unit_test

November 2, 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.

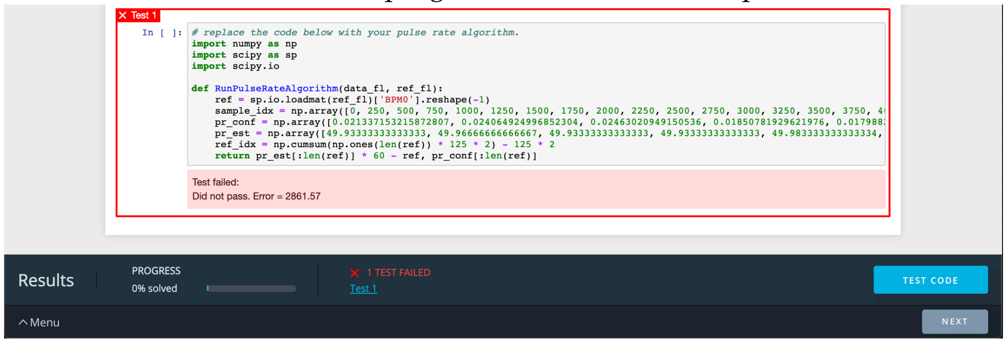
2. In the bottom right, click the Test Run button.

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**.



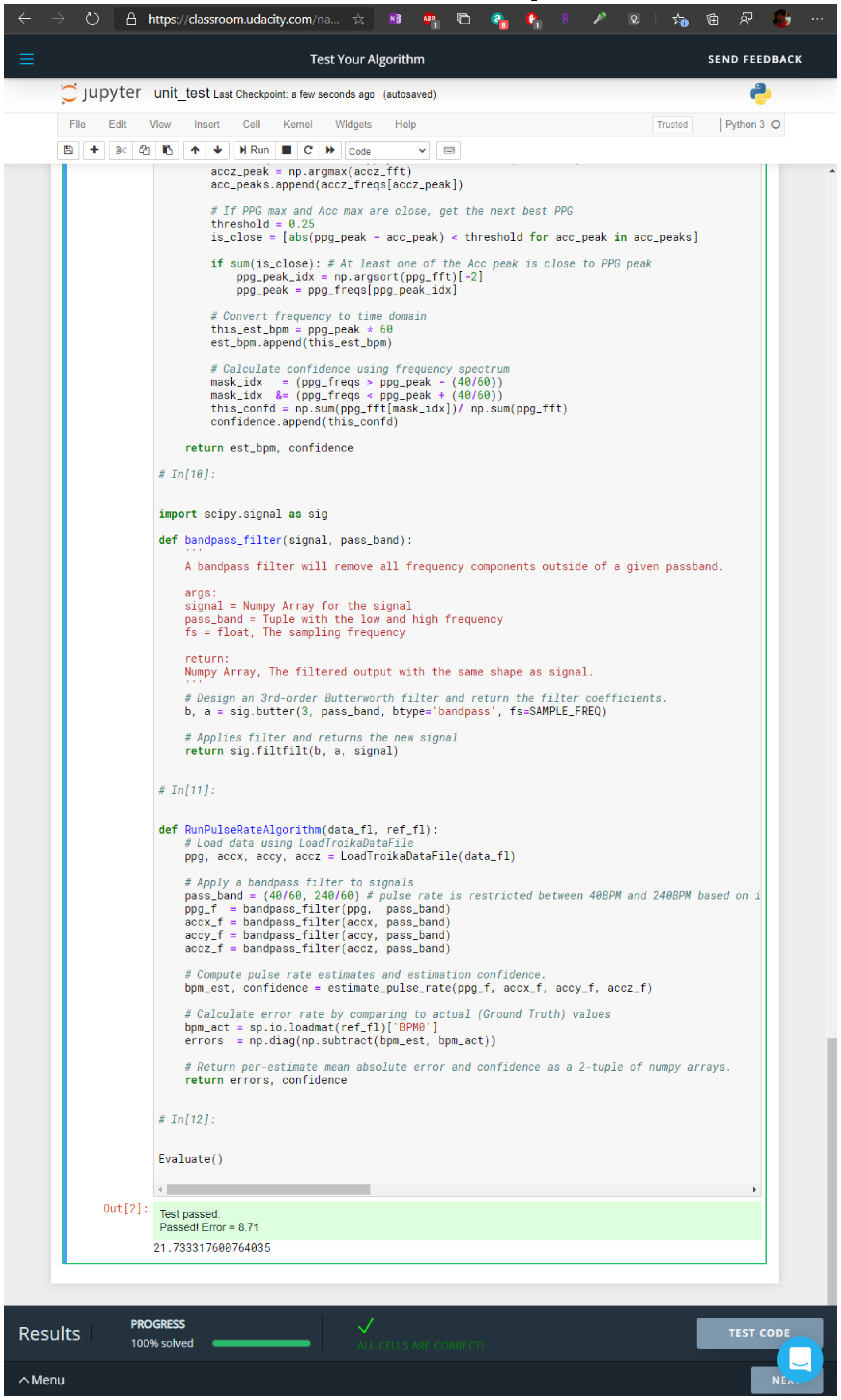
```
In [ ]: # replace the code below with your pulse rate algorithm.
        import numpy as np
        import scipy as sp
        import scipy.io

        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, 0.01850781929621976, 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
```

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.


https://classroom.udacity.com/na...
Test Your Algorithm
SEND FEEDBACK
jupyter unit_test Last Checkpoint: a few seconds ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
            accz_peak = np.argmax(accz_fft)
            acc_peaks.append(accz_freqs[accz_peak])

            # If PPG max and Acc max are close, get the next best PPG
            threshold = 0.25
            is_close = [abs(ppg_peak - acc_peak) < threshold for acc_peak in acc_peaks]

            if sum(is_close): # At least one of the Acc peak is close to PPG peak
                ppg_peak_idx = np.argsort(ppg_fft)[-2]
                ppg_peak = ppg_freqs[ppg_peak_idx]

            # Convert frequency to time domain
            this_est_bpm = ppg_peak * 60
            est_bpm.append(this_est_bpm)

            # Calculate confidence using frequency spectrum
            mask_idx   = (ppg_freqs > ppg_peak - (40/60))
            mask_idx  &= (ppg_freqs < ppg_peak + (40/60))
            this_confd = np.sum(ppg_fft[mask_idx])/ np.sum(ppg_fft)
            confidence.append(this_confd)

        return est_bpm, confidence

# In[10]:


import scipy.signal as sig

def bandpass_filter(signal, pass_band):
    '''
    A bandpass filter will remove all frequency components outside of a given passband.

    args:
    signal = Numpy Array for the signal
    pass_band = Tuple with the low and high frequency
    fs = float, The sampling frequency

    return:
    Numpy Array, The filtered output with the same shape as signal.
    '''
    # Design an 3rd-order Butterworth filter and return the filter coefficients.
    b, a = sig.butter(3, pass_band, btype='bandpass', fs=SAMPLE_FREQ)

    # Applies filter and returns the new signal
    return sig.filtfilt(b, a, signal)


# In[11]:


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

    # Apply a bandpass filter to signals
    pass_band = (40/60, 240/60) # pulse rate is restricted between 40BPM and 240BPM based on i
    ppg_f  = bandpass_filter(ppg,  pass_band)
    accx_f = bandpass_filter(accx, pass_band)
    accy_f = bandpass_filter(accy, pass_band)
    accz_f = bandpass_filter(accz, pass_band)

    # Compute pulse rate estimates and estimation confidence.
    bpm_est, confidence = estimate_pulse_rate(ppg_f, accx_f, accy_f, accz_f)

    # Calculate error rate by comparing to actual (Ground Truth) values
    bpm_act = sp.io.loadmat(ref_fl)['BPM0']
    errors  = np.diag(np.subtract(bpm_est, bpm_act))

    # Return per-estimate mean absolute error and confidence as a 2-tuple of numpy arrays.
    return errors, confidence


# In[12]:


Evaluate()
```

```
Out[2]: Test passed:
        Passed! Error = 8.71
        21.733317600764035
```

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

```
In [2]: #!/usr/bin/env python
        # coding: utf-8

        # ## Part 1: Pulse Rate Algorithm
        #
        # ### Contents
        # Fill out this notebook as part of your final project submission.
        #
        # **You will have to complete both the Code and Project Write-up sections.**
        # - The [Code](#Code) is where you will write a **pulse rate algorithm** and already inc
        #    - Imports - These are the imports needed for Part 1 of the final project.
        #      - [glob](https://docs.python.org/3/library/glob.html)
        #      - [numpy](https://numpy.org/)
        #      - [scipy](https://www.scipy.org/)
        # - The [Project Write-up](#Project-Write-up) to describe why you wrote the algorithm fo
        #
        #
        # ### Dataset
        # You will be using the **Troika**[1] dataset to build your algorithm. Find the dataset
        #
        # 1. Zhilin Zhang, Zhouyue Pi, Benyuan Liu, TROIKA: A General Framework for Heart Rate M
        #
        # -----

        # ### Code

        # In[1]:


        import glob

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


        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)

SAMPLE_FREQ = 125 # From Readme.pdf: All signals were sampled at 125 Hz


# In[8]:


def apply_fourier_transform(signal):
    '''
    Compute the one-dimensional discrete Fourier Transform for real input.

    args:
    signal = Numpy Array, signal
    fs = Float, Sample Frequency
    '''
    freqs = np.fft.rfftfreq(len(signal), 1/SAMPLE_FREQ)
    fft = np.abs(np.fft.rfft(signal)) # abs() is used as we are interested in Magnitude
    return freqs, fft


# **Note (from Readme.pdf)** \
# Variable 'BPM0', gives the BPM value in every 8-second time window. Note that two succ
# overlap by 6 seconds. Thus the first value in 'BPM0' gives the calcualted heart rate g
# the first 8 seconds, while the second value in 'BPM0' gives the calculated heart rate
# from the 3rd second to the 10th second.

# In[9]:
```

```
def estimate_pulse_rate(ppg, accx, accy, accz):
    '''
    Computes the pulse rate based on PPG values and the Accelerometer data

    args:
    ppg  = Numpy Array, Optically obtained plethysmogram signals
    accx = Numpy Array, X-axis signals from a 3-axis accelerometer
    accy = Numpy Array, Y-axis signals from a 3-axis accelerometer
    accz = Numpy Array, Z-axis signals from a 3-axis accelerometer

    returns:
    bpm_est = List, Estimated pulse rate
    confidence = List, Confidence value
    '''
    interval = 8     # From Readme.pdf: Variable 'BPM0', gives the BPM value in every 8-
    overlap  = 8 - 6 # From Readme.pdf: Two successive time windows overlap by 6 seconds

    # Convert seconds to frequency
    interval *= SAMPLE_FREQ
    overlap  *= SAMPLE_FREQ

    max_at = len(ppg) - interval

    est_bpm = []
    confidence = []

    for sig_start in range(0, max_at, overlap):
        sig_end = sig_start + interval

        this_est_bpm = 0
        acc_peaks = []

        # Find the maximum PPG signal frequency in the current signal
        this_ppg = ppg[sig_start:sig_end]
        ppg_freqs, ppg_fft = apply_fourier_transform(this_ppg)
        ppg_peak_idx = np.argmax(ppg_fft)
        ppg_peak = ppg_freqs[ppg_peak_idx]

        this_accx = accx[sig_start:sig_end]
        accx_freqs, accx_fft = apply_fourier_transform(this_accx)
        accx_peak = np.argmax(accx_fft)
        acc_peaks.append(accx_freqs[accx_peak])

        this_accy = accy[sig_start:sig_end]
        accy_freqs, accy_fft = apply_fourier_transform(this_accy)
        accy_peak = np.argmax(accy_fft)
        acc_peaks.append(accy_freqs[accy_peak])
        this_accz = accz[sig_start:sig_end]
        accz_freqs, accz_fft = apply_fourier_transform(this_accz)
        accz_peak = np.argmax(accz_fft)
        acc_peaks.append(accz_freqs[accz_peak])

        # If PPG max and Acc max are close, get the next best PPG
        threshold = 0.25
        is_close = [abs(ppg_peak - acc_peak) < threshold for acc_peak in acc_peaks]

        if sum(is_close): # At least one of the Acc peak is close to PPG peak
            ppg_peak_idx = np.argsort(ppg_fft)[-2]
            ppg_peak = ppg_freqs[ppg_peak_idx]

        # Convert frequency to time domain
        this_est_bpm = ppg_peak * 60
        est_bpm.append(this_est_bpm)

        # Calculate confidence using frequency spectrum
        mask_idx   = (ppg_freqs > ppg_peak - (40/60))
        mask_idx  &= (ppg_freqs < ppg_peak + (40/60))
        this_confd = np.sum(ppg_fft[mask_idx])/ np.sum(ppg_fft)
        confidence.append(this_confd)

    return est_bpm, confidence

# In[10]:


import scipy.signal as sig

def bandpass_filter(signal, pass_band):
    '''
    A bandpass filter will remove all frequency components outside of a given passband.

    args:
    signal = Numpy Array for the signal
    pass_band = Tuple with the low and high frequency
    fs = float, The sampling frequency

    return:
    Numpy Array, The filtered output with the same shape as signal.
    '''
    # Design an 3rd-order Butterworth filter and return the filter coefficients.
    b, a = sig.butter(3, pass_band, btype='bandpass', fs=SAMPLE_FREQ)

    # Applies filter and returns the new signal
    return sig.filtfilt(b, a, signal)
```

```
# In[11]:


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

    # Apply a bandpass filter to signals
    pass_band = (40/60, 240/60) # pulse rate is restricted between 40BPM and 240BPM base
    ppg_f  = bandpass_filter(ppg,  pass_band)
    accx_f = bandpass_filter(accx, pass_band)
    accy_f = bandpass_filter(accy, pass_band)
    accz_f = bandpass_filter(accz, pass_band)

    # Compute pulse rate estimates and estimation confidence.
    bpm_est, confidence = estimate_pulse_rate(ppg_f, accx_f, accy_f, accz_f)

    # Calculate error rate by comparing to actual (Ground Truth) values
    bpm_act = sp.io.loadmat(ref_fl)['BPM0']
    errors  = np.diag(np.subtract(bpm_est, bpm_act))

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


# In[12]:


Evaluate()
```

```
Out[2]: 21.733317600764035
```