unit_test

February 20, 2021

1 Test Your Algorithm

1.1 Instructions

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

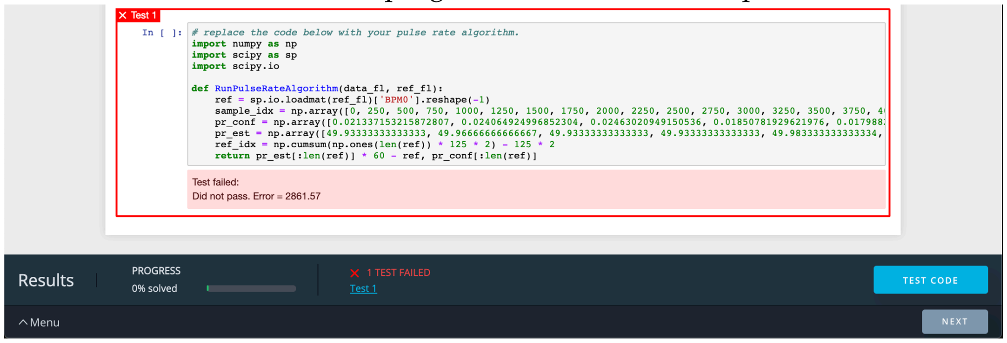
- Copy over all the **Code** section to the following Code block.
- Download as a Python (`.py`) and copy the code to the following Code block.

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**. 

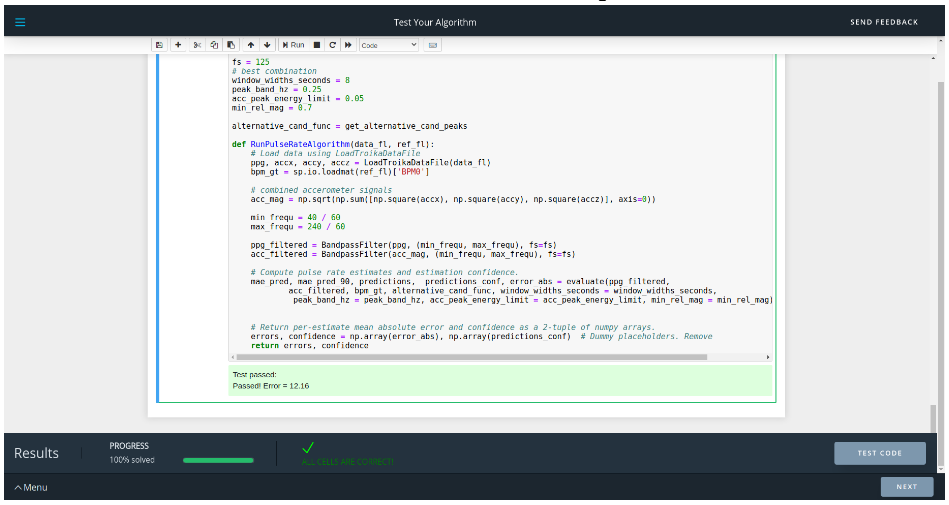
```
In [ ]: # replace the code below with your pulse rate algorithm.
        import numpy as np
        import scipy as sp
        import scipy.io

        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, 0.018507819296219766, 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
```

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.



```
fs = 125
# best combination
window_widths_seconds = 8
peak_band_hz = 0.25
acc_peak_energy_limit = 0.05
min_rel_mag = 0.7

alternative_cand_func = get_alternative_cand_peaks

def RunPulseRateAlgorithm(data_fl, ref_fl):
    # Load data using LoadTroikaDataFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)
    bpm_gt = sp.io.loadmat(ref_fl)['BPM0']

    # combined accerometer signals
    acc_mag = np.sqrt(np.sum([np.square(accx), np.square(accy), np.square(accz)], axis=0))

    min_frequ = 40 / 60
    max_frequ = 240 / 60

    ppg_filtered = BandpassFilter(ppg, (min_frequ, max_frequ), fs=fs)
    acc_filtered = BandpassFilter(acc_mag, (min_frequ, max_frequ), fs=fs)

    # Compute pulse rate estimates and estimation confidence.
    mae_pred, mae_pred_90, predictions,  predictions_conf, error_abs = evaluate(ppg_filtered,
            acc_filtered, bpm_gt, alternative_cand_func, window_widths_seconds = window_widths_seconds,
             peak_band_hz = peak_band_hz, acc_peak_energy_limit = acc_peak_energy_limit, min_rel_mag = min_rel_mag)


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

```
Test passed:
Passed! Error = 12.16
```

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


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

```python
# for confidence calculation
def peak_energy(fft_mag_window, freq_window, peak_hz, peak_band_hz):
    peak_band = (freq_window > (peak_hz - peak_band_hz)) &  (freq_window < (peak_hz + pe
    energy_frac = np.sum(fft_mag_window[peak_band]) / np.sum(fft_mag_window)
    return energy_frac

def get_peak_candidates(ppg_fft_abs_window, min_fraction, min_distance_hz):
    # get peaks
    pks_index = sp.signal.find_peaks(ppg_fft_abs_window)[0]
    pks_mag = ppg_fft_abs_window[pks_index]

    sorted_peaks = sorted(zip(pks_index,pks_mag), key = lambda x: x[1], reverse=True)

    # filter for min fraction
    max_peak_mag = sorted_peaks[0][1]
    max_peak_index = sorted_peaks[0][0]
    sorted_peaks_filtered = list(filter(lambda x: x[1]/max_peak_mag > min_fraction, sort
    #print(sorted_peaks_filtered)

    # filter for min distance
    min_distance_index = min_distance_hz / 0.125
    sorted_peaks_filtered_dist = list(filter(lambda x: np.abs(x[0]-max_peak_index) > min
    #print(sorted_peaks_filtered_dist)

    return sorted_peaks_filtered_dist

def get_alternative_cand_peaks(ppg_fft_mag_window, acc_fft_mag_window, freq_window, min_
    """
    Gets alternative candidate based on other suitable peaks in the ppg signal
    """
    pk_cands = get_peak_candidates(ppg_fft_mag_window, min_rel_mag, peak_band_hz)
    if pk_cands:
        pk_alt_index = pk_cands[0][0]
        alt_cand_frequ = freq_window[pk_alt_index]
    else:
        alt_cand_frequ = None
    return alt_cand_frequ

def get_bpm(ppg_window, acc_window, get_alternative_cand,
            peak_band_hz = 0.25, acc_peak_energy_limit = 0.3, min_rel_mag = 0.8):

    freq_window = np.fft.rfftfreq(len(ppg_window), 1/fs)

    # fft ppg
    ppg_fft_window = np.fft.rfft(ppg_window)
    ppg_fft_mag_window = np.abs(ppg_fft_window)

    # get ppg max frequency + bpm
    ppg_mag_max_window = np.max(ppg_fft_mag_window)
    ppg_mag_max_frequ_window = freq_window[np.argmax(ppg_fft_mag_window)]
    ppg_mag_max_bpm_window = ppg_mag_max_frequ_window * 60
    ppg_mag_at_max = ppg_fft_mag_window[np.argmax(ppg_fft_mag_window)]

    # fft acc
    acc_fft_window = np.fft.rfft(acc_window)
    acc_fft_mag_window = np.abs(acc_fft_window)

    # get ppg max frequency + bpm
    acc_mag_max_window = np.max(acc_fft_mag_window)
    acc_mag_max_frequ_window = freq_window[np.argmax(acc_fft_mag_window)]
    acc_mag_max_bpm_window = acc_mag_max_frequ_window * 60

    # get ppg alternative candidates
    alt_cand_frequ = get_alternative_cand(ppg_fft_mag_window, acc_fft_mag_window, freq_w

    # is primary ppg peak in range of acc peak?
    close_to_acc_peak = np.abs(ppg_mag_max_frequ_window - acc_mag_max_frequ_window) < (p

    # what is the energy of this acc peak?
    acc_peak_band_energy = peak_energy(acc_fft_mag_window, freq_window, ppg_mag_max_freq
    high_acc_peak_band_energy = acc_peak_band_energy > acc_peak_energy_limit

    # is the alternative peak high enough?
    alt_peak_high_enough = False
    if alt_cand_frequ:
        ppg_mag_at_alt_cand = ppg_fft_mag_window[np.argwhere(freq_window == alt_cand_fre
        alt_peak_high_enough = (ppg_mag_at_alt_cand / ppg_mag_at_max) > min_rel_mag

    # should we take other ppg frequency?
    if close_to_acc_peak and high_acc_peak_band_energy and alt_peak_high_enough:
        pred_frequ = alt_cand_frequ
    else:
        pred_frequ = ppg_mag_max_frequ_window
    pred_bpm = pred_frequ * 60

    # get confidence
    pred_confidence = peak_energy(ppg_fft_mag_window, freq_window, pred_frequ, peak_band

    return pred_bpm, pred_confidence, alt_cand_frequ

def evaluate(ppg_filtered, acc_filtered, bpm_gt, alternative_cand_func, window_widths_se
             peak_band_hz = 0.3, acc_peak_energy_limit = 0.2, min_rel_mag = 0.6):
    window_width_data_points = window_widths_seconds * fs
    window_overlap = window_overlap_seconds * fs

    predictions = []
    predictions_conf = []
    for window_index in range(len(bpm_gt)):
        start_index = window_overlap * window_index
        end_index = window_overlap * window_index + window_width_data_points

        ppg_window = ppg_filtered[start_index:end_index]
        acc_window = acc_filtered[start_index:end_index]
        pred_bpm, pred_confidence, alt_cand_frequ = get_bpm(ppg_window, acc_window, alte
                                                        peak_band_hz, acc_peak_energ

        predictions.append(pred_bpm)
        predictions_conf.append(pred_confidence)

    error_abs = np.abs(predictions - bpm_gt[:,0])
    mae_pred = np.mean(np.abs(predictions - bpm_gt[:,0]))

    combined = zip(predictions,predictions_conf,bpm_gt[:,0])
    combined_sorted = sorted(combined, key = lambda x: x[1], reverse = True)
    combined_90 = combined_sorted[:int(np.round(len(combined_sorted) * 0.9))]
    mae_pred_90 = np.mean([np.abs(pred-gt) for pred, conf, gt in combined_90])
    #sorted_errors = np.sort(np.abs(predictions - bpm_gt[:,0]))
    #mae_pred_90 = np.mean(sorted_errors[:int(len(sorted_errors) * 0.9)])

    return mae_pred, mae_pred_90, predictions,  predictions_conf, error_abs

fs = 125
# best combination
window_widths_seconds = 8
peak_band_hz = 0.25
acc_peak_energy_limit = 0.05
min_rel_mag = 0.7

alternative_cand_func = get_alternative_cand_peaks

def RunPulseRateAlgorithm(data_fl, ref_fl):
    # Load data using LoadTroikaDataFile
    ppg, accx, accy, accz = LoadTroikaDataFile(data_fl)
    bpm_gt = sp.io.loadmat(ref_fl)['BPM0']

    # combined accerometer signals
    acc_mag = np.sqrt(np.sum([np.square(accx), np.square(accy), np.square(accz)], axis=0

    min_frequ = 40 / 60
    max_frequ = 240 / 60

    ppg_filtered = BandpassFilter(ppg, (min_frequ, max_frequ), fs=fs)
    acc_filtered = BandpassFilter(acc_mag, (min_frequ, max_frequ), fs=fs)
```

```
# Compute pulse rate estimates and estimation confidence.
mae_pred, mae_pred_90, predictions,  predictions_conf, error_abs = evaluate(ppg_filt
        acc_filtered, bpm_gt, alternative_cand_func, window_widths_seconds = window_
         peak_band_hz = peak_band_hz, acc_peak_energy_limit = acc_peak_energy_limit,


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