

unit_test

April 17, 2020

1 Test Your Algorithm

1.1 Instructions

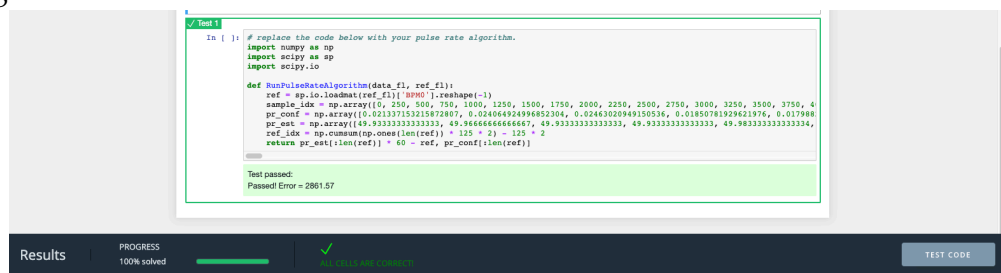
1. From the **Pulse Rate Algorithm** Notebook you can do one of the following:
 - Copy over all the **Code** section to the following Code block.
 - Download as a Python (.py) and copy the code to the following Code block.
2. In the bottom right, click the Test Run button.

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

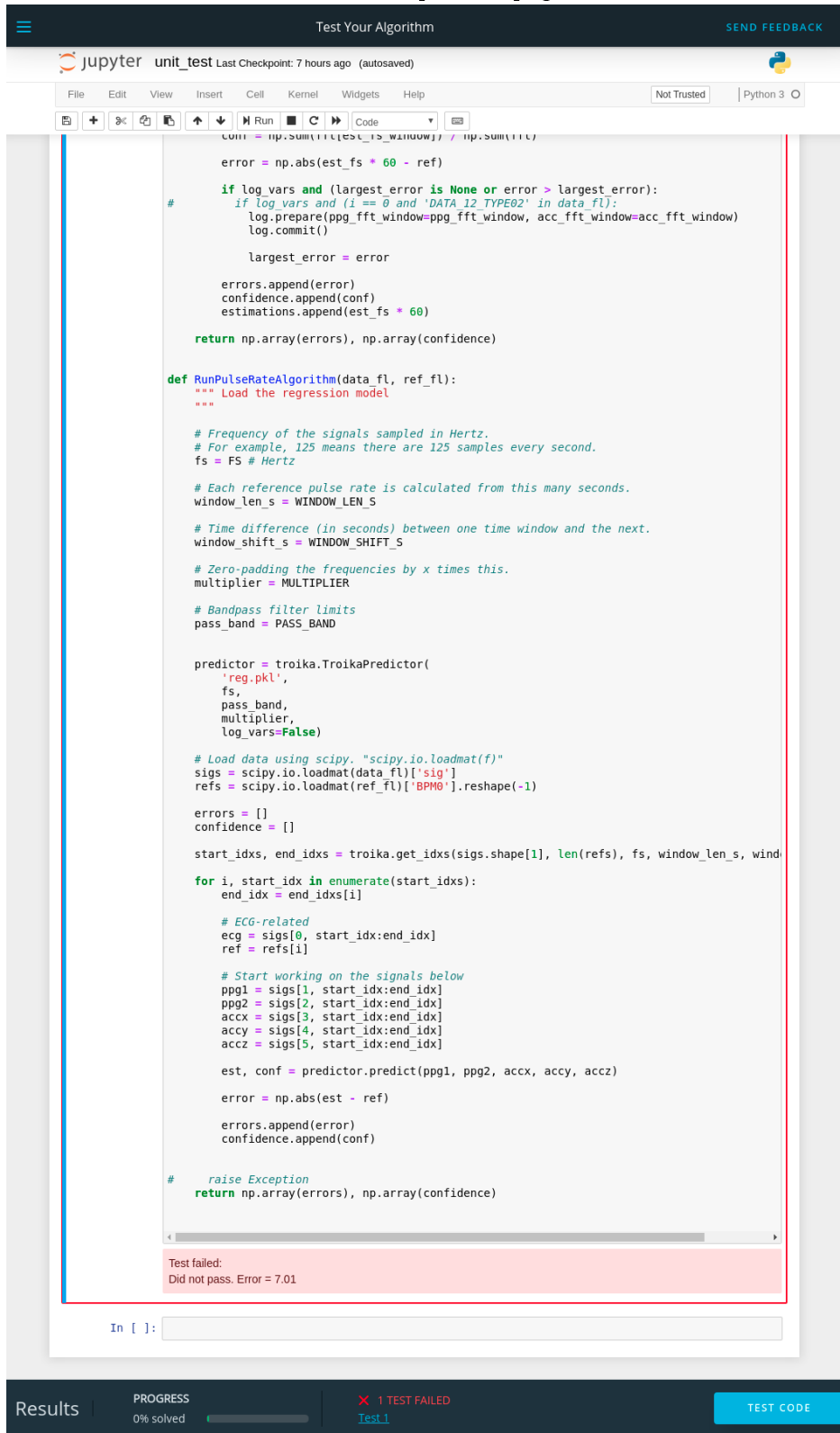
If your code passes the test, complete the following! You **must** include a screenshot of your code and the Test being **Passed**. Here is what the starter filler code looks like when the test is run and



should be similar.

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.



```
Test Your Algorithm SEND FEEDBACK

jupyter unit_test Last Checkpoint: 7 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

comm = np.sum((1/(est_fs_window)) / np.sum(1/c))

error = np.abs(est_fs * 60 - ref)

if log_vars and (largest_error is None or error > largest_error):
    if log_vars and (1 == 0 and 'DATA_12_TYPE02' in data_fl):
        log.prepare(ppg_fft_window=ppg_fft_window, acc_fft_window=acc_fft_window)
        log.commit()

    largest_error = error

errors.append(error)
confidence.append(conf)
estimations.append(est_fs * 60)

return np.array(errors), np.array(confidence)

def RunPulseRateAlgorithm(data_fl, ref_fl):
    """ Load the regression model
    """

    # Frequency of the signals sampled in Hertz.
    # For example, 125 means there are 125 samples every second.
    fs = FS # Hertz

    # Each reference pulse rate is calculated from this many seconds.
    window_len_s = WINDOW_LEN_S

    # Time difference (in seconds) between one time window and the next.
    window_shift_s = WINDOW_SHIFT_S

    # Zero-padding the frequencies by x times this.
    multiplier = MULTIPLIER

    # Bandpass filter limits
    pass_band = PASS_BAND

    predictor = troika.TroikaPredictor(
        'reg.pkl',
        fs,
        pass_band,
        multiplier,
        log_vars=False)

    # Load data using scipy. "scipy.io.loadmat(f)"
    sigs = scipy.io.loadmat(data_fl)['sig']
    refs = scipy.io.loadmat(ref_fl)['BPM0'].reshape(-1)

    errors = []
    confidence = []

    start_idx, end_idx = troika.get_idx(sigs.shape[1], len(refs), fs, window_len_s, wind

    for i, start_idx in enumerate(start_idx):
        end_idx = end_idx[i]

        # ECG-related
        ecg = sigs[0, start_idx:end_idx]
        ref = refs[i]

        # Start working on the signals below
        ppg1 = sigs[1, start_idx:end_idx]
        ppg2 = sigs[2, start_idx:end_idx]
        accx = sigs[3, start_idx:end_idx]
        accy = sigs[4, start_idx:end_idx]
        accz = sigs[5, start_idx:end_idx]

        est, conf = predictor.predict(ppg1, ppg2, accx, accy, accz)

        error = np.abs(est - ref)

        errors.append(error)
        confidence.append(conf)

    # raise Exception
    return np.array(errors), np.array(confidence)

Test failed:
Did not pass. Error = 7.01

In [ ]:
```

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

```

In [2]: import troika
import scipy.io
from tqdm import tqdm
import os

def load_data(path, fs, window_len_s, window_shift_s, past_window, pass_band, multiplier):
    """
    Loads Troika data

    Args:
        path: (string) Path to data folder
        fs: (int) Sampling frequency
        window_len_s: (int) Window length in seconds
        window_shift_s: (int) Overlap between each window
        pass_band: ((float, float)) min and max pass bands tuple
        multiplier: (int) The number of frequencies should be multiplied by this.

    Returns:
        (list) targets
        (list) subjects
        (list) features
    """
    data_fls, ref_fls = troika.LoadTroikaDataset(path)

    pbar = tqdm(list(zip(data_fls, ref_fls)), desc="Data Preparation")

    targets, subjects, features = [], [], []

    for data_fl, ref_fl in pbar:
        sigs = scipy.io.loadmat(data_fl)['sig']
        refs = scipy.io.loadmat(ref_fl)['BPM0'].reshape(-1)
        subject_name = os.path.basename(data_fl).split('.')[0]

        # Bandpass Filter

        start_idx, end_idx = troika.get_idx(sigs.shape[1], len(refs), fs, window_len_s)
        for i, start_idx in enumerate(start_idx):
            end_idx = end_idx[i]

            # ECG-related
            ecg = sigs[0, start_idx:end_idx]
            ref = refs[i]

            # Extract features
            ppg1 = sigs[1, start_idx: end_idx]
            ppg2 = sigs[2, start_idx: end_idx]
            accx = sigs[3, start_idx: end_idx]
            accy = sigs[4, start_idx: end_idx]

```

```

        accz = sigs[5, start_idx:end_idx]

        feature = troika.featurize(ppg1, ppg2, accx, accy, accz,
                                    fs, pass_band, multiplier)

        targets.append(ref)
        subjects.append(subject_name)
        features.append(feature)

    return (targets, subjects, features)

# FS = 125
# WINDOW_LEN_S = 8
# WINDOW_SHIFT_S = 2
# PAST_WINDOW = 3
# PASS_BAND = (40/60.0, 240/60.0)
# MULTIPLIER = 4

# targets, subjects, features = \
#     load_data(
#         'datasets/troika/training_data',
#         FS,
#         WINDOW_LEN_S,
#         WINDOW_SHIFT_S,
#         PAST_WINDOW,
#         PASS_BAND,
#         MULTIPLIER)

# reg, val_scores = troika.prepare_regressor(features, targets, subjects,
#                                             print_log=True,
#                                             pickle_path='reg.pkl')

```

Data Preparation: 100%|| 12/12 [00:05<00:00, 2.27it/s]

```

Iter 0 score: 499.3944758387033
Iter 1 score: 650.9848622965882
Iter 2 score: 507.8492775977213
Iter 3 score: 526.0132209951809
Iter 4 score: 236.51416149438037
Iter 5 score: 115.8442792462028
Iter 6 score: 288.7096178269115
Iter 7 score: 184.27483355831112
Iter 8 score: 123.94025815394207
Iter 9 score: 1409.7123119414257
Iter 10 score: 529.621707711759
Iter 11 score: 248.85773012604008
Regressor stored at reg.pkl

```

```

In [3]: # replace the code below with your pulse rate algorithm.
import numpy as np
import scipy as sp
import scipy.io
import scipy.signal
import troika

FS = 125
WINDOW_LEN_S = 8
WINDOW_SHIFT_S = 2
PAST_WINDOW = 3
PASS_BAND = (40/60.0, 240/60.0)
MULTIPLIER = 4

def RunPulseRateAlgorithm1(data_fl, ref_fl, log_vars=False):
    """ Simple version, uses only the PPG data.
    """

    # Frequency of the signals sampled in Hertz.
    # For example, 125 means there are 125 samples every second.
    fs = FS # Hertz

    # Each reference pulse rate is calculated from this many seconds.
    window_len_s = WINDOW_LEN_S # seconds

    # Time difference (in seconds) between one time window and the next.
    window_shift_s = 2 # seconds

    # Pulse rate restrictions
    rest_min_fs = 1 / 40 / 60.0
    rest_max_fs = 1 / 240 / 60.0

    pass_band = (40/60.0, 240/60.0)

    # Load data using scipy. "scipy.io.loadmat(f)"
    sigs = scipy.io.loadmat(data_fl)['sig']
    refs = scipy.io.loadmat(ref_fl)['BPM0'].reshape(-1)

    errors = []
    confidence = []
    estimations = []

    largest_error = None

```

```

start_idx, end_idx = troika.get_idx(sigs.shape[1], len(refs), fs, window_len_s, w
for i, start_idx in enumerate(start_idx):
    end_idx = end_idx[i]

    # ECG-related
    ecg = sigs[0, start_idx:end_idx]
    ref = refs[i]

    # Start working on the signals below
    ppg = np.mean(sigs[1:3, start_idx: end_idx], axis=0)
    accx = sigs[3, start_idx:end_idx]
    accy = sigs[4, start_idx:end_idx]
    accz = sigs[5, start_idx:end_idx]

    if log_vars:
        ppg1 = sigs[1, start_idx:end_idx]
        ppg2 = sigs[2, start_idx:end_idx]
        log.prepare(ppg1=ppg1, ppg2=ppg2, ppg=ppg,
                    accx=accx, accy=accy, accz=accz,
                    ref=ref, ecg=ecg, fl=data_fl, i=i)

    # Bandpass Filter
    ppg = troika.bandpass_filter(ppg, pass_band, fs)
    accx = troika.bandpass_filter(accx, pass_band, fs)
    accy = troika.bandpass_filter(accy, pass_band, fs)
    accz = troika.bandpass_filter(accz, pass_band, fs)

    if log_vars:
        log.prepare(ppg1_bp=ppg1, ppg2_bp=ppg2, ppg_bp=ppg,
                    accx_bp=accx, accy_bp=accy, accz_bp=accz)

    # Get PPG power spectrums
    n = len(ppg) * MULTIPLIER
    freqs = np.fft.rfftfreq(n, 1/fs)
    fft = np.abs(np.fft.rfft(ppg, n))
    fft[freqs <= pass_band[0]] = 0.0
    fft[freqs >= pass_band[1]] = 0.0

    # Get max magnitude's frequency as the estimation
    est_fs = freqs[np.argmax(fft)]

    fs_window = 5 / 60.0
    est_fs_window = (freqs >= est_fs - fs_window) & (freqs <= est_fs + fs_window)
    conf = np.sum(fft[est_fs_window]) / np.sum(fft)

    error = np.abs(est_fs * 60 - ref)

```

```

        if log_vars and (largest_error is None or error > largest_error):
#         if log_vars and (i == 0 and 'DATA_12_TYPE02' in data_fl):
            log.prepare(ppg_fft_window=ppg_fft_window, acc_fft_window=acc_fft_window)
            log.commit()

            largest_error = error

            errors.append(error)
            confidence.append(conf)
            estimations.append(est_fs * 60)

    return np.array(errors), np.array(confidence)

def RunPulseRateAlgorithm(data_fl, ref_fl):
    """ Load the regression model """

    # Frequency of the signals sampled in Hertz.
    # For example, 125 means there are 125 samples every second.
    fs = FS # Hertz

    # Each reference pulse rate is calculated from this many seconds.
    window_len_s = WINDOW_LEN_S

    # Time difference (in seconds) between one time window and the next.
    window_shift_s = WINDOW_SHIFT_S

    # Zero-padding the frequencies by x times this.
    multiplier = MULTIPLIER

    # Bandpass filter limits
    pass_band = PASS_BAND

    predictor = troika.TroikaPredictor(
        'reg.pkl',
        fs,
        pass_band,
        multiplier,
        log_vars=False)

    # Load data using scipy. "scipy.io.loadmat(f)"
    sigs = scipy.io.loadmat(data_fl)['sig']
    refs = scipy.io.loadmat(ref_fl)['BPM0'].reshape(-1)

    errors = []
    confidence = []

```

```

start_idx, end_idx = troika.get_idx(sigs.shape[1], len(refs), fs, window_len_s, w

for i, start_idx in enumerate(start_idx):
    end_idx = end_idx[i]

    # ECG-related
    ecg = sigs[0, start_idx:end_idx]
    ref = refs[i]

    # Start working on the signals below
    ppg1 = sigs[1, start_idx:end_idx]
    ppg2 = sigs[2, start_idx:end_idx]
    accx = sigs[3, start_idx:end_idx]
    accy = sigs[4, start_idx:end_idx]
    accz = sigs[5, start_idx:end_idx]

    est, conf = predictor.predict(ppg1, ppg2, accx, accy, accz)

    error = np.abs(est - ref)

    errors.append(error)
    confidence.append(conf)

# raise Exception
return np.array(errors), np.array(confidence)

```

In []: