```
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
```

```
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
```

```
((232, 60, 1000), (232, 1))
```

```
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
```

```
((30, 1000, 464), (464, 1))
```

```
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)

# Assuming data_labels is (464, 1), we flatten it to (464,)
y = data_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_scaled, y, test_size=0.1, random_state=42)


```

```
Transposed shape of data_concatenate: (1000, 464, 30)
Shape of X after reshaping: (464, 30000)
```

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}%")

```

```
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_concatenate.reshape(464, 30 * 1000)
print("Shape of X after reshaping:", X.shape)

# Assuming data_labels is (464, 1), we flatten it to (464,)
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_pca, 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)
```

```
Transposed shape of data_concatenate: (30, 1000, 464)
Shape of X after reshaping: (464, 30000)
Shape of X after PCA: (464, 256)
```

```
svclassifier = SVC(kernel='rbf',random_state=42)
evaluate_model(svclassifier, X_train_smote, y_train_smote, 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: 71.62%
Max CV Accuracy: 79.55%
Mean CV Precision: 73.61%
Max CV Precision: 95.00%
Mean CV Recall: 69.40%
Max CV Recall: 81.25%
Mean CV F1-Score: 70.17%
Max CV F1-Score: 80.85%
Test Accuracy: 80.85%
Test Precision: 85.71%
Test Recall: 75.00%
Test F1-Score: 80.00%
```

```
RFClassifier = RandomForestClassifier(random_state=42)
evaluate_model(RFClassifier, X_train_smote, y_train_smote, 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: 67.07%
Max CV Accuracy: 81.40%
Mean CV Precision: 71.53%
Max CV Precision: 94.12%
Mean CV Recall: 59.15%
Max CV Recall: 69.57%
Mean CV F1-Score: 63.81%
Max CV F1-Score: 80.00%
Test Accuracy: 70.21%
Test Precision: 72.73%
Test Recall: 66.67%
Test F1-Score: 69.57%
```

```
XGBvlassifier = XGBClassifier(random_state=42,use_label_encoder=False)
evaluate_model(XGBvlassifier, X_train_smote, y_train_smote, 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: 64.71%
Max CV Accuracy: 75.00%
Mean CV Precision: 64.67%
Max CV Precision: 83.33%
Mean CV Recall: 64.03%
Max CV Recall: 77.78%
Mean CV F1-Score: 63.47%
```

```
classifier = KNeighborsClassifier(n_neighbors =2)
evaluate_model(classifier, X_train_smote, y_train_smote, 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: 73.22%
Max CV Accuracy: 86.36%
Mean CV Precision: 66.83%
Max CV Precision: 83.87%
Mean CV Recall: 93.87%
Max CV Recall: 100.00%
Mean CV F1-Score: 77.39%
Max CV F1-Score: 89.66%
Test Accuracy: 87.23%
Test Precision: 84.62%
Test Recall: 91.67%
Test F1-Score: 88.00%
```

```
LDAclassifier = LinearDiscriminantAnalysis()
evaluate_model(LDAclassifier, X_train_smote, y_train_smote, X_test, y_test, kf)
```

```
Mean CV Accuracy: 71.84%
Max CV Accuracy: 84.09%
Mean CV Precision: 69.22%
Max CV Precision: 86.36%
Mean CV Recall: 72.66%
Max CV Recall: 92.59%
Mean CV F1-Score: 70.70%
Max CV F1-Score: 87.72%
Test Accuracy: 78.72%
Test Precision: 79.17%
Test Recall: 79.17%
Test F1-Score: 79.17%
```

```
pca.fit(X_scaled)
var= pca.explained_variance_ratio_
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
plt.figure(figsize=(8, 4))
plt.plot(range(256),var1,'ro-', linewidth=1)
plt.title('Cumulative Explained Variance')
plt.ylabel('Cumulative explained variance ratio')
plt.xlabel('Principal components')
plt.show()
```