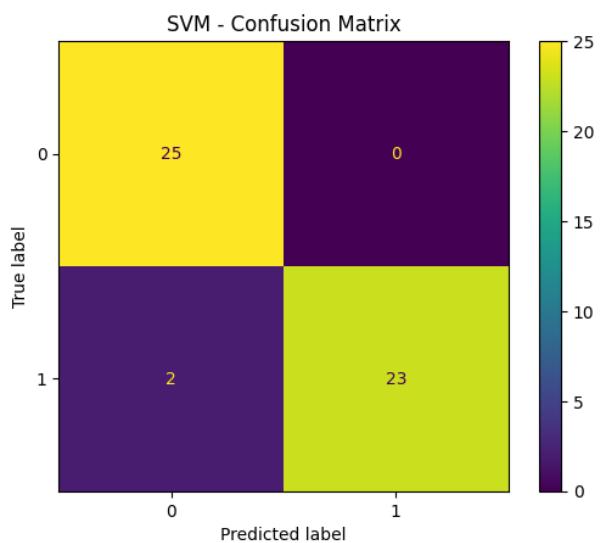


شکل ۱۵.۶: ماتریس آشتفتگی برای Logistic Regression ، دقت ۱۰۰%

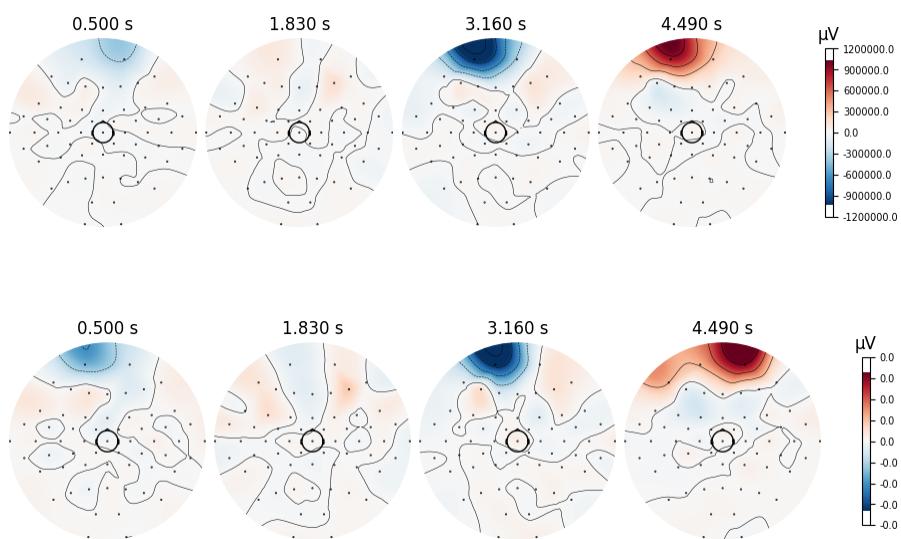


شکل ۱۶.۶: نمودار نواحی مغز در تست‌های مختلف و بررسی نقش کانال‌های انتخاب شده

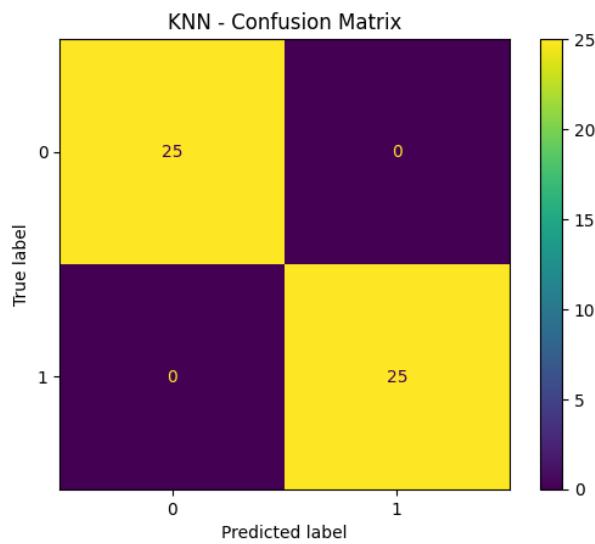
مشاهده می‌کنیم در این دیتابست کانال‌ها به طور کامل یک حرکت را پیش‌بینی می‌کنند و طبقه‌بند دقت بالایی را دارد.



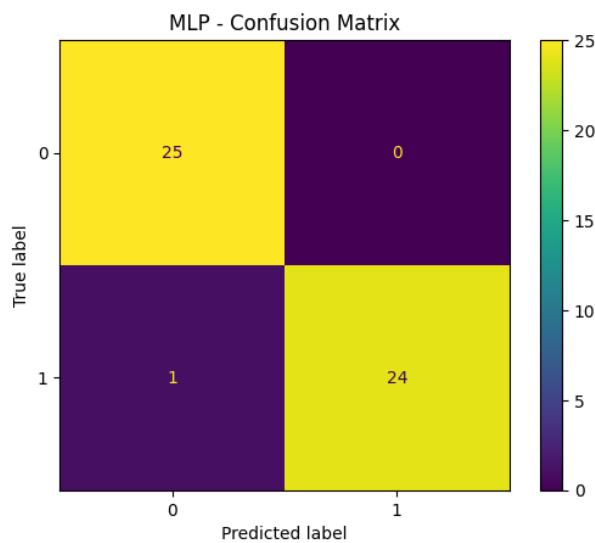
شکل ۱۷.۶: ماتریس آشتفتگی برای SVM ، دقت ۹۶٪



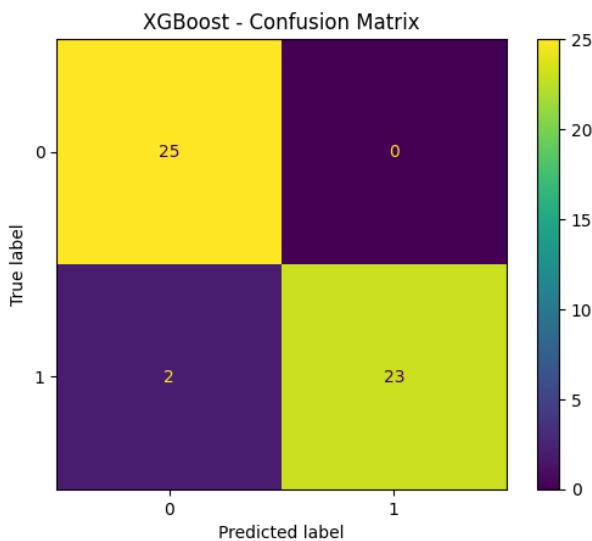
شکل ۱۸.۶: نمودار نواحی مغز در تست‌های مختلف و بررسی نقش کانال‌های انتخاب شده



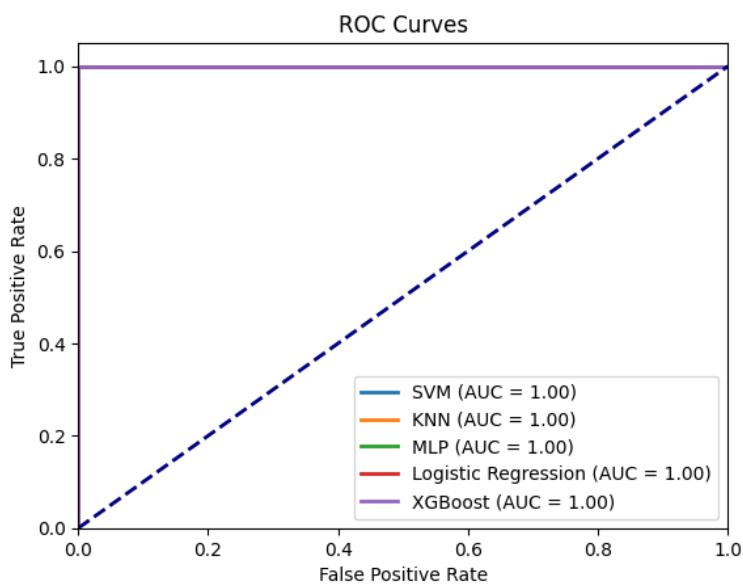
شکل ۱۹.۶: ماتریس آشفتگی برای KNN ، دقت ۱۰۰%



شکل ۲۰.۶: ماتریس آشفتگی برای MLP ، دقت ۹۸%



شکل ۲۱.۶: ماتریس آشتفتگی برای XG Boost ، دقت ۹۶٪

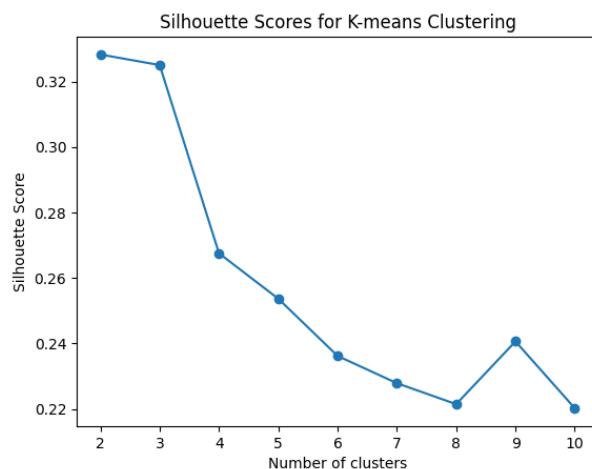


شکل ۲۲.۶: نمودار ROC

با انتخاب فیچرهای مناسب تمامی طبقه‌بندها با دقت بالایی داده‌ها را طبقه‌بندی می‌کنند. این اهمیت کاهش بعد و انتخاب فیچرهای مناسب را به خوبی نشان می‌دهد.

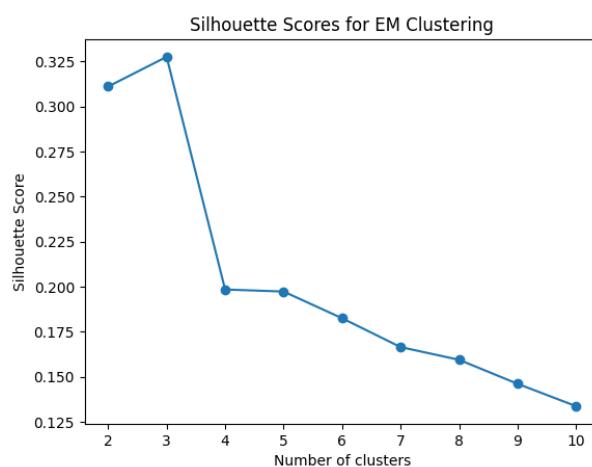
۴.۶ خوشبندی

K-means



شکل ۲۳.۶: نمودار silhouette score برای K-means به ازای تعداد خوشبندی‌های مختلف ا وجود اینکه تمامی مقادیر کمتر از ۰.۵ هستند، تعداد دو خوشبندی برای خوشبندی این روش انتخاب می‌شود.

EM

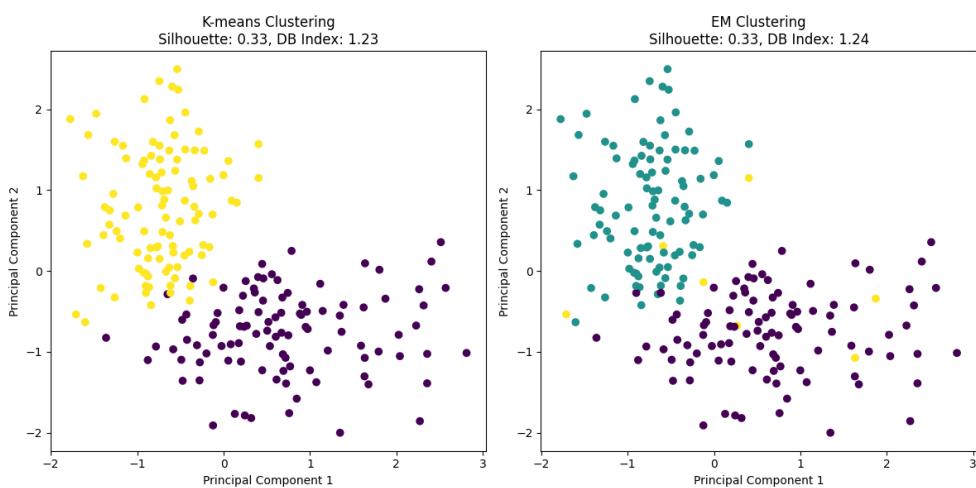


شکل ۲۴.۶: نمودار silhouette score برای EM به ازای تعداد خوشبندی‌های مختلف در این نمودار تعداد خوشبندی ۳ امتیاز بیشتری برای EM دارد در نتیجه با سه خوشبندی می‌شود.

Clustering

```
K-means - Silhouette Score: 0.3282431810858516  
K-means - Davies-Bouldin Index: 1.2288775871874669  
GMM - Silhouette Score: 0.3275788189983024  
GMM - Davies-Bouldin Index: 1.236658770751862
```

شکل ۲۵.۶: معیارهای گزارش شده برای خوشبندها



شکل ۲۶.۶: خوشبندی داده‌ها

مشاهده می‌کنیم تعداد کمی از داده‌ها در روش EM در خوشبندی سوم قرار می‌گیرند که صرفاً خطای مدل را بالا می‌برند. در مجموع باز هم روش K-means کارآمدتر می‌باشد.

۵.۶ مقایسه نتایج با دیتاست قبلی

این دیتاست از توزیع بهتری برخوردار بود بطوریکه با کاهش بعد اطلاعات زیادی از دست نرفت و طبقه‌بندی‌ها دقیق‌تری داشتند.

در خوشبندی الگوریتم EM دچار خطای تعداد خوشبندی‌های بهینه شد که یک خطای ذاتی است. در مجموع روند طی شده در این پژوهه روی این دیتاست خیلی بهتر جواب می‌دهد.