

unit_test

September 24, 2020

1 Test Your Algorithm

1.1 Instructions

1. From the **Pulse Rate Algorithm** Notebook you can do one of the following:
 - Copy over all the **Code** section to the following Code block.
 - Download as a Python (.py) and copy the code to the following Code block.
2. In the bottom right, click the Test Run button.

1.1.1 Didn't Pass

If your code didn't pass the test, go back to the previous Concept or to your local setup and continue iterating on your algorithm and try to bring your training error down before testing again.

1.1.2 Pass

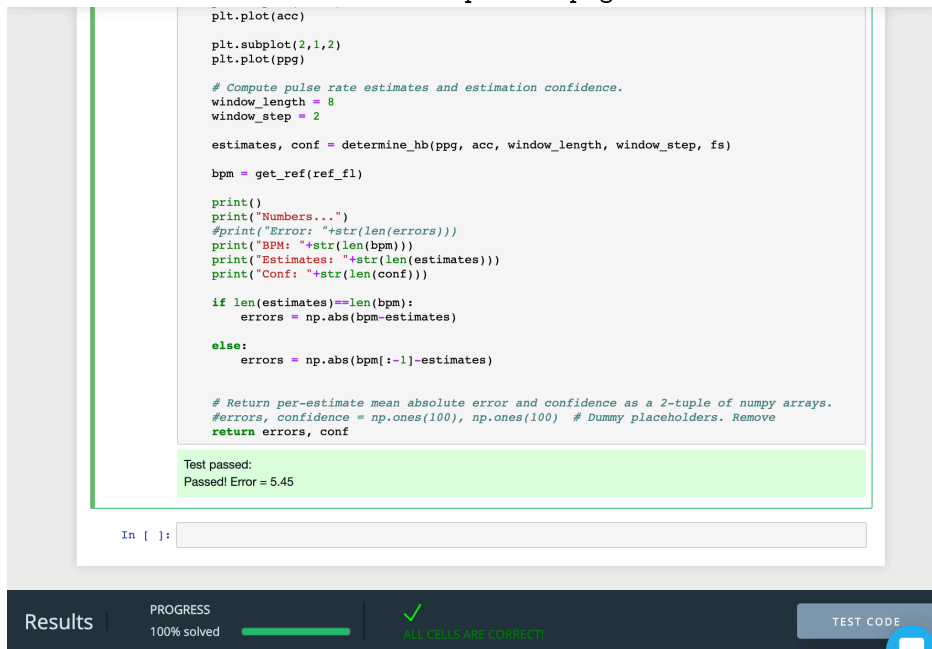
If your code passes the test, complete the following! You **must** include a screenshot of your code and the Test being **Passed**. Here is what the starter filler code looks like when the test is run and should be similar. A passed test will include in the notebook a green outline plus a box with **Test passed:** and in the Results bar at the bottom the progress bar will be at 100% plus a checkmark with



All cells passed.

1. Take a screenshot of your code passing the test, make sure it is in the format .png. If not a .png image, you will have to edit the Markdown render the image after Step 3. Here is an example of what the passed.png would look like
2. Upload the screenshot to the same folder or directory as this jupyter notebook.

3. Rename the screenshot to passed.png and it should show up below.



```

plt.plot(acc)

plt.subplot(2,1,2)
plt.plot(ppg)

# Compute pulse rate estimates and estimation confidence.
window_length = 8
window_step = 2

estimates, conf = determine_hb(ppg, acc, window_length, window_step, fs)

bpm = get_ref(ref_fl)

print()
print("Numbers...")
#print("Error: "+str(len(errors)))
print("BPM: "+str(len(bpm)))
print("Estimates: "+str(len(estimates)))
print("Conf: "+str(len(conf)))


if len(estimates)!=len(bpm):
    errors = np.abs(bpm-estimates)
else:
    errors = np.abs(bpm[:-1]-estimates)

# Return per-estimate mean absolute error and confidence as a 2-tuple of numpy arrays.
#errors, confidence = np.ones(100), np.ones(100) # Dummy placeholders. Remove
return errors, conf

Test passed:
Passed! Error = 5.45

```

In []:

Results | PROGRESS 100% solved |  ALL TESTS PASSED | TEST CODE

4. Download this jupyter notebook as a .pdf file.
5. Continue to Part 2 of the Project.

In [1]: *# replace the code below with your pulse rate algorithm.*

"""

import numpy as np
import scipy as sp
import scipy.io

def RunPulseRateAlgorithm(data_fl, ref_fl):
 ref = sp.io.loadmat(ref_fl)['BPM0'].reshape(-1)
 sample_idx = np.array([0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000])
 pr_conf = np.array([0.021337153215872807, 0.024064924996852304, 0.02463020949150536,
 pr_est = np.array([49.93333333333333, 49.96666666666667, 49.93333333333333, 49.93333333333333,
 *ref_idx = np.cumsum(np.ones(len(ref)) * 125 * 2) - 125 * 2*
 *return pr_est[:len(ref)] * 60 - ref, pr_conf[:len(ref)]*

"""

import glob

import numpy as np
import scipy as sp
import scipy.io
import matplotlib.pyplot as plt
import scipy.signal

def LoadTroikaDataset():

```

"""
Retrieve the .mat filenames for the troika dataset.

Review the README in ./datasets/troika/ to understand the organization of the .mat files.

Returns:
    data_fls: Names of the .mat files that contain signal data
    ref_fls: Names of the .mat files that contain reference data
    <data_fls> and <ref_fls> are ordered correspondingly, so that ref_fls[5] is the
        reference data for data_fls[5], etc...
"""
data_dir = "./datasets/troika/training_data"
data_fls = sorted(glob.glob(data_dir + "/DATA_*.mat"))
ref_fls = sorted(glob.glob(data_dir + "/REF_*.mat"))
return data_fls, ref_fls

def LoadTroikaDataFile(data_fl):
    """
    Loads and extracts signals from a troika data file.

    Usage:
        data_fls, ref_fls = LoadTroikaDataset()
        ppg, accx, accy, accz = LoadTroikaDataFile(data_fls[0])

    Args:
        data_fl: (str) filepath to a troika .mat file.

    Returns:
        numpy arrays for ppg, accx, accy, accz signals.
    """
    data = sp.io.loadmat(data_fl)['sig']
    return data[2:]

def AggregateErrorMetric(pr_errors, confidence_est):
    """
    Computes an aggregate error metric based on confidence estimates.

    Computes the MAE at 90% availability.

    Args:
        pr_errors: a numpy array of errors between pulse rate estimates and corresponding
            reference heart rates.
        confidence_est: a numpy array of confidence estimates for each pulse rate
            error.

    Returns:
        the MAE at 90% availability
    """

```

```

    # Higher confidence means a better estimate. The best 90% of the estimates
    # are above the 10th percentile confidence.
    percentile90_confidence = np.percentile(confidence_est, 10)

    # Find the errors of the best pulse rate estimates
    best_estimates = pr_errors[confidence_est >= percentile90_confidence]

    # Return the mean absolute error
    return np.mean(np.abs(best_estimates))

#Added/ Edited functions

def BandpassFilter(signal, pass_band, fs):
    """Bandpass Filter.

    Args:
    signal: (np.array) The input signal
    pass_band: (tuple) The pass band. Frequency components outside
    the two elements in the tuple will be removed.
    fs: (number) The sampling rate of <signal>

    Returns:
    (np.array) The filtered signal
    """

    b, a = sp.signal.butter(3, pass_band, btype='bandpass', fs=fs)
    return sp.signal.filtfilt(b, a, signal)

def fourier(signal, fs):
    freqs = np.fft.rfftfreq(len(signal), 1/fs)
    fft = np.abs(np.fft.rfft(signal))

    # Filter data
    fft[freqs <= 70/60.0] = 0.0
    fft[freqs >= 190/60.0] = 0.0

    return fft, freqs

def calculate_confidence(freqs, fft_mag, bpm_f):
    window_f = 30/60
    fundamental_freq_window = (freqs > bpm_f - window_f) & (freqs < bpm_f + window_f)
    return np.sum(fft_mag[fundamental_freq_window]) / np.sum(fft_mag)

def determine_hb(ppg, acc, window_length, window_step, fs):

    window_length_fs = window_length*fs
    window_step_fs = window_step*fs

```

```

est_bpm_list = []
confidence_list = []

prev_est = 40/60

for i in range(0, len(ppg)-window_length_fs, window_step_fs):

    #print("i: "+str(i))
    #for each window...
    ppg_window = ppg[i:i+window_length_fs]
    acc_window = acc[i:i+window_length_fs]

    #Get fourier transforms for ppg and acc
    fft_ppg, freqs_ppg = fourier(ppg_window, fs)
    fft_acc, freqs_acc = fourier(acc_window, fs)

    #get max
    freq_max_ppg = freqs_ppg[np.argmax(fft_ppg)]
    freq_max_acc = freqs_acc[np.argmax(fft_acc)]

    #ppg
    plt.subplot(4,1,1)
    plt.title("PPG")
    plt.plot(freqs_ppg, fft_ppg)
    plt.xlabel(' ')
    plt.xlim(0,10)
    plt.ylabel('FFT Magnitude')

    #acc
    plt.subplot(4,1,2)
    plt.title("ACC")
    plt.plot(freqs_acc, fft_acc)
    plt.xlabel(' ')
    plt.xlim(0,10)
    plt.ylabel('FFT Magnitude')

    k = 1

    while np.abs(freq_max_acc-freq_max_ppg) <= 0.2 and k <=2:
        k+=1
        freq_max_ppg = freqs_ppg[np.argsort(fft_ppg, axis=0)[-k]]
        freq_max_acc = freqs_acc[np.argsort(fft_acc, axis=0)[-k]]

    est_bpm = freq_max_ppg
    prev_est = est_bpm
    est_bpm_list.append(est_bpm*60) #multiplied by 60

```

```

        #Calculate and append confidence
        confidence = calculate_confidence(freqs_ppg, fft_ppg, est_bpm)
        confidence_list.append(confidence)

    return est_bpm_list, confidence_list

def get_ref(ref_fl):

    ref = sp.io.loadmat(ref_fl)['BPM0'].reshape(-1)
    return ref

def Evaluate():
    """
    Top-level function evaluation function.

    Runs the pulse rate algorithm on the Troika dataset and returns an aggregate error m

    Returns:
        Pulse rate error on the Troika dataset. See AggregateErrorMetric.
    """
    # Retrieve dataset files
    data_fls, ref_fls = LoadTroikaDataset()
    errs, confs = [], []
    for data_fl, ref_fl in zip(data_fls, ref_fls):
        # Run the pulse rate algorithm on each trial in the dataset
        errors, confidence = RunPulseRateAlgorithm(data_fl, ref_fl)
        errs.append(errors)
        confs.append(confidence)
        # Compute aggregate error metric
    errs = np.hstack(errs)
    confs = np.hstack(confs)
    return AggregateErrorMetric(errs, confs)

def RunPulseRateAlgorithm(data_fl, ref_fl):

    # Load data using LoadTroikaDataFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)

    # fs
    fs=125

    #Bandpass the data
    pass_band = (40/60., 240/60.)
    ppg = BandpassFilter(ppg, pass_band, fs)
    accx = BandpassFilter(accx, pass_band, fs)
    accy = BandpassFilter(accy, pass_band, fs)
    accz = BandpassFilter(accz, pass_band, fs)

```

```

acc = np.sqrt(accx**2 + accy**2 + accz**2)

print(len(acc[acc>1]))
print(len(acc))
print(acc.shape)

#plot for trial
plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
plt.plot(acc)

plt.subplot(2,1,2)
plt.plot(ppg)

# Compute pulse rate estimates and estimation confidence.
window_length = 8
window_step = 2

estimates, conf = determine_hb(ppg, acc, window_length, window_step, fs)

bpm = get_ref(ref_fl)

print()
print("Numbers...")
#print("Error: "+str(len(errors)))
print("BPM: "+str(len(bpm)))
print("Estimates: "+str(len(estimates)))
print("Conf: "+str(len(conf)))

if len(estimates)==len(bpm):
    errors = np.abs(bpm-estimates)

else:
    errors = np.abs(bpm[:-1]-estimates)

# Return per-estimate mean absolute error and confidence as a 2-tuple of numpy array
#errors, confidence = np.ones(100), np.ones(100) # Dummy placeholders. Remove
return errors, conf

```

In []: