unit_test

June 27, 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.

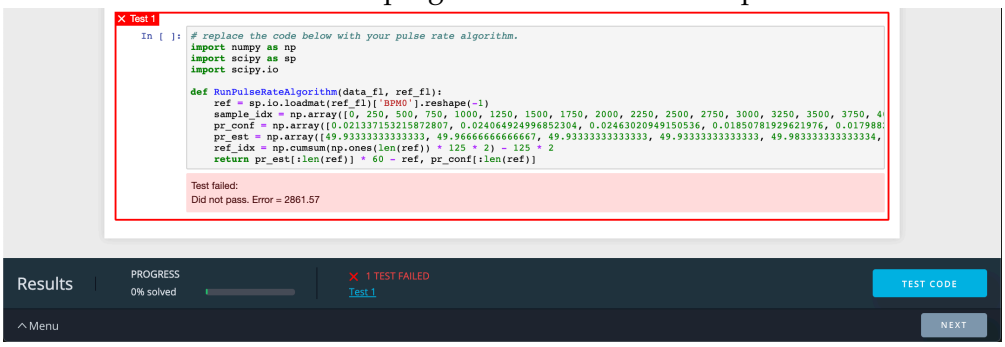
2. In the bottom right, click the Test Run button.

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.

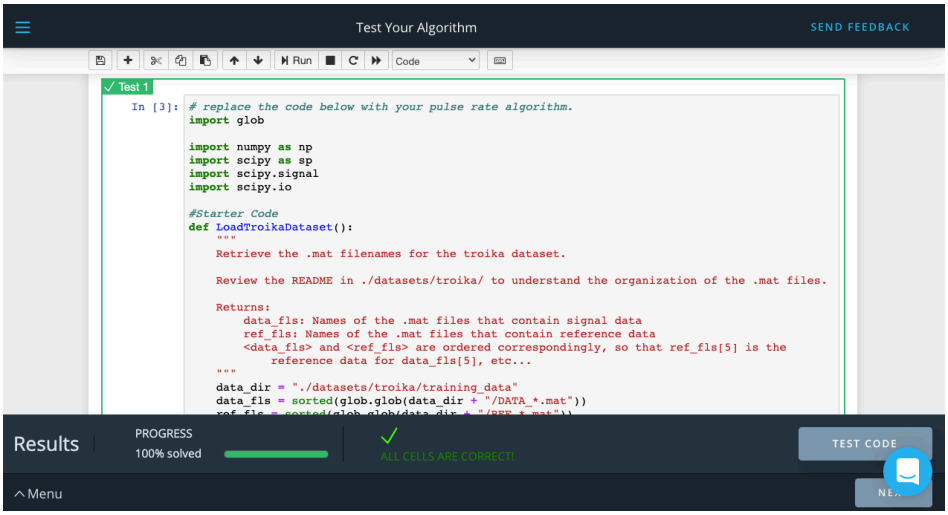
1.1.2 Pass

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.



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

```
In [3]: # replace the code below with your pulse rate algorithm.
        import glob

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

        #Starter Code
        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 LoadTroikaRefFile(ref_fl):
    ref = sp.io.loadmat(ref_fl)
    return ref

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
    """
    # 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))

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)

#Algorithm functions start here and are below
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 moving_pulse_confidence_estimate(ppg, acc, ref, fs):
    """a function to determine the pulse rate and confidence estimate in a given 8 secon
    2 seconds, window and increment values were arrived at from the Troika dataset Readm

    Args:
        ppg: (np.array) The bandpass filtered ppg signal
        acc: (np.array) The bandpass filtered summed acc signal
        fs: (number) The sampling rate of <signal>
        ref: (np.array) a numpy array of reference pulse rates from the Troika dataset

    Returns:
        (tuple of numpy arrays) :  a tuple of numpy arrays for the mean absolute error b
    """

    start_ind = 0
    # 8 second sampling window
```

```
    window = int(fs*8)
    end_ind = window
    # increment/output every 2 seconds
    increment = int(fs*2)
    pr_ests = []
    confs = []
    while end_ind < len(ppg):
        ppg_w = ppg[start_ind:end_ind]
        acc_w = acc[start_ind:end_ind]

        pr_estimate, conf = pulse_confidence_calc(ppg_w, ppg, acc_w, fs)
        pr_ests.append(pr_estimate)
        confs.append(conf)

        start_ind += increment
        end_ind += increment
    errors = calc_errors(pr_ests, ref)
    return np.asarray(errors), np.asarray(confs)

def pulse_confidence_calc(ppg, full_ppg, acc, fs):
    """a function to calculate both the estimated pulse rate and confidence through the
    1) The estimated pulse rate is calculated by taking the fast fourier transform(fft)
    in the given signal window. Then the frequency where the maximum fft magnitude for t

    2) The confidence is determined by summing the spectral energy at the frequency for

    Args:
        ppg: (np.array) The bandpass filtered ppg signal for the given window
        full_ppg: (np.array) The full bandpass filtered ppg signal for confidence estima
        acc: (np.array) The bandpass filtered summed acc signal for the given window
        fs: (number) The sampling rate of <signal>

    Returns:
        pr_est, conf: (tuple - floating point numbers) : the pulse rate estimate in the
    """

    est_pr = 0
    conf_est = 0
    seconds = 60

    #compute fft freqs and mags for full bandpass filtered ppg signal
    full_ppg_freqs = np.fft.rfftfreq(len(full_ppg), 1 / fs)
    full_ppg_fft_mags = np.abs(np.fft.rfft(full_ppg))
    spectral_energy_ppg_spect = np.square(np.abs(full_ppg_fft_mags))

    #compute fft freqs, mags and freq where greatest ppg mag occurs
    ppg_freqs = np.fft.rfftfreq(len(ppg), 1 / fs)
    ppg_fft_mags = np.abs(np.fft.rfft(ppg))
```

```
    ppg_window = (ppg_freqs >= 0) & (ppg_freqs <= np.max(ppg_freqs))
    ppg_max_freq = ppg_freqs[np.argmax(ppg_fft_mags[ppg_window])]

    #compute fft freqs, mags and freq where greatest acc mag occurs
    acc_freqs = np.fft.rfftfreq(len(acc), 1 / fs)
    acc_fft_mags = np.abs(np.fft.rfft(acc))
    acc_window = (acc_freqs >= 0) & (acc_freqs <= np.max(acc_freqs))
    acc_max_freq = acc_freqs[np.argmax(acc_fft_mags[acc_window])]

    #if frequency where max ppg mag occurs is greater than or equal to
    #the frequency where the max acc mag occurs this is the estimated pulse rate
    #so compute it and the confidence estimate.
    #In the case where the frequencies are equal assume that the magnitude for the ppg s
    #is much greater than the magnitude for the acc signal
    if ppg_max_freq >= acc_max_freq:
        est_pr = ppg_max_freq * seconds

    #if frequency where max ppg mag occurs is less than
    #the frequency where the max acc mag occurs pick another frequency where the max
    #ppg mag occurs, this will be the estimated pulse rate,
    #compute it and the confidence estimate
    elif acc_max_freq > ppg_max_freq:
        new_window = (ppg_freqs > acc_max_freq) | (ppg_freqs < acc_max_freq)
        ppg_max_freq = ppg_freqs[np.argmax(ppg_fft_mags[new_window])]
        est_pr = ppg_max_freq * seconds

    ppg_max_window = (ppg_freqs == ppg_max_freq)
    max_energy = ppg_fft_mags[ppg_max_window]
    spectral_energy_max_ppg = np.square(np.abs(max_energy))
    conf = np.sum(spectral_energy_max_ppg)/np.sum(spectral_energy_ppg_spect)

    return est_pr, conf

def calc_errors(estimated_pr, ref):
    """a function to calculate both mean absolute error between the estimated pulse rate
    The function first interpolates the value of estimated pulse using the ref labels ar

    Args:
        estimated_pr: (np.array): a numpy array of estimated pulse rates
        ref: (np.array) a numpy array of reference pulse rates from the Troika dataset

    Returns:
        maes:(list) : a list of element wise mean absolute errors between the estimated
    """
    ref_ts = ref[:]
    estimated_pr_ts = estimated_pr[:]
    est_pr_interp = np.interp(ref_ts, estimated_pr_ts, estimated_pr)
```

```
    maes = []
    for sig in zip(est_pr_interp ,ref):
        mae = np.mean(np.abs(sig[0]-sig[1]))
        maes.append(mae)
    return maes

def RunPulseRateAlgorithm(data_fl, ref_fl):
    # Load data and ref using LoadTroikaDataFile and LoadTroikaRefFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)
    ref = LoadTroikaRefFile(ref_fl)
    ref = ref['BPM0']

    # sum the accelerometer channels into one accelo channel to aggregate the motion art
    sum_accelos = np.add(accx,accy,accz)

    #sampling frequency of 125Hz
    fs = 125

    # The passband was determined by looking at plots of the fft for the ppg and acc sig
    # and from the paper provided in the readme.md
    pass_band = [0.4, 5]
    f_ppg_signal = BandpassFilter(ppg, pass_band, fs)
    f_accelos_signal = BandpassFilter(sum_accelos, pass_band, fs)

    # Compute pulse rate estimates and estimation confidence.
    errors, confidences = moving_pulse_confidence_estimate(f_ppg_signal, f_accelos_signa

    # Return per-estimate mean absolute error and confidence as a 2-tuple of numpy array
    return errors, confidences
```

In [ ]: