```
pip install mpld3
```

```
from sklearn.decomposition import PCA
from sklearn.utils import shuffle

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

import sklearn.metrics as mt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score, precision_score, accuracy_score
from sklearn.metrics import confusion_matrix, f1_score, classification_report
from sklearn.model_selection import cross_val_score
from scipy.io import loadmat
import numpy as np
from sklearn.metrics import mutual_info_score
from scipy.stats import kendalltau
from scipy.stats import spearmanr
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from scipy.io import loadmat
```

```
##Experiment 1 Subject 1 & 2 Trail 1&2
E1Sub1Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject1Trial1.mat')['X']
E1Sub2Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject2Trial1.mat')['X']

E1Sub3Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject3Trial1.mat')['X']
E1Sub4Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject4Trial1.mat')['X']

E1Sub1Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject1Trial2.mat')['X']
E1Sub2Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject2Trial2.mat')['X']

E1Sub3Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject3Trial2.mat')['X']
E1Sub4Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment1-Hyper-CSP/Subject4Trial2.mat')['X']

##Experiment 2 Subject 3 & 4 Trail 1&2
E2Sub1Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject1Trial1.mat')['X']
E2Sub2Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject2Trial1.mat')['X']

E2Sub3Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject3Trial1.mat')['X']
E2Sub4Trial1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject4Trial1.mat')['X']

E2Sub1Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject1Trial2.mat')['X']
E2Sub2Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject2Trial2.mat')['X']

E2Sub3Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject3Trial2.mat')['X']
E2Sub4Trial2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment-2-Hyper-CSP/Subject4Trial2.mat')['X']
data = np.concatenate([E1Sub1Trial1,E1Sub2Trial1,E1Sub3Trial1,E1Sub4Trial1,E1Sub1Trial2,E1Sub2Trial2,E1Sub3Trial2,E
                       E2Sub1Trial1,E2Sub2Trial1,E2Sub3Trial1,E2Sub4Trial1,E2Sub1Trial2,E2Sub2Trial2,E2Sub3Trial2,E

Labels1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject12Trial1Label.mat')['Labels'].T   ##Labels
Labels2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject12Trial2Label.mat')['Labels'].T
Labels3 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject34Trial1Label.mat')['Labels'].T
Labels4 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject34Trial2Label.mat')['Labels'].T
Labels5 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject12Trial1Label.mat')['Labels'].T
Labels6 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject12Trial2Label.mat')['Labels'].T
Labels7 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject34Trial1Label.mat')['Labels'].T
Labels8 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject34Trial2Label.mat')['Labels'].T

Labels = np.concatenate([Labels1,Labels1,Labels2,Labels2,Labels3,Labels3,Labels4,Labels4,
                         Labels5,Labels5,Labels6,Labels6,Labels7,Labels7,Labels8,Labels8], axis=0)

data.shape, Labels.shape
```

```
((464, 30), (464, 1))
```

```
data1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject12Trial1.mat')['data']
data2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject12Trial2.mat')['data']
data3 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject34Trial1.mat')['data']
data4 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject34Trial2.mat')['data']
data5  = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject12Trial1.mat')['data']
data6  = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject12Trial2.mat')['data']
data7  = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject34Trial1.mat')['data']
data8  = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject34Trial2.mat')['data']
data = np.concatenate([data1,data2,data3,data4,data5,data6,data7,data8],axis=2)

subj1 = data[:30,:,:]
subj1 = subj1.reshape(subj1.shape[2],subj1.shape[0],subj1.shape[1])
subj2 = data[30:,:,:]
subj2 = subj2.reshape(subj2.shape[2],subj2.shape[0],subj2.shape[1])
subj = data.reshape(data.shape[2],data.shape[0],data.shape[1])
Labels1 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject12Trial1Label.mat')['Labels'].T   ##Labels
Labels2 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject12Trial2Label.mat')['Labels'].T
Labels3 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject34Trial1Label.mat')['Labels'].T
Labels4 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 1/Subject34Trial2Label.mat')['Labels'].T
Labels5 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject12Trial1Label.mat')['Labels'].T
Labels6 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject12Trial2Label.mat')['Labels'].T
Labels7 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject34Trial1Label.mat')['Labels'].T
Labels8 = loadmat('/content/drive/MyDrive/GRAPH EEG/Experiment 2/Subject34Trial2Label.mat')['Labels'].T


Labels = np.concatenate([Labels1,Labels2,Labels3,Labels4,Labels5,Labels6,Labels7,Labels8], axis=0)
subj.shape,Labels.shape
```

```
# data_subj1 = data[:30,:,:]
# data_subj2 = data[30:,:,:]
# data_concatenate = np.concatenate([data_subj1,data_subj2],axis=2)

# data_labels = np.concatenate([Labels,Labels],axis=0)
# data_concatenate.shape, data_labels.shape
```

```
# import numpy as np
# import matplotlib.pyplot as plt
# from sklearn.preprocessing import StandardScaler
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# from sklearn.svm import SVC
# from sklearn.metrics import accuracy_score, classification_report
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.neural_network import MLPClassifier
# from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, KFold, cross_validate
# from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, f1_score

# # # Correct transpose
# # data_concatenate = np.transpose(data_concatenate, (2, 0, 1))
# # print("Transposed shape of data_concatenate:", data_concatenate.shape)

# # # Now, reshape to (464, 30 * 1000)
# # X = data_concatenate.reshape(464, 30 * 1000)
# # print("Shape of X after reshaping:", X.shape)


# X = data
# # Assuming data_labels is (464, 1), we flatten it to (464,)
# y = Labels
# #y = y.ravel()

# # # Normalize the data
# # scaler = StandardScaler()
# # X_scaled = scaler.fit_transform(X)

# # Split the data into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)


```

Extra Work

```python
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import cross_validate, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import compute_sample_weight

from sklearn.utils.validation import has_fit_parameter

kf = KFold(n_splits=10, shuffle=True, random_state=42)

def evaluate_model(model, X_train, y_train, X_test, y_test, kf):
    scoring = ['accuracy', 'precision', 'recall', 'f1']

    if has_fit_parameter(model, 'sample_weight'):
        # Use sample weights if the model supports them
        sample_weight = compute_sample_weight('balanced', y_train)
        cv_results = cross_validate(model, X_train, y_train, cv=kf, scoring=scoring,
                                    fit_params={'sample_weight': sample_weight})
    else:
        # Otherwise, perform standard cross-validation
        cv_results = cross_validate(model, X_train, y_train, cv=kf, scoring=scoring)

    # Mean CV scores
    mean_cv_accuracy = np.mean(cv_results['test_accuracy']) * 100
    mean_cv_precision = np.mean(cv_results['test_precision']) * 100
    mean_cv_recall = np.mean(cv_results['test_recall']) * 100
    mean_cv_f1 = np.mean(cv_results['test_f1']) * 100

    # Max CV scores
    max_cv_accuracy = np.max(cv_results['test_accuracy']) * 100
    max_cv_precision = np.max(cv_results['test_precision']) * 100
    max_cv_recall = np.max(cv_results['test_recall']) * 100
    max_cv_f1 = np.max(cv_results['test_f1']) * 100

    print(f"Mean CV Accuracy: {mean_cv_accuracy:.2f}%")
    print(f"Max CV Accuracy: {max_cv_accuracy:.2f}%")
    print(f"Mean CV Precision: {mean_cv_precision:.2f}%")
    print(f"Max CV Precision: {max_cv_precision:.2f}%")
    print(f"Mean CV Recall: {mean_cv_recall:.2f}%")
    print(f"Max CV Recall: {max_cv_recall:.2f}%")
    print(f"Mean CV F1-Score: {mean_cv_f1:.2f}%")
    print(f"Max CV F1-Score: {max_cv_f1:.2f}%")

    # Train and evaluate on the test set
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Test scores
    test_accuracy = accuracy_score(y_test, y_pred) * 100
    test_precision = precision_score(y_test, y_pred, average='binary') * 100
    test_recall = recall_score(y_test, y_pred, average='binary') * 100
    test_f1 = f1_score(y_test, y_pred, average='binary') * 100

    print(f"Test Accuracy: {test_accuracy:.2f}%")
    print(f"Test Precision: {test_precision:.2f}%")
    print(f"Test Recall: {test_recall:.2f}%")
    print(f"Test F1-Score: {test_f1:.2f}%")

```

```python
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline as make_pipeline_imb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
# Correct transpose
# data_concatenate = np.transpose(data_concatenate, (2, 0, 1))
# print("Transposed shape of data_concatenate:", data_concatenate.shape)

# Now, reshape to (464, 30 * 1000)
X = data
print("Shape of X after reshaping:", X.shape)
```

```python

# Assuming data_labels is (464, 1), we flatten it to (464,)
y = Labels
y = y.ravel()

# Normalize the data
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X)

# # Apply PCA to reduce dimensionality
# pca = PCA(n_components=0.85)  # Keep 95% of variance
# X_pca = pca.fit_transform(X_scaled)
# print("Shape of X after PCA:", X_pca.shape)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42)

# # Apply SMOTE for imbalance handling before model training
# smote = SMOTE()
# X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
Shape of X after reshaping: (464, 30)
```

```python
svclassifier = SVC(kernel='rbf',random_state=42)
evaluate_model(svclassifier, X_train, y_train, X_test, y_test, kf)
# svclassifier.fit(X_train_smote, y_train_smote)
# predicted_labels = svclassifier.predict(X_test)
# print(mt.classification_report(y_test,predicted_labels))
```

```
Mean CV Accuracy: 58.26%
Max CV Accuracy: 66.67%
Mean CV Precision: 57.16%
Max CV Precision: 71.43%
Mean CV Recall: 51.01%
Max CV Recall: 64.00%
Mean CV F1-Score: 52.82%
Max CV F1-Score: 66.67%
Test Accuracy: 68.09%
Test Precision: 76.47%
Test Recall: 54.17%
Test F1-Score: 63.41%
```

```python
RFClassifier = RandomForestClassifier(random_state=42)
evaluate_model(RFClassifier, X_train, y_train, X_test, y_test, kf)
# RFClassifier.fit(X_train_smote, y_train_smote)
# predicted_labels = RFClassifier.predict(X_test)
# print(mt.classification_report(y_test,RFClassifier.predict(X_test)))
```

```
Mean CV Accuracy: 59.73%
Max CV Accuracy: 71.43%
Mean CV Precision: 58.27%
Max CV Precision: 80.00%
Mean CV Recall: 51.66%
Max CV Recall: 75.00%
Mean CV F1-Score: 53.98%
Max CV F1-Score: 68.18%
Test Accuracy: 61.70%
Test Precision: 68.75%
Test Recall: 45.83%
Test F1-Score: 55.00%
```

```python
XGBvlassifier = XGBClassifier(random_state=42,use_label_encoder=False)
evaluate_model(XGBvlassifier, X_train, y_train, X_test, y_test, kf)
# XGBvlassifier.fit(X_train_smote, y_train_smote)
# predicted_labels = XGBvlassifier.predict(X_test)
# print(mt.classification_report(y_test,predicted_labels))
```

```
Mean CV Accuracy: 61.14%
Max CV Accuracy: 69.05%
Mean CV Precision: 58.75%
Max CV Precision: 72.00%
Mean CV Recall: 58.81%
Max CV Recall: 75.00%
Mean CV F1-Score: 58.30%
Max CV F1-Score: 73.47%
Test Accuracy: 57.45%
Test Precision: 58.33%
Test Recall: 58.33%
Test F1-Score: 58.33%
```

```python
classifier = KNeighborsClassifier(n_neighbors =2)
evaluate_model(classifier, X_train, y_train, X_test, y_test, kf)
# classifier.fit(X_train, y_train)
# y_pred = classifier.predict(X_test)
# print(mt.classification_report(y_test,y_pred))
```

```
Mean CV Accuracy: 64.99%
Max CV Accuracy: 75.61%
Mean CV Precision: 79.97%
Max CV Precision: 90.00%
Mean CV Recall: 36.88%
Max CV Recall: 47.62%
Mean CV F1-Score: 49.92%
Max CV F1-Score: 60.61%
Test Accuracy: 63.83%
Test Precision: 81.82%
Test Recall: 37.50%
Test F1-Score: 51.43%
```

```python
LDAclassifier = LinearDiscriminantAnalysis()
evaluate_model(LDAclassifier, X_train, y_train, X_test, y_test, kf)
```

```
Mean CV Accuracy: 57.06%
Max CV Accuracy: 73.81%
Mean CV Precision: 55.01%
Max CV Precision: 77.78%
Mean CV Recall: 52.11%
Max CV Recall: 75.00%
Mean CV F1-Score: 52.63%
Max CV F1-Score: 71.79%
Test Accuracy: 65.96%
Test Precision: 68.18%
Test Recall: 62.50%
Test F1-Score: 65.22%
```

```
Start coding or generate with AI.
```