```python
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
import pandas as pd
import glob
from mpl_toolkits import mplot3d
from scipy import io
from scipy.signal import butter, lfilter, freqz
from scipy.interpolate import interp2d
from statistics import median
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
WINDOW_SIZE = 150    # 20:9.76ms, 150:73.2ms
TEST_RATIO = 0.3
SEGMENT_N = 3
PLOT_RANDOM_DATA = False
PLOT_CONFUSION_MATRIX = True
ACTUAL_COLUMN=24
ACTUAL_RAW=7
IDLE_GESTURE_EXIST = True

def load_mat_files(dataDir):
    pathname=dataDir + "/**/*.mat"
    files = glob.glob(pathname, recursive=True)
    sessions=dict()
    #In idle gesture, we just use 2,4,7,8,11,13,19,25,26,30th tries in order to match the number of datas
    for one_file in files:
        session_name=one_file.split("\\")[-2]
        if not session_name in sessions:
            if one_file[-5:]=="0.mat" and IDLE_GESTURE_EXIST == True:
                sessions[session_name]=np.array([io.loadmat(one_file)['gestures'][[1,3,6,7,10,12,18,24,25,29]]])
            else: sessions[session_name]=np.array([io.loadmat(one_file)['gestures']])
            continue
        if one_file[-5:]=="0.mat" and IDLE_GESTURE_EXIST == True:
            sessions[session_name]=np.append(sessions[session_name], [io.loadmat(one_file)['gestures'][[1,3,6,7,10,12,18,24,25,29]]], axis=0)
            continue
        sessions[session_name]=np.append(sessions[session_name], [io.loadmat(one_file)['gestures']], axis=0)
    return sessions

def butter_bandpass_filter(data, lowcut=20.0, highcut=400.0, fs=2048, order=4):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    y = lfilter(b, a, data)
    return y

def plot_bandpass_filtered_data(data):
    plt.figure(1)
    plt.clf()
```

```
    plt.plot(data, label='Noisy signal')

    y = butter_bandpass_filter(data)
    plt.plot(y, label='Filtered signal')
    plt.xlabel('time (seconds)')
    plt.grid(True)
    plt.axis()
    plt.legend(loc='upper left')
    plt.show()

def compute_RMS(datas):
    return np.sqrt(np.mean(np.array(datas)**2))

def compute_RMS_gestures(gestures):
    for i_ges in range(gestures.shape[0]):
        for i_try in range(gestures.shape[1]):
            for i_win in range(gestures.shape[2]):
                for i_ch in range(gestures.shape[3]):
                    RMS_gestures[i_ges][i_try][i_win]
[i_ch]=compute_RMS(gestures[i_ges][i_try][i_win][i_ch])
    return RMS_gestures

def average_for_channel(gesture):
    average=np.array([])
    for i_ch in range(gesture.shape[2]):
        sum=0
        for i_win in range(gesture.shape[1]):
            for i_try in range(gesture.shape[0]):
                sum+=gesture[i_try][i_win][i_ch]
        average=np.append(average, [sum/(gesture.shape[1]*gesture.shape[0])])
    return average

def base_normalization(RMS_gestures):
    average_channel_idle_gesture=average_for_channel(RMS_gestures[0])
    for i_ges in range(RMS_gestures.shape[0]):   # Including idle gesture
        for i_try in range(RMS_gestures.shape[1]):
            for i_win in range(RMS_gestures.shape[2]):
                for i_ch in range(RMS_gestures.shape[3]):
                    RMS_gestures[i_ges][i_try][i_win][i_ch]-
=average_channel_idle_gesture[i_ch]
    return RMS_gestures

def extract_ACTIVE_window_i(RMS_gestures):
    for i_ges in range(len(RMS_gestures)):
        for i_try in range(len(RMS_gestures[i_ges])):
            # Segmentation : Determine whether ACTIVE : Compute summarized RMS
            sum_RMSs=[sum(window) for window in RMS_gestures[i_ges][i_try]]
            threshold=sum(sum_RMSs)/len(sum_RMSs)
            # Segmentation : Determine whether ACTIVE
            i_ACTIVEs=[]
            for i_win in range(len(RMS_gestures[i_ges][i_try])):
                if sum_RMSs[i_win] > threshold and i_win>0:     # Exclude 0th index
                    i_ACTIVEs.append(i_win)
            for i in range(len(i_ACTIVEs)):
                if i==0:
                    continue
                if i_ACTIVEs[i]-i_ACTIVEs[i-1] == 2:
                    i_ACTIVEs.insert(i, i_ACTIVEs[i-1]+1)
            # Segmentation : Determine whether ACTIVE : Select the longest
contiguous sequences
```

```
            segs=[]
            contiguous = 0
            for i in range(len(i_ACTIVEs)):
                if i == len(i_ACTIVEs)-1:
                    if contiguous!=0:
                        segs.append((start, contiguous))
                    break
                if i_ACTIVEs[i+1]-i_ACTIVEs[i] == 1:
                    if contiguous == 0:
                        start=i_ACTIVEs[i]
                    contiguous+=1
                else:
                    if contiguous != 0:
                        contiguous+=1
                        segs.append((start, contiguous))
                        contiguous=0
            if len(segs)==0:
                seg_start= sorted(i_ACTIVEs, reverse=True)[0]
                seg_len=1
            else:
                seg_start, seg_len = sorted(segs, key=lambda seg: seg[1], reverse=True)[0]
            # Segmentation : Return ACTIVE window indexes
            if i_try==0:
                i_one_try_ACTIVE = np.array([[seg_start, seg_len]])
                continue
            i_one_try_ACTIVE = np.append(i_one_try_ACTIVE, [[seg_start, seg_len]], axis=0)
        if i_ges==0:
            i_ACTIVE_windows = np.array([i_one_try_ACTIVE])
            continue
        i_ACTIVE_windows = np.append(i_ACTIVE_windows, [i_one_try_ACTIVE], axis=0)
    return i_ACTIVE_windows

def medfilt(channel, kernel_size=3):
    filtered=np.zeros(len(channel))
    for i in range(len(channel)):
        if i-kernel_size//2 <0 or i+kernel_size//2 >=len(channel):
            continue
        filtered[i]=median([channel[j] for j in range(i-kernel_size//2, i+kernel_size//2+1)])
    return filtered

def ACTIVE_filter(i_ACTIVE_windows, pre_processed_gestures):
    # ACTIVE_filter : delete if the window is not ACTIVE
    list_pre_processed_gestures=pre_processed_gestures.tolist()
    for i_ges in range(len(list_pre_processed_gestures)):
        for i_try in range(len(list_pre_processed_gestures[i_ges])):
            for i_win in reversed(range(len(list_pre_processed_gestures[i_ges][i_try]))):
                if not i_win in range(i_ACTIVE_windows[i_ges][i_try][0], i_ACTIVE_windows[i_ges][i_try][0]+i_ACTIVE_windows[i_ges][i_try][1]):
                    del list_pre_processed_gestures[i_ges][i_try][i_win]
    return np.array(list_pre_processed_gestures)

def Repartition_N_Compute_RMS(ACTIVE_pre_processed_gestures, N=SEGMENT_N):
    # List all the data of each channel without partitioning into windows
    ACTIVE_N_gestures=[[[[] for i_ch in range(len(ACTIVE_pre_processed_gestures[0][0][0]))] for i_try in range(ACTIVE_pre_processed_gestures.shape[1])] for i_ges in range(ACTIVE_pre_processed_gestures.shape[0])]     # CONSTANT
```

```
    for i_ges in range(len(ACTIVE_pre_processed_gestures)):
        for i_try in range(len(ACTIVE_pre_processed_gestures[i_ges])):
            for i_seg in range(len(ACTIVE_pre_processed_gestures[i_ges][i_try])):
                for i_ch in range(len(ACTIVE_pre_processed_gestures[i_ges][i_try][i_seg])):
                    ACTIVE_N_gestures[i_ges][i_try][i_ch].extend(ACTIVE_pre_processed_gestures[i_ges][i_try][i_seg][i_ch])
    # Compute RMS in N large windows
    for i_ges in range(len(ACTIVE_N_gestures)):
        for i_try in range(len(ACTIVE_N_gestures[i_ges])):
            for i_ch in range(len(ACTIVE_N_gestures[i_ges][i_try])):
                RMSs=[]
                for i  in range(N):
                    RMSs.append(compute_RMS(ACTIVE_N_gestures[i_ges][i_try][i_ch][(len(ACTIVE_N_gestures[i_ges][i_try][i_ch])//N)*i:(len(ACTIVE_N_gestures[i_ges][i_try][i_ch])//N)*(i+1)]))
                ACTIVE_N_gestures[i_ges][i_try][i_ch]=np.array(RMSs)
            ACTIVE_N_gestures[i_ges][i_try]=np.array(ACTIVE_N_gestures[i_ges][i_try]).transpose()   # Change (4,10,168,N) -> (4,10,N,168)
    return np.array(ACTIVE_N_gestures)

def mean_normalization(ACTIVE_N_RMS_gestures):
    for i_ges in range(len(ACTIVE_N_RMS_gestures)):
        for i_try in range(len(ACTIVE_N_RMS_gestures[i_ges])):
            for i_Lwin in range(len(ACTIVE_N_RMS_gestures[i_ges][i_try])):
                delta=max(ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin])-min(ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin])
                Mean=np.mean(ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin])
                for i_ch in range(len(ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin])):
                    ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin][i_ch]=(ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin][i_ch]-Mean)/delta
    return ACTIVE_N_RMS_gestures

def construct_X_y(mean_normalized_RMS):
    X=np.reshape(mean_normalized_RMS, (mean_normalized_RMS.shape[0]*mean_normalized_RMS.shape[1]*mean_normalized_RMS.shape[2], mean_normalized_RMS.shape[3]))
    y=np.array([])
    for i_ges in range(mean_normalized_RMS.shape[0]):
        for i in range(mean_normalized_RMS.shape[1]):   # # of tries
            for j in range(mean_normalized_RMS.shape[2]):  # # of Larege windows
                y=np.append(y, [i_ges])
    return X, y

def plot_confusion_matrix(y_test, kinds, y_pred):
    mat = confusion_matrix(y_test, y_pred)
    sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=kinds, yticklabels=kinds)
    plt.xlabel('true label')
    plt.ylabel('predicted label')
    plt.axis('auto')
    plt.show()

def check(x, prin=0):
    print("length: ", len(x))
    print("type: ", type(x))
    if type(x) == type(np.array([])): print("shape: ", x.shape)
    if prin==1: print(x)
    raise ValueError("-------------WORKING LINE--------------")

```

```
216 def check_segment_len(ACTIVE_RMS_gestures):
217     for i in range(len(ACTIVE_RMS_gestures)):
218         print("%d번째 gesture의 각 try의 segment 길이들 : " %i, end='')
219         for j in range(len(ACTIVE_RMS_gestures[i])):
220             print(len(ACTIVE_RMS_gestures[i][j]), end=' ')
221         print()
222 
223 def plot_some_data(gestures):
224     # Choose random three data
225     chose=[]
226     for i in range(3):
227         rand_ges = random.randint(1, len(gestures)-1)    # Except idle gesture
228         rand_try = random.randint(0, len(gestures[rand_ges])-1)
229         rand_win = random.randint(0, len(gestures[rand_ges][rand_try])-1)
230         chose.append((rand_ges, rand_try, rand_win))
231     # Plot
232     y,x=np.meshgrid(range(ACTUAL_RAW),range(ACTUAL_COLUMN))
233     fig, ax = plt.subplots(nrows=3)
234     im=[]
235     for i in range(len(chose)):
236         df = pd.DataFrame({"x":x.flatten(),
"y":y.flatten(),"value":gestures[chose[i][0]][chose[i][1]][chose[i]
[2]].flatten()}).pivot(index="y", columns="x", values="value")
237         im.append(ax[i].imshow(df.values, cmap="viridis", vmin=0, vmax=1))
238         ax[i].set_title("%dth active window in %dth try in %dth gesture" %(chose[i]
[2], chose[i][1], chose[i][0]))
239         fig.colorbar(im[i], ax=ax[i])
240     plt.tight_layout()
241     plt.show()
242 
243 def extract_X_y_for_one_session(gestures):
244     # Signal Pre-processing & Construct windows
245     list_gestures=gestures.tolist()
246     for i_ges in range(len(list_gestures)):
247         for i_try in range(len(list_gestures[i_ges])):
248             for i_ch in range(len(list_gestures[i_ges][i_try][0])):
249                 list_gestures[i_ges][i_try].append(np.array(list_gestures[i_ges]
[i_try][0][i_ch]))
250     gestures=np.delete(np.array(list_gestures), 0, 2)
251 
252     for i_ges in range(gestures.shape[0]):
253         for i_try in range(gestures.shape[1]):
254             # Segmentation : Data processing : Discard useless data
255             gestures[i_ges, i_try, 0]=np.delete(gestures[i_ges, i_try,
0],np.s_[7:192:8],0)
256             # Preprocessing : Apply butterworth band-pass filter
257             gestures[i_ges, i_try, 0]=butter_bandpass_filter(gestures[i_ges, i_try,
0])
258             # Segmentation : Data processing : Divide continuous data into 150
samples window
259             gestures[i_ges, i_try, 0]=np.delete(gestures[i_ges, i_try, 0],
list(range((gestures[i_ges, i_try, 0].shape[1]//WINDOW_SIZE)*WINDOW_SIZE,
gestures[i_ges, i_try, 0].shape[1])), 1)
260             gestures[i_ges, i_try, 0]=np.reshape(gestures[i_ges, i_try, 0],
(gestures[i_ges, i_try, 0].shape[0], gestures[i_ges, i_try,
0].shape[1]//WINDOW_SIZE, WINDOW_SIZE))
261 
262     ################################### FROM HERE
########################################
263     # Segmentation : Compute RMS
```

```
    RMS_gestures=compute_RMS_gestures(gestures)
    # Segmentation : Base normalization
    RMS_gestures=base_normalization(RMS_gestures)
    # Segmentation : Median filtering
    for i_ges in range(len(RMS_gestures)):
        for i_try in range(len(RMS_gestures[i_ges])):
            channels=RMS_gestures[i_ges][i_try].transpose()
            for i_ch in range(len(channels)):
                channels[i_ch]=medfilt(channels[i_ch])
            RMS_gestures[i_ges][i_try]=channels.transpose()
    # Segmentation : Dertermine which window is ACTIVE
    i_ACTIVE_windows=extract_ACTIVE_window_i(RMS_gestures.tolist())

    # Feature extraction : Filter only ACTIVE windows
    ACTIVE_pre_processed_gestures=ACTIVE_filter(i_ACTIVE_windows, pre_processed_gestures)
    # Feature extraction : Partition existing windows into N large windows and compute RMS for each large window
    ACTIVE_N_RMS_gestures=Repartition_N_Compute_RMS(ACTIVE_pre_processed_gestures, SEGMENT_N)
    # Feature extraction : Mean normalization for all channels in each window
    mean_normalized_RMS=mean_normalization(ACTIVE_N_RMS_gestures)

    # Plot one data
    if PLOT_RANDOM_DATA==True:
        plot_some_data(mean_normalized_RMS)

    # Naive Bayes classifier : Construct X and y
    X, y = construct_X_y(mean_normalized_RMS)
    return X, y

def plot_ch(data,i_gest,i_try,i_ch):
    plt.plot(data[i_gest][5][0][89,:])
    plt.show()

def main():
    sessions=load_mat_files("./data/")  # Dict : sessions
    init_session=1
    for session in sessions.values():
        # Input data for each session
        X_session, y_session=extract_X_y_for_one_session(session)
        print("Processing...%d" %(sessions.values().index(session)))
        if init_session==1:
            X=np.array(X_session)
            y=np.array(y_session)
            init_session=0
            continue
        X=np.append(X, X_session, axis=0)
        y=np.append(y, y_session)
    kinds=list(set(y))

    # Naive Bayes classifier : Basic method : NOT LOOCV
    gnb = GaussianNB()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_RATIO, random_state=0)
    y_pred = gnb.fit(X_train, y_train).predict(X_test)
    print("Accuracy : %d%%" % (100-(((y_test != y_pred).sum()/X_test.shape[0])*100)))
    if PLOT_CONFUSION_MATRIX:
        plot_confusion_matrix(y_test, kinds, y_pred)
```

```
319
320 main()
```