unit_test

October 14, 2022

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.

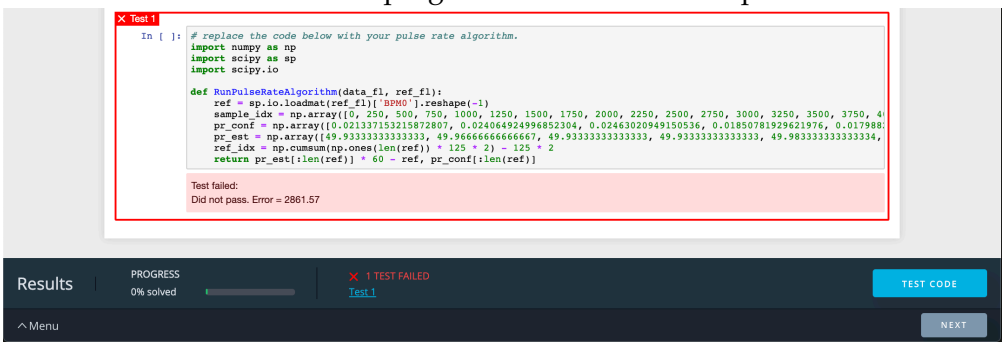
2. In the bottom right, click the Test Run button.

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.

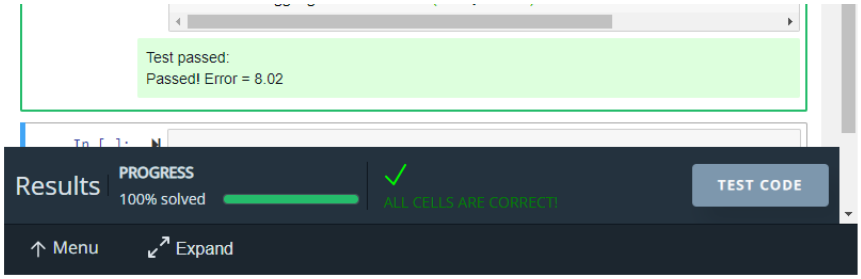
1.1.2 Pass

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 [1]: import glob
        import numpy as np
        import scipy as sp
        import scipy.signal
        import scipy.io
        from matplotlib import pyplot as plt

        plt.rcParams['figure.figsize'] = (12, 8)

        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 bandpass_filter(signal, fs):
    '''
    Bandpass filter to only allow frequencies  between 40BPM (0.66Hz) and 240BPM (40Hz).
    Args:
        sig: a numpy array of periodic signals
        fs: sample rate at which periodic signals were captured
    Returns:
        filtered signal based on the bandpass frequencies
    '''
    pass_band=(40/60.0, 240/60.0)
    b, a = scipy.signal.butter(3, pass_band, btype='bandpass', fs=fs)

    return scipy.signal.filtfilt(b, a, signal)


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)

    #print("percentile90_confidence",percentile90_confidence )
    # 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 calculate_confidence(freqs, fft, freq):
    """
    Calculate the confidence using the energy in the frequency spectrum.
    Args:
        freqs: frequency spectrum generated  the Fourier Transform from the signal.
        fft: The magnitude coefficient generated from the FFT of the signal.
        freq: The fundametal frequency of the signal for confidence calculation.

    Returns:
        confidence: float.
    """
    #Based on the EDA done, using half-window width of 1 to calculate the enegery at fre
    window_width = 2
    window = (freqs > freq - window_width/2) & (freqs < freq + window_width/2)
    confidence = np.sum(fft[window]) / np.sum(fft)

    return confidence

def fourier_transform(signal, fs):
    """
   Calculate the Fast Fourier Transform for the signal sequence.
   Args:
       signal: The signal such from PPG or accelerometer.
       fs: The signal sampling rate.

    returns:
        freqs: The frequency bins based on the signal and the sampling rate
        fft: The magnitude coefficients corresonding to the freqs.
    """

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


    return freqs, fft


def estimate(ppg, acc, window_length, window_shift, fs):
    """
    Estimate the heart rate (bmp) and confidence

    Args:
        PPG: ppg signal -band passed
        acc: The magnitude of the three-axis accelerometer signals - band passed
        window_length: The time window that will be processed
        window_shift: The time shift for sliding the window_length
```

```
    Returns:
       bpm_estimates:The estimated heart rates
       confidence: The corresponding  for the estimated heart-rate
    """

    window_length = window_length * fs
    window_shift = window_shift * fs


    bpm_estimates = []
    confidence = []

    for i in range(0, len(ppg) - window_length +1 , window_shift):
        ppg_window = ppg[i:i + window_length]
        acc_window = acc[i:i + window_length]

        # Fourier Transform for the current window
        ppg_freqs, ppg_fft = fourier_transform(ppg_window, fs)
        acc_freqs, acc_fft = fourier_transform(acc_window, fs)


        # Select frequency with largest 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 = calculate_confidence(ppg_freqs, ppg_fft, ppg_freq)
        # Determine if the energy at the next frequency would be higher (e.g. higher con
        # ppg frequency is same as acc frequency.
        if  np.abs(ppg_freq - acc_freq) == 0:
            second_freq = ppg_freqs[np.argsort(ppg_fft, axis=0)[-2]]
            second_conf = calculate_confidence(ppg_freqs, ppg_fft, second_freq)
            #If the confidence of the second frequency is higher than the first, take th
            if second_conf > conf:
                freq, conf = second_freq, second_conf

        bpm_estimates.append(freq * 60) # Converting to BMP
        confidence.append(conf)

    return bpm_estimates, confidence


def RunPulseRateAlgorithm(data_fl, ref_fl, fs=125):


    # Windowing our ppg and acc signal to estimate
    window_length = 8
    window_shift = 2
```

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

        # Compute pulse rate estimates and estimation confidence.
        #Bandpass filtering
        ppg_filtered = bandpass_filter(ppg,fs)
        accx_filtered = bandpass_filter(accx,fs)
        accy_filtered = bandpass_filter(accy,fs)
        accz_filtered = bandpass_filter(accz,fs)

        acc_mag_filtered = bandpass_filter(np.sqrt(accx_filtered**2 + accy_filtered**2 + acc

        #Ground truth for MAE calculation
        ground_truth = scipy.io.loadmat(ref_fl)['BPM0'].reshape(-1)

        #Get BMP estimates, confidence and error metrics
        predictions, confidence = estimate(ppg_filtered, acc_mag_filtered, window_length, wi
        errors = np.abs(np.subtract(predictions, ground_truth))

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


    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, fs=125)
            errs.append(errors)
            confs.append(confidence)
            # Compute aggregate error metric
        errs = np.hstack(errs)
        confs = np.hstack(confs)

        return AggregateErrorMetric(errs, confs)
```

In [ ]: