unit_test

September 14, 2020

1 Test Your Algorithm

1.1 Instructions

1. From the **Pulse Rate Algorithm** Notebook you can do one of the following:

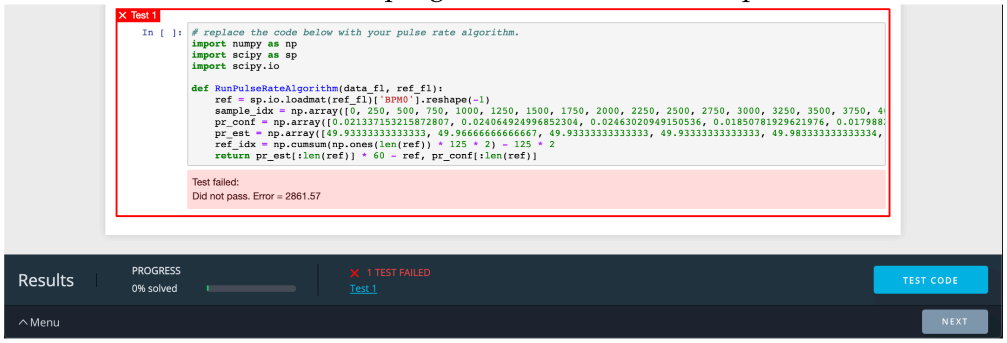
- Copy over all the **Code** section to the following Code block.
- Download as a Python (`.py`) and copy the code to the following Code block.

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 

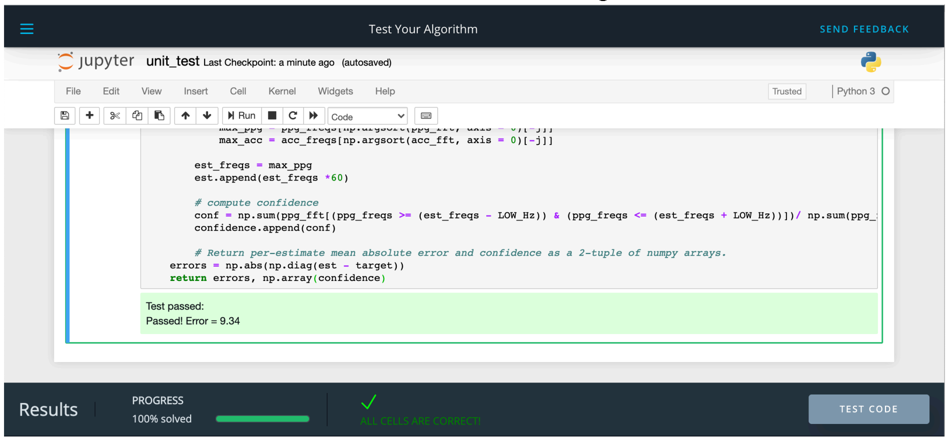
```
In [ ]: # replace the code below with your pulse rate algorithm.
        import numpy as np
        import scipy as sp
        import scipy.io

        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, 3250, 3500, 3750, 4
            pr_conf = np.array([0.021337153215872807, 0.024064924996852304, 0.02463020949150536, 0.018507819296219 76, 0.017988
            pr_est = np.array([49.93333333333333, 49.96666666666667, 49.93333333333333, 49.93333333333333, 49.983333333333334,
            ref_idx = np.cumsum(np.ones(len(ref)) * 125 * 2) - 125 * 2
            return pr_est[:len(ref)] * 60 - ref, pr_conf[:len(ref)]
```

```
Test failed:
Did not pass. Error = 2861.57
```

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.


Test Your Algorithm
SEND FEEDBACK
jupyter unit_test Last Checkpoint: a minute ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
            max_acc = acc_freqs[np.argsort(acc_fft, axis = 0)[-j]]

        est_freqs = max_ppg
        est.append(est_freqs *60)

        # compute confidence
        conf = np.sum(ppg_fft[(ppg_freqs >= (est_freqs - LOW_Hz)) & (ppg_freqs <= (est_freqs + LOW_Hz))])/ np.sum(ppg_
        confidence.append(conf)

        # Return per-estimate mean absolute error and confidence as a 2-tuple of numpy arrays.
    errors = np.abs(np.diag(est - target))
    return errors, np.array(confidence)
```

```
Test passed:
Passed! Error = 9.34
```

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
import glob

import pandas as pd
from matplotlib import pyplot as plt

from scipy.io import loadmat
import scipy.signal

%matplotlib inline

# set the sampling rate to 125Hz
fs = 125

# use the 40-240BPM range to create your pass band
LOW_BPM, HIGH_BPM = 40, 240
LOW_Hz, HIGH_Hz = LOW_BPM/60 , HIGH_BPM/60

# window sizes in seconds:
window_length_s  = 8
window_shift_s = 2

# transform to data points:
window_length = window_length_s * fs
window_shift = window_shift_s * fs

overlap = 6

def LoadTroikaDataset():
    """
    Retrieve the .mat filenames for the troika dataset.

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

    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 correspondin
            reference heart rates.
        confidence_est: a numpy array of confidence estimates for each pulse rate
            error.

    Returns:
        the MAE at 90% availability
    """
    # HIGH_Hzer 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))

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 ground_truth(ref_fl):
    """ loads the ground truth values
    args:
        ref_fl: Name of the .mat file that contains reference data
    Returns:
        A numpy array of the ground truth values for the input ref_fl
    """
    target = sp.io.loadmat(ref_fl)['BPM0']
    return target

def bandpass_filter(signal, LOW_Hz, HIGH_Hz,  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, (LOW_Hz, HIGH_Hz), btype='bandpass', fs=fs)
    return sp.signal.filtfilt(b, a, signal)

def Featurize(signal, fs):
    """
    A featurization of the accelerometer signal.
    Args:
        signal: (np.array) The input signal
        fs: (number) The sampling rate of <signal>
    Returns:
         n-tuple of accelerometer features
    """

    # fft of ppg signal
    fft_len = 2*len(signal)

    # Create an array of frequency bins
    freqs = np.fft.rfftfreq(fft_len, 1 / fs)

    # Take an FFT of the centered signal
    fft = np.abs(np.fft.rfft(signal - np.mean(signal), fft_len))

    # Filter fft between low_bpm and high_pbm after narwing the range
    fft[freqs <= (LOW_BPM + 20)/60] = 0.0
    fft[freqs >= (HIGH_BPM - 60)/60] = 0.0

    # signal mean  and energy
    mean = np.mean(signal)
    energy = np.sum(np.square(signal - mean))

    return freqs, fft, energy


def RunPulseRateAlgorithm(data_fl, ref_fl):
    # Load data using LoadTroikaDataFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)

    # load the ground truth
    target = ground_truth(ref_fl)

    # filter signals
    ppg  = bandpass_filter(ppg, LOW_Hz, HIGH_Hz, fs)
```

```
accx = bandpass_filter(accx, LOW_Hz, HIGH_Hz, fs)
accy = bandpass_filter(accy, LOW_Hz, HIGH_Hz, fs)
accz = bandpass_filter(accz, LOW_Hz, HIGH_Hz, fs)

# calculate acc from (x,y,z)
acc = np.sqrt(accx**2 + accy**2 + accz**2)

est, confidence = [],[]

for i in range(0, len(ppg) - window_length, window_shift):

    ppg_freqs, ppg_fft, ppg_energy = Featurize(ppg[i:i+window_length], fs)
    acc_freqs, acc_fft, acc_energy = Featurize(acc[i:i+window_length], fs)

    max_ppg = ppg_freqs[np.argmax(ppg_energy)]
    max_acc = acc_freqs[np.argmax(acc_energy)]

    j = 1
    while np.abs(max_ppg - max_acc) <= 0 and j <=2:
        j += 1
        max_ppg = ppg_freqs[np.argsort(ppg_fft, axis = 0)[-j]]
        max_acc = acc_freqs[np.argsort(acc_fft, axis = 0)[-j]]

    est_freqs = max_ppg
    est.append(est_freqs *60)

    # compute confidence
    conf = np.sum(ppg_fft[(ppg_freqs >= (est_freqs - LOW_Hz)) & (ppg_freqs <= (est_f
    confidence.append(conf)

# Return per-estimate MAE and confidence.
errors = np.abs(np.diag(est - target))
return errors, np.array(confidence)
```