```python
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame
from glob import glob
from time import time
from random import randint
from seaborn import heatmap
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
from sklearn.mixture import GaussianMixture
WINDOW_SIZE = 150    # 20:9.76ms, 150:73.2ms
TEST_RATIO = 0.3
SEGMENT_N = 3
ACTUAL_COLUMN=24
ACTUAL_RAW=7

PLOT_PRINT_PROCESSING = False
PRINT_TIME_CONSUMING = True
GMM_CALIBRATE = False
GNB_CLASSIFY = True
PLOT_CONFUSION_MATRIX = True

def load_mat_files(dataDir):
    if PRINT_TIME_CONSUMING: t_load_mat_files=time()
    pathname=dataDir + "/**/*.mat"
    files = 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":
                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":
            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)
    if PRINT_TIME_CONSUMING: print("Loading mat files: %.2f" %(time()-t_load_mat_files))
    return sessions

def plot_a_data(data):
    plt.imshow(data, cmap='hot_r', interpolation='nearest', vmin=0, vmax=0.0035)
    plt.show()

```

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

def base_normalization(RMS_gestures):
    if PRINT_TIME_CONSUMING: t_base_normalization=time()
    # Compute mean value of each channel of idle gesture
    average_channel_idle_gesture=np.mean(np.mean(RMS_gestures[0], 2), 0)
    # Subtract above value from every channel
    if PRINT_TIME_CONSUMING: print("## base_normalization: %.2f" %(time()-t_base_normalization))
    return np.transpose(np.transpose(RMS_gestures,(0,1,3,2))-average_channel_idle_gesture,(0,1,3,2))

def extract_ACTIVE_window_i(RMS_gestures):
    if PRINT_TIME_CONSUMING: t_extract_ACTIVE_window_i=time()
    RMS_gestures=np.transpose(RMS_gestures,(0,1,3,2))
    ## Determine whether ACTIVE : Compute summarized RMS
    sum_RMSs=np.sum(RMS_gestures,3)
    thresholds=np.reshape(np.repeat(np.sum(sum_RMSs,2)/sum_RMSs.shape[2], RMS_gestures.shape[2], axis=1),sum_RMSs.shape)
    ## Determine whether ACTIVE : Determining & Selecting the longest contiguous sequences
    i_ACTIVE_windows=np.zeros((sum_RMSs.shape[:-1]+(2,))).tolist()
    sum_RMSs=sum_RMSs-thresholds
    for i_ges in range(sum_RMSs.shape[0]):
        for i_try in range(sum_RMSs.shape[1]):
            contiguous = 0
            MAX_contiguous = 0
            for i_win in range(sum_RMSs.shape[2]):
                sandwitch=i_win!=0 and i_win!=sum_RMSs.shape[2]-1 and sum_RMSs[i_ges, i_try, i_win-1]>0 and sum_RMSs[i_ges, i_try, i_win+1]>0
                if sum_RMSs[i_ges, i_try, i_win]>0 or sandwitch:
                    if contiguous==0: i_start=i_win
                    contiguous+=1
                    if i_win!=sum_RMSs.shape[2]-1: continue
                if contiguous!=0:
                    if MAX_contiguous<contiguous:
                        MAX_start=i_start
                        MAX_contiguous=contiguous
                    else:
                        contiguous=0
            i_ACTIVE_windows[i_ges][i_try][0]=MAX_start
            i_ACTIVE_windows[i_ges][i_try][1]=MAX_contiguous
    if PRINT_TIME_CONSUMING: print("## extract_ACTIVE_window_i: %.2f" %(time()-t_extract_ACTIVE_window_i))
    return np.array(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
```

```python
        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, gestures):
    # ACTIVE_filter : delete if the window is not ACTIVE
    if PRINT_TIME_CONSUMING: t_ACTIVE_filter=time()
    list_gestures=np.transpose(gestures, (0,1,3,2,4)).tolist()
    for i_ges in range(i_ACTIVE_windows.shape[0]):
        for i_try in range(i_ACTIVE_windows.shape[1]):
            del list_gestures[i_ges][i_try][:i_ACTIVE_windows[i_ges][i_try][0]]
            del list_gestures[i_ges][i_try][i_ACTIVE_windows[i_ges][i_try][0]+i_ACTIVE_windows[i_ges][i_try][1]:]
    if PRINT_TIME_CONSUMING: print("## ACTIVE_filter: %.2f" %(time()-t_ACTIVE_filter))
    return np.array(list_gestures)

def Repartition_N_Compute_RMS(ACTIVE_gestures, N=SEGMENT_N):
    if PRINT_TIME_CONSUMING: t_Repartition_N_Compute_RMS=time()
    # List all the data of each channel without partitioning into windows
    ACTIVE_N_gestures=[[[[] for i_ch in range(len(ACTIVE_gestures[0][0][0]))] for i_try in range(ACTIVE_gestures.shape[1])] for i_ges in range(ACTIVE_gestures.shape[0])]     # CONSTANT
    for i_ges in range(len(ACTIVE_gestures)):
        for i_try in range(len(ACTIVE_gestures[i_ges])):
            for i_seg in range(len(ACTIVE_gestures[i_ges][i_try])):
                for i_ch in range(len(ACTIVE_gestures[i_ges][i_try][i_seg])):
                    ACTIVE_N_gestures[i_ges][i_try][i_ch].extend(ACTIVE_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)
    if PRINT_TIME_CONSUMING: print("## Repartition_N_Compute_RMS: %.2f" %(time()-t_Repartition_N_Compute_RMS))
    return np.array(ACTIVE_N_gestures)

def mean_normalization(ACTIVE_N_RMS_gestures):
    if PRINT_TIME_CONSUMING: t_mean_normalization=time()
    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])
                ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin]=(ACTIVE_N_RMS_gestures[i_ges][i_try][i_Lwin]-Mean)/delta
    if PRINT_TIME_CONSUMING: print("## mean_normalization: %.2f" %(time()-t_mean_normalization))
    return ACTIVE_N_RMS_gestures

def check_segment_len(i_ACTIVE_windows):
```

```python
    for i in range(len(i_ACTIVE_windows)):
        print("%d번째 gesture의 각 try의 segment 길이들 : " %i, end='')
        for j in range(len(i_ACTIVE_windows[i])):
            print(i_ACTIVE_windows[i][j][1], end=' ')
        print()

def plot_some_data(gestures):
    # Choose random three data
    chose=[]
    for i in range(3):
        rand_ges = randint(1, len(gestures)-1)    # Except idle gesture
        rand_try = randint(0, len(gestures[rand_ges])-1)
        rand_win = randint(0, len(gestures[rand_ges][rand_try])-1)
        chose.append((rand_ges, rand_try, rand_win))
    # Plot
    y,x=np.meshgrid(range(ACTUAL_RAW),range(ACTUAL_COLUMN))
    fig, ax = plt.subplots(nrows=3)
    im=[]
    for i in range(len(chose)):
        df = 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")
        im.append(ax[i].imshow(df.values, cmap="viridis", vmin=0, vmax=1))
        ax[i].set_title("%dth active window in %dth try in %dth gesture" %(chose[i][2], chose[i][1], chose[i][0]))
        fig.colorbar(im[i], ax=ax[i])
    plt.tight_layout()
    plt.show()

def plot_some_X_y(X, y):
    # Choose random three data
    chose=[randint(0,len(X)-1) for i in range(10)]
    # Plot
    yy,xx=np.meshgrid(range(ACTUAL_RAW),range(ACTUAL_COLUMN))
    fig, ax = plt.subplots(nrows=10)
    im=[]
    for i in range(len(chose)):
        df = DataFrame({"x":xx.flatten(), "y":yy.flatten(),"value":X[chose[i]].flatten()}).pivot(index="y", columns="x", values="value")
        im.append(ax[i].imshow(df.values, cmap="viridis", vmin=0, vmax=1))
        ax[i].set_title("%d gesture data" %(y[chose[i]]))
        fig.colorbar(im[i], ax=ax[i])
    plt.tight_layout()
    plt.show()

def refined_data_for_one_session(pre_gestures):
    if PRINT_TIME_CONSUMING: t_refined_data_for_one_session=time()
    # Especially for Ref1, data reshaping into one array
    gestures=np.zeros((pre_gestures.shape[0], pre_gestures.shape[1])).tolist() #CONSTANT
    for i_ges in range(len(pre_gestures)):
        for i_try in range(len(pre_gestures[i_ges])):
            gestures[i_ges][i_try]=pre_gestures[i_ges][i_try][0].copy()
    gestures=np.array(gestures)

    # Signal Pre-processing & Construct windows
    ## Segmentation : Data processing : Discard_useless_data
    if PRINT_TIME_CONSUMING: t_Discard_useless_data=time()
    gestures=np.delete(gestures,np.s_[7:192:8],2)
```

```python
    if PRINT_TIME_CONSUMING: print("# Discard_useless_data: %.2f" %(time()-t_Discard_useless_data))
    if PLOT_PRINT_PROCESSING: plot_ch(gestures, 3, 2, 50)
    ## Preprocessing : Apply_butterworth_band_pass_filter
    if PRINT_TIME_CONSUMING: t_Apply_butterworth_band_pass_filter=time()
    gestures=np.transpose(gestures, (0,1,3,2))
    for i_ges in range(len(gestures)):
        for i_try in range(len(gestures[i_ges])):
            for i_time in range(len(gestures[i_ges][i_try])):
                gestures[i_ges, i_try, i_time]=butter_bandpass_filter(gestures[i_ges, i_try, i_time])
    gestures=np.transpose(gestures, (0,1,3,2))
    if PRINT_TIME_CONSUMING: print("# Apply_butterworth_band_pass_filter: %.2f" %(time()-t_Apply_butterworth_band_pass_filter))
    if PLOT_PRINT_PROCESSING: plot_ch(gestures, 3, 2, 50)
    ## Segmentation : Data processing : Divide_continuous_data_into_150_samples_window
    if PRINT_TIME_CONSUMING: t_Divide_continuous_data_into_150_samples_window=time()
    gestures=np.delete(gestures, np.s_[(gestures.shape[3]//WINDOW_SIZE)*WINDOW_SIZE:], 3)
    gestures=np.reshape(gestures,(gestures.shape[0], gestures.shape[1], gestures.shape[2], gestures.shape[3]//WINDOW_SIZE, WINDOW_SIZE))
    if PRINT_TIME_CONSUMING: print("# Divide_continuous_data_into_150_samples_window: %.2f" %(time()-t_Divide_continuous_data_into_150_samples_window))

    # Determine ACTIVE windows
    ## Segmentation : Compute_RMS
    if PRINT_TIME_CONSUMING: t_Compute_RMS=time()
    RMS_gestures=gestures.copy()
    RMS_gestures=np.apply_along_axis(compute_RMS, 4, RMS_gestures)
    if PRINT_TIME_CONSUMING: print("# Compute_RMS: %.2f" %(time()-t_Compute_RMS))
    ## Segmentation : Base normalization
    if PLOT_PRINT_PROCESSING: plot_a_data(RMS_gestures[3,2])
    RMS_gestures=base_normalization(RMS_gestures)
    if PLOT_PRINT_PROCESSING: plot_a_data(RMS_gestures[3,2])
    ## Segmentation : Median_filtering
    if PRINT_TIME_CONSUMING: t_Median_filtering=time()
    RMS_gestures=np.apply_along_axis(medfilt, 3, RMS_gestures)
    if PRINT_TIME_CONSUMING: print("# Median filtering: %.2f" %(time()-t_Median_filtering))
    if PLOT_PRINT_PROCESSING: plot_a_data(RMS_gestures[3,2])
    ## Segmentation : Dertermine which window is ACTIVE
    i_ACTIVE_windows=extract_ACTIVE_window_i(RMS_gestures)

    # Feature extraction : Filter only ACTIVE windows
    ACTIVE_gestures=ACTIVE_filter(i_ACTIVE_windows, 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_gestures)
    # Feature extraction : Mean normalization for all channels in each window
    mean_normalized_RMS=mean_normalization(ACTIVE_N_RMS_gestures)

    # Plot one data
    if PLOT_PRINT_PROCESSING: plot_some_data(mean_normalized_RMS)
    if PRINT_TIME_CONSUMING: print("#refined_data_for_one_session: %.2f\n" %(time()-t_refined_data_for_one_session))
    return mean_normalized_RMS

def plot_ch(data,i_gest,i_try=2,i_ch=50):
```

```
    plt.plot(data[i_gest][i_try][i_ch,:])
    plt.show()

def plot_confusion_matrix(y_test, kinds, y_pred):
    mat = confusion_matrix(y_test, y_pred)
    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 construct_X_y(refined_data):
    if PRINT_TIME_CONSUMING: t_construct_X_y=time()
    X=np.reshape(refined_data,
(refined_data.shape[0]*refined_data.shape[1]*refined_data.shape[2]*refined_data.shap
e[3], refined_data.shape[4]))
    y=np.array([])
    for i in range(refined_data.shape[0]):  # # of sessions
        for i_ges in range(refined_data.shape[1]):
            for j in range(refined_data.shape[2]):   # # of tries
                for k in range(refined_data.shape[3]):  # # of Larege windows
                    y=np.append(y, [i_ges])
    if PRINT_TIME_CONSUMING: print("## construct_X_y: %.2f" %(time()-
t_construct_X_y))
    return X, y

def gnb_classifier(refined_data):
    if PRINT_TIME_CONSUMING: t_gnb_classifier=time()
    # Construct X and y
    X, y = construct_X_y(refined_data)
    if PLOT_PRINT_PROCESSING: plot_some_X_y(X, y)
    # Classifying
    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 PRINT_TIME_CONSUMING: print("#gnb_classifier: %.2f" %(time()-
t_gnb_classifier))
    if PLOT_CONFUSION_MATRIX: plot_confusion_matrix(y_test, list(set(y)), y_pred)

def gmm_calibration(refined_data):
    if PRINT_TIME_CONSUMING: t_gmm_calibration=time()
    """
    #interpolate
    y,x=np.meshgrid(range(ACTUAL_RAW),range(ACTUAL_COLUMN))
    interpolated_X=[]
    for i_session in range(X.shape[0]):
        interpolated_X.append([])
        for i_data in range(X.shape[1]):
            interpolated_X[-1].append(interp2d(y,x,X[i_session,
i_data],kind='cubic'))
    if PLOT_PRINT_PROCESSING: plot_a_data(X[0,130])

    gmm = GaussianMixture(n_components=2).fit(X)
    print(gmm)
    probs = gmm.predict_proba(X)
    print(probs[:5].round(3))
```

```python
    """
    if PRINT_TIME_CONSUMING: print("#gmm_calibration: %.2f" %(time()-
t_gmm_calibration))

def main():
    if PRINT_TIME_CONSUMING: t_main=time()
    sessions=load_mat_files("./data/")  # Dict : sessions
    init_session=1
    for session in sessions.values():
        # Input data for each session
        refined_data_session=refined_data_for_one_session(session)
        if init_session==1:
            refined_data=np.array([refined_data_session])
            init_session=0
            continue
        refined_data=np.append(refined_data, [refined_data_session], axis=0)

    # Calibraion : GMM method
    if GMM_CALIBRATE: gmm_calibration(refined_data)
    # Naive Bayes classifier : Basic method : NOT LOOCV
    if GNB_CLASSIFY: gnb_classifier(refined_data)
    if PRINT_TIME_CONSUMING: print("main: %.2f" %(time()-t_main))

main()
```