```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pywt
import os
import datetime

from load_files import getInputLoadFile, get_user_input
from ArtifactClassifiers import predict_binary_classifier, predict_multiclass_classifier

matplotlib.rcParams['ps.useafm'] = True
matplotlib.rcParams['pdf.use14corefonts'] = True
matplotlib.rcParams['text.usetex'] = True



def getWaveletData(data):
    '''
    This function computes the wavelet coefficients

    INPUT:
        data:          DataFrame, index is a list of timestamps at 8Hz, columns include EDA, filtered_eda

    OUTPUT:
        wave1Second:    DateFrame, index is a list of timestamps at 1Hz, columns include
OneSecond_feature1, OneSecond_feature2, OneSecond_feature3
        waveHalfSecond: DateFrame, index is a list of timestamps at 2Hz, columns include
HalfSecond_feature1, HalfSecond_feature2
    '''
    startTime = data.index[0]

    # Create wavelet dataframes
```

```
    oneSecond = pd.date_range(start=startTime, periods=len(data), freq='1s')
    halfSecond = pd.date_range(start=startTime, periods=len(data), freq='500L')

    # Compute wavelets
    cA_n, cD_3, cD_2, cD_1 = pywt.wavedec(data['EDA'], 'Haar', level=3) #3 = 1Hz, 2 = 2Hz, 1=4Hz

    # Wavelet 1 second window
    N = int(len(data)/8)
    coeff1 = np.max(abs(np.reshape(cD_1[0:4*N],(N,4))), axis=1)
    coeff2 = np.max(abs(np.reshape(cD_2[0:2*N],(N,2))), axis=1)
    coeff3 = abs(cD_3[0:N])
    wave1Second =
pd.DataFrame({'OneSecond_feature1':coeff1,'OneSecond_feature2':coeff2,'OneSecond_feature3':c
oeff3})
    wave1Second.index = oneSecond[:len(wave1Second)]

    # Wavelet Half second window
    N = int(np.floor((len(data)/8.0)*2))
    coeff1 = np.max(abs(np.reshape(cD_1[0:2*N],(N,2))),axis=1)
    coeff2 = abs(cD_2[0:N])
    waveHalfSecond = pd.DataFrame({'HalfSecond_feature1':coeff1,'HalfSecond_feature2':coeff2})
    waveHalfSecond.index = halfSecond[:len(waveHalfSecond)]

    return wave1Second,waveHalfSecond


def getDerivatives(eda):
    deriv = (eda[1:-1] + eda[2:])/ 2. - (eda[1:-1] + eda[:-2])/ 2.
    second_deriv = eda[2:] - 2*eda[1:-1] + eda[:-2]
    return deriv,second_deriv
```

```
def getDerivStats(eda):
    deriv, second_deriv = getDerivatives(eda)
    maxd = max(deriv)
    mind = min(deriv)
    maxabsd = max(abs(deriv))
    avgabsd = np.mean(abs(deriv))
    max2d = max(second_deriv)
    min2d = min(second_deriv)
    maxabs2d = max(abs(second_deriv))
    avgabs2d = np.mean(abs(second_deriv))

    return maxd,mind,maxabsd,avgabsd,max2d,min2d,maxabs2d,avgabs2d


def getStats(data):
    eda = data['EDA'].values
    filt = data['filtered_eda'].values
    maxd,mind,maxabsd,avgabsd,max2d,min2d,maxabs2d,avgabs2d = getDerivStats(eda)
    maxd_f,mind_f,maxabsd_f,avgabsd_f,max2d_f,min2d_f,maxabs2d_f,avgabs2d_f =
getDerivStats(filt)
    amp = np.mean(eda)
    amp_f = np.mean(filt)
    return amp,
maxd,mind,maxabsd,avgabsd,max2d,min2d,maxabs2d,avgabs2d,amp_f,maxd_f,mind_f,maxabsd_f,a
vgabsd_f,max2d_f,min2d_f,maxabs2d_f,avgabs2d_f


def computeWaveletFeatures(waveDF):
    maxList = waveDF.max().tolist()
    meanList = waveDF.mean().tolist()
    stdList = waveDF.std().tolist()
    medianList = waveDF.median().tolist()
```

```
    aboveZeroList = (waveDF[waveDF>0]).count().tolist()


    return maxList,meanList,stdList,medianList,aboveZeroList



def getWavelet(wave1Second,waveHalfSecond):
    max_1,mean_1,std_1,median_1,aboveZero_1 = computeWaveletFeatures(wave1Second)
    max_H,mean_H,std_H,median_H,aboveZero_H = computeWaveletFeatures(waveHalfSecond)
    return
max_1,mean_1,std_1,median_1,aboveZero_1,max_H,mean_H,std_H,median_H,aboveZero_H



def getFeatures(data,w1,wH):
    # Get DerivStats

amp,maxd,mind,maxabsd,avgabsd,max2d,min2d,maxabs2d,avgabs2d,amp_f,maxd_f,mind_f,maxabs
d_f,avgabsd_f,max2d_f,min2d_f,maxabs2d_f,avgabs2d_f = getStats(data)
    statFeat =
np.hstack([amp,maxd,mind,maxabsd,avgabsd,max2d,min2d,maxabs2d,avgabs2d,amp_f,maxd_f,min
d_f,maxabsd_f,avgabsd_f,max2d_f,min2d_f,maxabs2d_f,avgabs2d_f])


    # Get Wavelet Features
    max_1,mean_1,std_1,median_1,aboveZero_1,max_H,mean_H,std_H,median_H,aboveZero_H =
getWavelet(w1,wH)
    waveletFeat =
np.hstack([max_1,mean_1,std_1,median_1,aboveZero_1,max_H,mean_H,std_H,median_H,aboveZe
ro_H])


    all_feat = np.hstack([statFeat,waveletFeat])


    if np.Inf in all_feat:
        print("Inf")


    if np.NaN in all_feat:
```

```
        print("NaN")


    return list(all_feat)



def createFeatureDF(data):
    '''
    INPUTS:
        filepath:          string, path to input file
    OUTPUTS:
        features:          DataFrame, index is a list of timestamps for each 5 seconds, contains all the features
        data:              DataFrame, index is a list of timestamps at 8Hz, columns include AccelZ, AccelY, AccelX, Temp, EDA, filtered_eda
    '''
    # Load data from q sensor
    wave1sec,waveHalf = getWaveletData(data)


    # Create 5 second timestamp list
    timestampList = data.index.tolist()[0::40]


    # feature names for DataFrame columns
    allFeatureNames = ['raw_amp','raw_maxd','raw_mind','raw_maxabsd','raw_avgabsd','raw_max2d','raw_min2d','raw_maxabs2d','raw_avgabs2d','filt_amp','filt_maxd','filt_mind',

'filt_maxabsd','filt_avgabsd','filt_max2d','filt_min2d','filt_maxabs2d','filt_avgabs2d','max_1s_1','max_1s_2','max_1s_3','mean_1s_1','mean_1s_2','mean_1s_3',

'std_1s_1','std_1s_2','std_1s_3','median_1s_1','median_1s_2','median_1s_3','aboveZero_1s_1','aboveZero_1s_2','aboveZero_1s_3','max_Hs_1','max_Hs_2','mean_Hs_1',

'mean_Hs_2','std_Hs_1','std_Hs_2','median_Hs_1','median_Hs_2','aboveZero_Hs_1','aboveZero_Hs_2']
```

```
    # Initialize Feature Data Frame
    features = pd.DataFrame(np.zeros((len(timestampList),len(allFeatureNames))),columns=allFeatureNames,index=timestampList)

    # Compute features for each 5 second epoch
    for i in range(len(features)-1):
        start = features.index[i]
        end = features.index[i+1]
        this_data = data[start:end]
        this_w1 = wave1sec[start:end]
        this_w2 = waveHalf[start:end]
        features.iloc[i] = getFeatures(this_data,this_w1,this_w2)
    return features


def classifyEpochs(features,featureNames,classifierName):
    '''
    This function takes the full features DataFrame and classifies each 5 second epoch into artifact, questionable, or clean

    INPUTS:
        features:          DataFrame, index is a list of timestamps for each 5 seconds, contains all the features
        featureNames:      list of Strings, subset of feature names needed for classification
        classifierName:    string, type of SVM (binary or multiclass)

    OUTPUTS:
        labels:            Series, index is a list of timestamps for each 5 seconds, values of -1, 0, or 1 for artifact, questionable, or clean
    '''
    # Only get relevant features
```

```
    features = features[featureNames]
    X = features[featureNames].values

    # Classify each 5 second epoch and put into DataFrame
    if 'Binary' in classifierName:
        featuresLabels = predict_binary_classifier(X)
    elif 'Multi' in classifierName:
        featuresLabels = predict_multiclass_classifier(X)

    return featuresLabels



def getSVMFeatures(key):
    '''
    This returns the list of relevant features

    INPUT:
        key:            string, either "Binary" or "Multiclass"

    OUTPUT:
        featureList:    list of Strings, subset of feature names needed for classification
    '''
    if key == "Binary":
        return
['raw_amp','raw_maxabsd','raw_max2d','raw_avgabs2d','filt_amp','filt_min2d','filt_maxabs2d','max
_1s_1',
                        'mean_1s_1','std_1s_1','std_1s_2','std_1s_3','median_1s_3']
    elif key == "Multiclass":
        return
['filt_maxabs2d','filt_min2d','std_1s_1','raw_max2d','raw_amp','max_1s_1','raw_maxabs2d','raw_av
gabs2d',
                          'filt_max2d','filt_amp']
```

```
    else:
        print('Error!! Invalid key, choose "Binary" or "Multiclass"\n\n')
        return


def classify(classifierList):
    '''
    This function wraps other functions in order to load, classify, and return the label for each 5 second epoch of Q sensor data.

    INPUT:
        classifierList:        list of strings, either "Binary" or "Multiclass"
    OUTPUT:
        featureLabels:         Series, index is a list of timestamps for each 5 seconds, values of -1, 0, or 1 for artifact, questionable, or clean
        data:                  DataFrame, only output if fullFeatureOutput=1, index is a list of timestamps at 8Hz, columns include AccelZ, AccelY, AccelX, Temp, EDA, filtered_eda
    '''
    # Constants
    oneHour = 8*60*60 # 8(samp/s)*60(s/min)*60(min/hour) = samp/hour
    fiveSec = 8*5

    # Load data
    data, _ = getInputLoadFile()

    # Get pickle List and featureNames list
    featureNameList = [[]]*len(classifierList)
    for i in range(len(classifierList)):
        featureNames = getSVMFeatures(classifierList[i])
        featureNameList[i]=featureNames

    # Get the number of data points, hours, and labels
```

```
    rows = len(data)
    num_labels = int(np.ceil(float(rows)/fiveSec))
    hours = int(np.ceil(float(rows)/oneHour))

    # Initialize labels array
    labels = -1*np.ones((num_labels,len(classifierList)))

    for h in range(hours):
        # Get a data slice that is at most 1 hour long
        start = h*oneHour
        end = min((h+1)*oneHour,rows)
        cur_data = data[start:end]

        features = createFeatureDF(cur_data)

        for i in range(len(classifierList)):
            # Get correct feature names for classifier
            classifierName = classifierList[i]
            featureNames = featureNameList[i]

            # Label each 5 second epoch
            temp_labels = classifyEpochs(features, featureNames, classifierName)
            labels[(h*12*60):(h*12*60+temp_labels.shape[0]),i] = temp_labels

    return labels,data


def plotData(data,labels,classifierList,filteredPlot=0,secondsPlot=0):
    '''
    This function plots the Q sensor EDA data with shading for artifact (red) and questionable data
(grey).
```

```
    Note that questionable data will only appear if you choose a multiclass classifier

  INPUT:

    data:              DataFrame, indexed by timestamps at 8Hz, columns include EDA and
filtered_eda

    labels:            array, each row is a 5 second period and each column is a different classifier

    filteredPlot:      binary, 1 for including filtered EDA in plot, 0 for only raw EDA on the plot,
defaults to 0

    secondsPlot:       binary, 1 for x-axis in seconds, 0 for x-axis in minutes, defaults to 0


  OUTPUT:

    [plot]             the resulting plot has N subplots (where N is the length of classifierList) that
have linked x and y axes

                       and have shading for artifact (red) and questionable data (grey)


  '''


  # Initialize x axis

  if secondsPlot:

    scale = 1.0

  else:

    scale = 60.0

  time_m = np.arange(0,len(data))/(8.0*scale)


  # Initialize Figure

  plt.figure(figsize=(10,5))


  # For each classifier, label each epoch and plot

  for k in range(np.shape(labels)[1]):

    key = classifierList[k]


    # Initialize Subplots
```

```
    if k==0:
        ax = plt.subplot(len(classifierList),1,k+1)
    else:
        ax = plt.subplot(len(classifierList),1,k+1,sharex=ax,sharey=ax)

    # Plot EDA
    ax.plot(time_m,data['EDA'])

    # For each epoch, shade if necessary
    for i in range(0,len(labels)-1):
        if labels[i,k]==-1:
            # artifact
            start = i*40/(8.0*scale)
            end = start+5.0/scale
            ax.axvspan(start, end, facecolor='red', alpha=0.7, edgecolor ='none')
        elif labels[i,k]==0:
            # Questionable
            start = i*40/(8.0*scale)
            end = start+5.0/scale
            ax.axvspan(start, end, facecolor='.5', alpha=0.5,edgecolor ='none')

    # Plot filtered data if requested
    if filteredPlot:
        ax.plot(time_m-.625/scale,data['filtered_eda'], c='g')
        plt.legend(['Raw SC','Filtered SC'],loc=0)

    # Label and Title each subplot
    plt.ylabel('$\mu$S')
    plt.title(key)

# Only include x axis label on final subplot
```

```
    if secondsPlot:
        plt.xlabel('Time (s)')
    else:
        plt.xlabel('Time (min)')

    # Display the plot
    plt.subplots_adjust(hspace=.3)
    plt.show()
    return


if __name__ == "__main__":
    numClassifiers = int(get_user_input('Would you like 1 classifier (Binary or Multiclass) or both (enter 1 or 2): '))

    # Create list of classifiers
    if numClassifiers==1:
        temp_clf = int(get_user_input("Select a classifier:\n1: Binary\n2: Multiclass\n:"))
        while temp_clf != 1 and temp_clf !=2:
            temp_clf = get_user_input("Something went wrong. Enter the number 1 or 2.\n Select a classifier:\n1: Binary\n2: Multiclass):")
        if temp_clf == 1:
            print('Binary Classifier selected')
            classifierList = ['Binary']
        elif temp_clf == 2:
            print('Multiclass Classifier selected')
            classifierList = ['Multiclass']
    else:
        classifierList = ['Binary', 'Multiclass']

    # Classify the data
    labels, data = classify(classifierList)
```

```
    # Plotting the data
    plotDataInput = get_user_input('Do you want to plot the labels? (y/n): ')

    if plotDataInput=='y':
        # Include filter plot?
        filteredPlot = get_user_input('Would you like to include filtered data in your plot? (y/n): ')
        if filteredPlot=='y':
            filteredPlot=1
        else:
            filteredPlot=0

        # X axis in seconds?
        secondsPlot = get_user_input('Would you like the x-axis to be in seconds or minutes? (sec/min):
')
        if secondsPlot=='sec':
            secondsPlot=1
        else:
            secondsPlot=0

        # Plot Data
        plotData(data,labels,classifierList,filteredPlot,secondsPlot)

        print("Remember! Red is for epochs with artifact, grey is for epochs that are questionable, and
no shading is for clean epochs")

    # Saving the data
    saveDataInput = get_user_input('Do you want to save the labels? (y/n): ')

    if saveDataInput=='y':
        outputPath = get_user_input('Output directory: ')
```

```
    outputLabelFilename= get_user_input('Output filename: ')

    # Save labels
    fullOutputPath = os.path.join(outputPath,outputLabelFilename)
    if fullOutputPath[-4:] != '.csv':
       fullOutputPath = fullOutputPath+'.csv'

    featureLabels = pd.DataFrame(labels, index=pd.date_range(start=data.index[0],
periods=len(labels), freq='5s'),
                                 columns=classifierList)

    featureLabels.reset_index(inplace=True)
    featureLabels.rename(columns={'index':'StartTime'}, inplace=True)
    featureLabels['EndTime'] = featureLabels['StartTime']+datetime.timedelta(seconds=5)
    featureLabels.index.name = 'EpochNum'

    cols = ['StartTime', 'EndTime']
    cols.extend(classifierList)

    featureLabels = featureLabels[cols]
    featureLabels.rename(columns={'Binary': 'BinaryLabels', 'Multiclass': 'MulticlassLabels'},
                         inplace=True)

    featureLabels.to_csv(fullOutputPath)

    print("Labels saved to " + fullOutputPath)
    print("Remember! The first column is timestamps and the second column is the labels (-1 for
artifact, 0 for questionable, 1 for clean)")

  print('--------------------------------')
  print("Please also cite this project:")
```

```
    print("Taylor, S., Jaques, N., Chen, W., Fedor, S., Sano, A., & Picard, R. Automatic identification of artifacts in electrodermal activity data. In Engineering in Medicine and Biology Conference. 2015")

    print('---------------------------------')
```