unit_test

August 10, 2023

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.

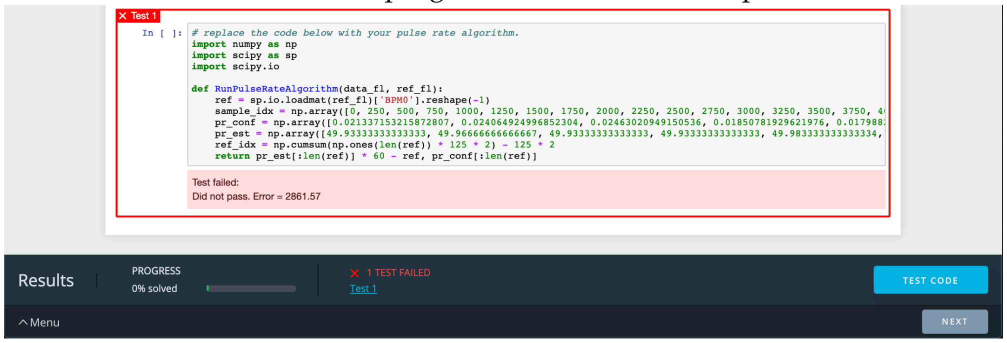
2. In the bottom right, click the Test Run button.

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



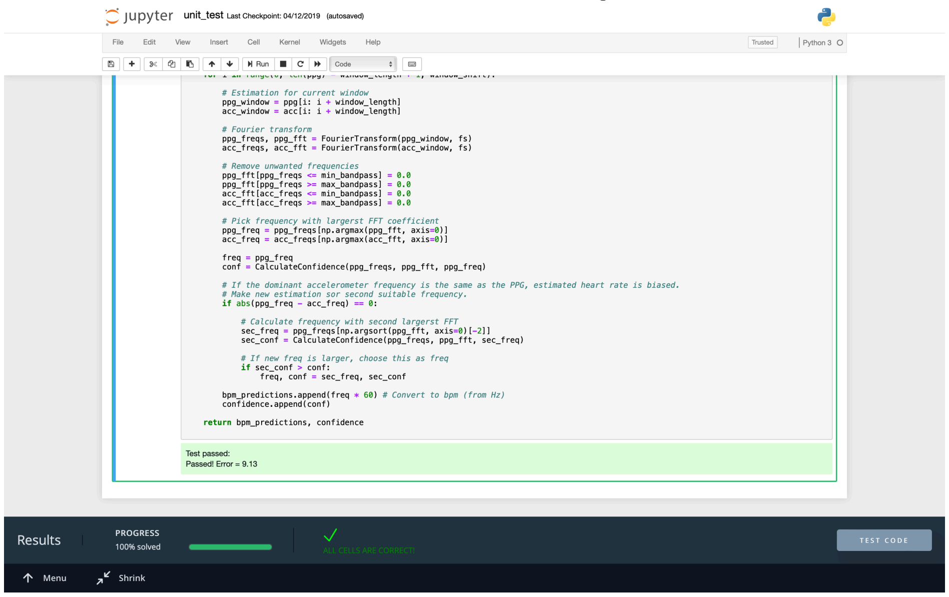
```
In [ ]: # replace the code below with your pulse rate algorithm.
        import numpy as np
        import scipy as sp
        import scipy.io

        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,
            pr_conf = np.array([0.021337153215872807, 0.024064924996852304, 0.02463020949150536, 0.01850781929621976, 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.



```
    for i in range(0, len(ppg) - window_length + 1, window_shift):

        # Estimation for current window
        ppg_window = ppg[i: i + window_length]
        acc_window = acc[i: i + window_length]

        # Fourier transform
        ppg_freqs, ppg_fft = FourierTransform(ppg_window, fs)
        acc_freqs, acc_fft = FourierTransform(acc_window, fs)

        # Remove unwanted frequencies
        ppg_fft[ppg_freqs <= min_bandpass] = 0.0
        ppg_fft[ppg_freqs >= max_bandpass] = 0.0
        acc_fft[acc_freqs <= min_bandpass] = 0.0
        acc_fft[acc_freqs >= max_bandpass] = 0.0

        # Pick frequency with largerst FFT coefficient
        ppg_freq = ppg_freqs[np.argmax(ppg_fft, axis=0)]
        acc_freq = acc_freqs[np.argmax(acc_fft, axis=0)]

        freq = ppg_freq
        conf = CalculateConfidence(ppg_freqs, ppg_fft, ppg_freq)

        # If the dominant accelerometer frequency is the same as the PPG, estimated heart rate is biased.
        # Make new estimation sor second suitable frequency.
        if abs(ppg_freq - acc_freq) == 0:

            # Calculate frequency with second largerst FFT
            sec_freq = ppg_freqs[np.argsort(ppg_fft, axis=0)[-2]]
            sec_conf = CalculateConfidence(ppg_freqs, ppg_fft, sec_freq)

            # If new freq is larger, choose this as freq
            if sec_conf > conf:
                freq, conf = sec_freq, sec_conf

        bpm_predictions.append(freq * 60) # Convert to bpm (from Hz)
        confidence.append(conf)

    return bpm_predictions, confidence
```

```
Test passed:
Passed! Error = 9.13
```

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

```
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,
        #     pr_conf = np.array([0.021337153215872807, 0.024064924996852304, 0.02463020949150
        #     pr_est = np.array([49.93333333333333, 49.96666666666667, 49.93333333333333, 49.93
        #     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 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 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
    """
    # 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)

def RunPulseRateAlgorithm(data_fl, ref_fl, fs=125):
    # Load data using LoadTroikaDataFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)

    # Compute pulse rate estimates and estimation confidence.
    #select bandpass filter within 40-240 Hz
    ppg, accx, accy, accz = BandFilter(ppg, fs), BandFilter(accx, fs), BandFilter(accy,

    # 3D accelerometer average
    acc = np.sqrt(accx**2 + accy**2 + accz**2)

    # Load ground truth as column vector and convert to row vector
    ground_truth = scipy.io.loadmat(ref_fl)['BPM0'].reshape(-1)

    # Calculate heart rate prediction, confidence and errors
    predictions, confidence = PredictHeartRate(acc, ppg, fs)
    errors = np.abs(np.subtract(predictions, ground_truth))

    # Return per-estimate mean absolute error and confidence
    errors, confidence = np.array(errors), np.array(confidence)
    return errors, confidence
```

```python
def BandFilter(signal, fs, bandpass=(40/60.0, 240/60.0)):
    """ Function to filter the frequency between 20 Hz and 240 Hz.
    Args:
        signal: A numpy array of signal
        fs (int): Sampling rate of the signal
        bandpass(tuple):bandpass within 40-240 Hz
    returns:
        predictions, confidences
    """
    b, a = scipy.signal.butter(3, bandpass, btype='bandpass', fs=fs) # 3: Order of the f
    return scipy.signal.filtfilt(b, a, signal)


def FourierTransform(signal, fs):
    """
    Calculate the Fast Fourier Transfrom of given sequence
    Args:
        signal: A numpy array for ppm or acc signal
        fs (int): Sampling frequency in Hz
    Returns:
        freqs: A numpy array Frequency bins
        fft (float): Magnitude of FFT
    """
    n_samples = len(signal)  # no zero padding
    freqs = np.fft.rfftfreq(n_samples, 1/fs)
    fft = np.abs(np.fft.rfft(signal, n_samples))
    return freqs, fft


def CalculateConfidence(freqs, fft, freq, window_width =1):
    """ Function to find the confidence level from Fourier transform
    Args:
        freqs: A numpy array of frequency
        ffts: A numpy array of FFT magnitude
        freq (float): frequency to calculate the confidence
    returns:
        confidence
    """
    window = (freqs > freq - window_width) & (freqs < freq + window_width)
    confidence = np.sum(fft[window]) / np.sum(fft)

    return confidence

def PredictHeartRate(acc, ppg, fs):
    """
    Function to predict heart rate and calculate confidence using ppg and accelerometer

    Args:
        acc: A numpy array of acceleration signal file
        ppg: A numpy array of PPG signal file
        fs (int): Sampling frequency in Hz
    returns:
        predictions: A list of predicited heart rates
        confidence: A list of estimated confidences
    """
    # Define window shapes with given values to function
    window_length=8
    window_shift=2
    min_bandpass=40/60.0
    max_bandpass=240/60.0
    window_length = window_length * fs
    window_shift = window_shift * fs

    # Create empty lists to store calculations
    bpm_predictions = []
    confidence = []

    # Make calculations with shifting windows
    for i in range(0, len(ppg) - window_length + 1, window_shift):

        # Estimation for current window
        ppg_window = ppg[i: i + window_length]
        acc_window = acc[i: i + window_length]

        # Fourier transform
        ppg_freqs, ppg_fft = FourierTransform(ppg_window, fs)
        acc_freqs, acc_fft = FourierTransform(acc_window, fs)

        # Remove unwanted frequencies
        ppg_fft[ppg_freqs <= min_bandpass] = 0.0
        ppg_fft[ppg_freqs >= max_bandpass] = 0.0
        acc_fft[acc_freqs <= min_bandpass] = 0.0
        acc_fft[acc_freqs >= max_bandpass] = 0.0

        # Pick frequency with largerst FFT coefficient
        ppg_freq = ppg_freqs[np.argmax(ppg_fft, axis=0)]
        acc_freq = acc_freqs[np.argmax(acc_fft, axis=0)]

        freq = ppg_freq
        conf = CalculateConfidence(ppg_freqs, ppg_fft, ppg_freq)

        # If the dominant accelerometer frequency is the same as the PPG, estimated hear
        # Make new estimation sor second suitable frequency.
        if abs(ppg_freq - acc_freq) == 0:

            # Calculate frequency with second largerst FFT
            sec_freq = ppg_freqs[np.argsort(ppg_fft, axis=0)[-2]]
            sec_conf = CalculateConfidence(ppg_freqs, ppg_fft, sec_freq)

            # If new freq is larger, choose this as freq
            if sec_conf > conf:
                freq, conf = sec_freq, sec_conf

        bpm_predictions.append(freq * 60) # Convert to bpm (from Hz)
        confidence.append(conf)

    return bpm_predictions, confidence
```