unit_test

May 14, 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.

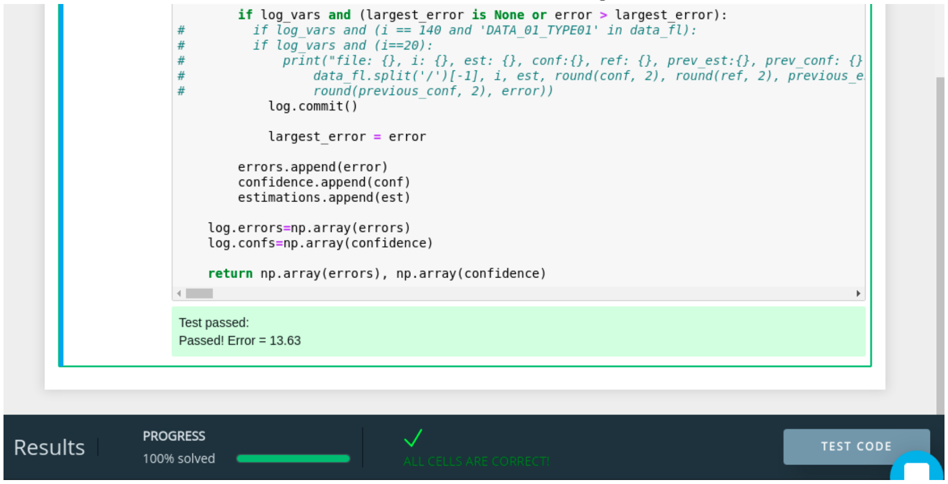
1.1.2 Pass

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.

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 [ ]:
```
# 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, 0.0246302094915053
#     pr_est = np.array([49.93333333333333, 49.96666666666667, 49.93333333333333, 49.933
#     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.stats
import scipy.io
import scipy.signal
import matplotlib.pyplot as plt
import pandas as pd

import sys
sys.path.append("../lib")
import troika
import pickle

# Fiddle with figure settings here:
```

```python
plt.rcParams['figure.figsize'] = (10,8)
plt.rcParams['font.size'] = 14
plt.rcParams['image.cmap'] = 'plasma'
plt.rcParams['axes.linewidth'] = 2
# Set the default colour cycle (in case someone changes it...)
from cycler import cycler
cols = plt.get_cmap('tab10').colors
plt.rcParams['axes.prop_cycle'] = cycler(color=cols)


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

def Predict(ppg, accx, accy, accz,
            previous_est=None, previous_conf=None,
              fs=125, pass_band=(40/60.0, 210/60.0), multiplier=8,
              ppg_mag_height=0.55, acc_mag_height=0.8, ppg_min_dist=0.2,
              num_best=2, acc_num_best_arg=2, conf_near=30/60.0, log_vars=False):
    """
    Create features based on some inputs.

    Args:
        ppg (numpy.array) PPG signals
        accx (numpy.array) IMU signals axis x
        accy (numpy.array) IMU signals axis y
        accz (numpy.array) IMU signals axis z
        fs: (int) Sampling frequency in Hz
        pass_band: ((float, float)) min and max pass bands tuple
        multiplier: (int) The number of frequencies should be multiplied by this
        ppg_mag_height: (float) `height` argument used in the scipy.signal.find_peaks()
        acc_mag_height: (float) `height` argument used in the scipy.signal.find_peaks()
        ppg_min_dist: (float) Magnitude distance between two highest PPG signals
        num_best: (int) Number of strongest signals to get
        conf_near: (float) What is considered near (Hz) when calculating confidence
        log_vars: (bool) Logging function

    Returns:
        List of features
    """
    global log


    if log_vars:
```

```
        log.prepare(ppg=ppg,
                    accx=accx, accy=accy, accz=accz)

    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)

    n = len(ppg) * multiplier
    freqs = np.fft.rfftfreq(n, 1/fs)

    # Get PPG power spectrums
    fft = np.abs(np.fft.rfft(ppg, n))
#     fft[freqs <= pass_band[0]] = 0.0
    fft[freqs <= 1.0] = 0.0
    fft[freqs >= pass_band[1]] = 0.0

    # Get L2-norms of accelerations
    acc_l2 = np.sqrt(accx ** 2 + accy ** 2 + accz ** 2)

    # Get acceleration power spectrums
    acc_fft = np.abs(np.fft.rfft(acc_l2, n))
#     acc_fft[freqs <= pass_band[0]] = 0.0
    acc_fft[freqs <= 1.0] = 0.0
    acc_fft[freqs >= pass_band[1]] = 0.0

    # Get max magnitude's frequency as one of the features
    if log_vars:
        log.prepare(ppg_bp=ppg,
                    accx_bp=accx, accy_bp=accy, accz_bp=accz,
                    freqs=freqs, fft=fft, acc_fft=acc_fft)

    peaks, _ = scipy.signal.find_peaks(fft,
                                       height=ppg_mag_height*np.max(fft),
                                       distance=1)
    max_ppg_fs = freqs[peaks]

    # Get max magnitude's acc_l2 as one of the features
    acc_peaks, _ = scipy.signal.find_peaks(acc_fft,
                                           height=acc_mag_height*np.max(acc_fft),
                                           distance=50)
    max_acc_fs = freqs[acc_peaks]

    # If there is no peak, simply get the largest PPG frequency
    if len(max_ppg_fs) == 0:
#         print("a")
        best_fit = freqs[np.argmax(fft)]
```

```
    # If there is another peak...
    elif len(max_ppg_fs) > 1:
#         print("b")
        max_ppg_ids = np.argpartition(fft[peaks], -num_best)[-num_best:]
        max_ppg_ids = max_ppg_ids[np.argsort(-fft[peaks][max_ppg_ids])]
        max_ppg_fs = max_ppg_fs[max_ppg_ids]
        best_fit = max_ppg_fs[0]

        # If distance between the best and second best is bigger than ppg_min_dist, get
        ppg_best_distance = fft[peaks][max_ppg_ids[0]] - fft[peaks][max_ppg_ids[1]]
        ppg_min_dist_val = fft[peaks][max_ppg_ids[0]] * ppg_min_dist
#         print("len(acc_peaks):", len(acc_peaks))
#         print("ppg_best_distance:", ppg_best_distance)
#         print("ppg_min_dist_val:", ppg_min_dist_val)
        if len(acc_peaks) > 0 and ppg_best_distance < ppg_min_dist_val:
#             print("start comparing best ppgs")
            # Get max PPG frequency closest to max acc frequency
            if len(acc_peaks) < acc_num_best_arg:
                acc_num_best = len(acc_peaks)
            else:
                acc_num_best = acc_num_best_arg
            max_acc_ids = np.argpartition(acc_fft[acc_peaks], -acc_num_best)[-acc_num_be
            max_acc_ids = max_acc_ids[np.argsort(-acc_fft[acc_peaks][max_acc_ids])]
            max_acc_fs = max_acc_fs[max_acc_ids]

            closest_i = 0
            closest_dist = np.inf
            for i in range(num_best):
#                 print("checking", i)
#                 print("  acc_peaks is", acc_peaks)
#                 print("  acc_fft[acc_peaks] is", acc_fft[acc_peaks])
#                 print("  max_acc_id is", max_acc_id)
#                 print("  max_acc_fs is", max_acc_fs)
#                 print("  ppg freq:", max_ppg_fs[i])
                # Distance between acc and ppg
                dist = 0
#                 print("  dist:", dist)
                if closest_dist > dist:
#                     print("  set closest")
                    for j in range(acc_num_best):
                        # Distance between acc and ppg
                        dist_j = np.abs(max_acc_fs[j] - max_ppg_fs[i])
                        dist += dist_j

                    closest_dist = dist
                    closest_i = i
#             print("closest_i is", closest_i)
            best_fit = max_ppg_fs[closest_i]
```

```
        if log_vars:
            log.prepare(peaks=peaks,
                        max_ppg_ids=max_ppg_ids,
                        max_ppg_ids_0=max_ppg_ids[0],
                        max_ppg_ids_1=max_ppg_ids[1],
                        best_fft=fft[peaks][max_ppg_ids[0]],
                        second_best_fft=fft[peaks][max_ppg_ids[1]],

                        max_acc_id=max_acc_id,
                        max_acc_fs=max_acc_fs,
                        best_acc=acc_fft[acc_peaks][max_acc_id]
                       )

    # If there is only one peak, use it.
    else:
#         print("c")
        best_fit = max_ppg_fs[0]

    if log_vars:
        log.prepare(best_fit=best_fit)

#     print(max_ppg_fs)
#     print(freqs[peaks])
#     print(fft[peaks])
#     print("best_fit: {}".format(best_fit))

    conf = CalcConfidence(best_fit, freqs, fft, near=conf_near)

    est = best_fit * 60

    return (est, conf)

def CalcConfidence(candidate, freqs, fft, near=5/60.0):
    candidate_fs_ids = (freqs >= candidate - near) & (freqs <= candidate + near)
    candidate_fs = freqs[candidate_fs_ids]
#     print(candidate_fs)
    candidate_mag = fft[candidate_fs_ids]
#     print(candidate_mag)
    candidate_pow = np.sum(candidate_mag)
#     print(candidate_pow)
    candidate_pow_relative = candidate_pow / sum(fft)
#     print(candidate_pow_relative)
    return candidate_pow_relative

largest_error = None
def RunPulseRateAlgorithm(data_fl, ref_fl, log_vars=True):
    """
```

```python
    Run Pulse Rate Algorithm

    Loads data and model, then calculate errors and confidence rates from the data.

    Args:
        data_fl: (string) Path to data file (MATLAB data)
        ref_fl: (string) Path to reference data file (MATLAB data)
        log_vars: (bool) If True, log some variables in the `log` object for further ana

    Returns:
        (np.array) Error scores
        (np.array) Confidence rates
    """
    global log, largest_error

    # 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


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

    if log_vars:
        log.prepare(sigs=sigs, refs=refs, fl=data_fl)

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



    start_idxs, end_idxs = troika.get_idxs(sigs.shape[1], len(refs),
                                           fs=fs,
                                           window_len_s=window_len_s,
```

```
                                          window_shift_s=window_shift_s)

    previous_est, previous_conf = None, None

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

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

        # Start working on the signals below
        # Note: Use only the 2nd PPG signal for this exercise as the 1st one is too clos
        ppg = 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 = Predict(ppg, accx, accy, accz, previous_est=previous_est, previous_c
        previous_est = est
        previous_conf = conf


        if log_vars:
            log.prepare(i=i, ref=ref, ecg=ecg, est=est, conf=conf)

        error = np.abs(est - ref)
#         print("e: {}, r: {}, er: {}".format(est, ref, error))

        if log_vars and (largest_error is None or error > largest_error):
#         if log_vars and (i == 140 and 'DATA_01_TYPE01' in data_fl):
#         if log_vars and (i==20):
#             print("file: {}, i: {}, est: {}, conf:{}, ref: {}, prev_est:{}, prev_conf:
#                 data_fl.split('/')[-1], i, est, round(conf, 2), round(ref, 2), previou
#                 round(previous_conf, 2), error))
            log.commit()

            largest_error = error

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

    log.errors=np.array(errors)
    log.confs=np.array(confidence)

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