Test Your Algorithm

Instructions

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

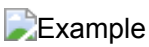
    - Copy over all the **Code** section to the following Code block.
    - Download as a Python (`.py`) and copy the code to the following Code block.
2. In the bottom right, click the Test Run button.

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.

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**.

 Example

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.

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

 Passed

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

In [2]:
```
import glob
from tqdm import tqdm
import numpy as np
import scipy as sp
import scipy.io

import scipy.signal
import os.path
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.metrics import mean_squared_error
def LoadTroikaDataset():
    """
    Retrieve the .mat filenames for the troika dataset.

    Review the README in ./datasets/troika/ to understand the organizat
ion of the .mat files.

    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 r
ef_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):
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    """
    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 corresponding
            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 es
timates
    #    are above the 10th percentile confidence.
    percentile90_confidence = np.percentile(confidence_est, 10)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    # Find the errors of the best pulse rate estimates
    best_estimates = pr_errors[confidence_est >= percentile90_confidenc
e]

    # 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 metric.

    Returns:
        Pulse rate error on the Troika dataset. See AggregateErrorMetri
c.
    """
    # 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)

def RunPulseRateAlgorithm(data_fl, ref_fl):

    Fs = 125 # Sample Frequency
    window_len = 8 # Window to calculate PR
    window_shift = 2  # Difference between windows

    reg, scores = Regressor()
    targets, features, sigs, subs = Data_window8(data_fl, ref_fl)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    error, confidence = [], []
    for i,feature in enumerate(features):
        est = reg.predict(np.reshape(feature, (1, -1)))[0]

        ppg, accx, accy, accz = sigs[i]

        ppg = Filter(ppg)
        accx = Filter(accx)
        accy = Filter(accy)
        accz = Filter(accz)

        n = len(ppg) * 3
        freq = np.fft.rfftfreq(n, 1/Fs)
        fft = np.abs(np.fft.rfft(ppg,n))
        fft[freq <= 40/60.0] = 0.0
        fft[freq >= 240/60.0] = 0.0

        est_Fs = est / 55.0
        Fs_win = 30  / 60.0
        Fs_win_e = (freq >= est_Fs - Fs_win) & (freq <= est_Fs +Fs_win)
        conf = np.sum(fft[Fs_win_e])/np.sum(fft)

        error.append(np.abs((est-targets[i])))
        confidence.append(conf)
    return np.array(error), np.array(confidence)

def Data_window8(data_fl, ref_fl):

    Fs=125  # Sampling frequency
    window_len = 6 # Window to calculate PR
    window_shift = 2 # Difference between windows

    sig = LoadTroikaDataFile(data_fl)
    ref = scipy.io.loadmat(ref_fl)["BPM0"]
    ref = np.array([x[0] for x in ref])
    subject_name = os.path.basename(data_fl).split('.')[0]
    start_indxs, end_indxs = Indexator(sig.shape[1], len(ref), Fs, wind
ow_len,window_shift)
    targets, features, sigs, subs = [], [], [], []
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    for i, s in enumerate(start_indxs):
        start_i =  start_indxs[i]
        end_i = end_indxs[i]

        ppg = sig[0, start_i:end_i]
        accx = sig[1, start_i:end_i]
        accy = sig[2, start_i:end_i]
        accz = sig[3, start_i:end_i]

        ppg = Filter(ppg)
        accx = Filter(accx)
        accy = Filter(accy)
        accz = Filter(accz)

        feature, ppg, accx, accy, accz = CreateFeature(ppg, accx, accy,
 accz)

        sigs.append([ppg, accx, accy, accz])
        targets.append(ref[i])
        features.append(feature)
        subs.append(subject_name)

    return (np.array(targets), np.array(features), sigs, subs)

def Data_window6():

    Fs=125 # Sampling rate
    window_len = 6 # Window to calculate PR
    window_shift = 2 # Difference between windows

    data_fls, ref_fls = LoadTroikaDataset()
    pbar = tqdm(list(zip(data_fls, ref_fls)), desc="Prepare Data")
    targets, features, sigs, subs = [], [], [], []
    for data_fl, ref_fl in pbar:
        sig = LoadTroikaDataFile(data_fl)
        ref = scipy.io.loadmat(ref_fl)["BPM0"]
        ref = np.array([x[0] for x in ref])
        subject_name = os.path.basename(data_fl).split('.')[0]
        start_indxs, end_indxs = Indexator(sig.shape[1], len(ref), Fs,
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
window_len,window_shift)
        for i, s in enumerate(start_indxs):
            start_i =  start_indxs[i]
            end_i = end_indxs[i]

            ppg = sig[0, start_i:end_i]
            accx = sig[1, start_i:end_i]
            accy = sig[2, start_i:end_i]
            accz = sig[3, start_i:end_i]


            ppg = Filter(ppg)
            accx = Filter(accx)
            accy = Filter(accy)
            accz = Filter(accz)

            feature, ppg, accx, accy, accz = CreateFeature(ppg, accx, a
ccy, accz)

            sigs.append([ppg, accx, accy, accz])
            targets.append(ref[i])
            features.append(feature)
            subs.append(subject_name)

    return (np.array(targets), np.array(features), sigs, subs)

def CreateFeature(ppg, accx, accy, accz):
    """ Create features """
    ppg = Filter(ppg)
    accx = Filter(accx)
    accy = Filter(accy)
    accz = Filter(accz)


    Fs = 125
    n = len(ppg) * 4
    freq = np.fft.rfftfreq(n, 1/Fs)
    fft = np.abs(np.fft.rfft(ppg,n))
    fft[freq <= 40/60.0] = 0.0
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    fft[freq >= 240/60.0] = 0.0

    acct = np.sqrt(accx**2 + accy**2 + accz**2) # Total signal of acc

    acc_fft = np.abs(np.fft.rfft(acct, n))
    acc_fft[freq <= 40/60.0] = 0.0
    acc_fft[freq >= 240/60.0] = 0.0

    ppg_feature = freq[np.argmax(fft)]
    acc_feature = freq[np.argmax(acc_fft)]

    return (np.array([ppg_feature, acc_feature]), ppg, accx, accy, accz
)

def RegressionAlg(features, targets, subs):
    """ The regression model"""
    AdaBoostRegressor
    regression = RandomForestRegressor(n_estimators=400,max_depth=16)
    scores = []
    lf = KFold(n_splits=5)
    splits = lf.split(features,targets,subs)
    for i, (train_idx, test_idx) in enumerate(splits):
        X_train, y_train = features[train_idx], targets[train_idx]
        X_test, y_test = features[test_idx], targets[test_idx]
        regression.fit(X_train, y_train)
        y_pred = regression.predict(X_test)
        score = Error(y_test, y_pred)
        scores.append(score)

    return (regression, scores)

def Filter(signal):
    """Bandpass filter between 40 and 240 BPM"""
    pass_band=(40/60.0, 240/60.0)
    Fs = 125
    b, a = scipy.signal.butter(3, pass_band, btype='bandpass', fs=Fs)
    return scipy.signal.filtfilt(b, a, signal)

def Indexator(sig_len, ref_len, Fs=125, window_len_s=10, window_shift_s
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
=2):
    """
    Find start and end index to iterate over a set of signals
    """
    # Set the length of the biggest signal with regards to the referenc
e signal
    if ref_len < sig_len:
        n = ref_len
    else:
        n = sig_len

    # Start Indexes
    start_indxs = (np.cumsum(np.ones(n) * Fs * window_shift_s) - Fs * w
indow_shift_s).astype(int)

    # End Indexes (same size as the start indexes array)
    end_indxs = start_indxs + window_len_s * Fs
    return (start_indxs, end_indxs)

def Predict(reg,feature, ppg, accx, accy, accz):
    """Predict based on the regressor"""
    est = reg.predict(np.reshape(feature, (1, -1)))[0]

def Error(y_test, y_pred):
    """
    Calculate error score of a Prediction
    """
    return mean_squared_error(y_test, y_pred)


def Regressor():
    fname = "outfile.npy"
    reg, scores = [], []
    if os.path.isfile(fname):
        [reg,scores] = np.load(fname,allow_pickle=True)
        #
    else:
        targets, features, sigs, subs = Data_window6()
        reg, scores = RegressionAlg(features, targets, subs)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        np.save("outfile", [reg,scores])
    return reg, scores
```

In [ ]:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD