Physionet Hz Converter

November 23, 2021

[1]:
```
import wfdb
import numpy as np
import os
import pandas as pd
from random import randint
import math
import h5py
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
from collections import Counter
from wfdb.processing import resample_multichan
import glob
import sys
import traceback
from functools import reduce
```

[6]:
```
def get_file_list(BASE_DIR):
    record_files = []
    #file_endings = ['.dat', '.hea', '.xyz']
    with open(os.path.join(BASE_DIR, 'RECORDS')) as recs:
        record_files = recs.read().splitlines()
        recs.close()
    return record_files

def get_file_path_list(base_dir, file_endings=['.dat'], remove_extension=True):
    record_file_paths = []
    for end in file_endings:
        end_set = []
        for path, subdirs, files in os.walk(base_dir):
            for name in files:
                if name.endswith(end):
                    if remove_extension:
                        end_set.append(os.path.splitext(os.path.join(path,␣
 ↪name))[0])
                    else:
                        end_set.append(os.path.join(path, name))
        record_file_paths.append(set(end_set))
```

```
    if remove_extension:
        return list(reduce(lambda a, b: a & b, record_file_paths))
    else:
        return list(reduce(lambda a, b: a | b, record_file_paths))

def read_record_infos(record_path):
    record = wfdb.rdrecord(record_path)
    return {
        'record_name' : record.record_name,
        'file_name' : record.file_name,
        'record_comments' : record.comments,
        'number_of_signals' : record.n_sig,
        'is_physical_signal' : not (record.p_signal is None),
        'is_digital_signal' : not (record.d_signal is None),
        'signal_sampling_frequency' : record.fs,
        'signal_length' : record.sig_len,
        'signal_channel_names' : record.sig_name,
        'signal_channel_units' : record.units
    }

def read_annotation(record_path, physical=True,␣
↪return_label_elements=['symbol', 'label_store', 'description']):
    try:
        annotation = wfdb.rdann(record_path, 'atr',␣
↪return_label_elements=return_label_elements)
        #print(record_path)
        #print('sample:', annotation.sample, 'symbol', annotation.symbol,␣
↪'contained labels', annotation.description)
        return (annotation.sample, annotation.symbol, annotation.label_store,␣
↪annotation.description)
    except ValueError as ve:
        print(record_path, ' annotation read failed:', ve)
        return None

def read_annotation_object(record_path, physical=True,␣
↪return_label_elements=['symbol', 'label_store', 'description']):
    try:
        annotation = wfdb.rdann(record_path, 'atr',␣
↪return_label_elements=return_label_elements)
        return annotation
    except ValueError as ve:
        print(record_path, ' annotation read failed:', ve)
        return None

def read_signal(record_path, physical=True):
    #print(record_path)
```

```
    record = wfdb.rdrecord(record_path, physical=physical)
    #print_object_attributes(record)
    if physical:
        data = record.p_signal
    else:
        data = record.d_signal
    return data

def resample_frequency(record_signal_array, annotation_object, hz_frq_in:int,␣
↪hz_frq_out:int):
    return resample_multichan(record_signal_array, annotation_object,␣
↪hz_frq_in, hz_frq_out, resamp_ann_chan=0)

def resample_frequency_all(record_file_paths, hz_frq_out, output_filepath,␣
↪physical=True, resample_annotation=True, overwrite=False):
    print('Resamling of files has started:', record_files)
    if not os.path.exists(output_filepath):
        os.makedirs(output_filepath)
    for record_path in record_file_paths:
        file = os.path.basename(record_path)
        print("Resampling:", file)
        if overwrite or not glob.glob(os.path.join(output_filepath, file) +'.
↪*'):
            file_infos = read_record_infos(record_path)
            print("read record infos")
            hz_frq_in = file_infos['signal_sampling_frequency']
            record_signal = read_signal(record_path, physical=physical)
            print("read record signal", record_signal.shape)

            try:
                if resample_annotation:
                    annotation_object = read_annotation_object(record_path,␣
↪physical=physical, return_label_elements=['symbol'])#, 'label_store'])␣
↪cannot use, wfdb is buggy
                    print("read record annotation")
                    resampled_signal, resampled_ann =␣
↪resample_frequency(record_signal, annotation_object, hz_frq_in, hz_frq_out)␣
↪#Throws random assertionerrors at times
                    print("computed resampled signal", resampled_signal.shape)
                    #wfdb.wrann(file+"resampled", 'atr', sample=resampled_ann.
↪sample, label_store=annotation_object.label_store, write_dir=output_filepath)
                    a_temp = wfdb.Annotation(file, 'atr', resampled_ann.sample,␣
↪symbol=annotation_object.symbol, fs=hz_frq_out,␣
↪label_store=annotation_object.label_store)
                    a_temp.wrann(write_fs=True, write_dir=output_filepath)
                else:
```

```
                    resampled_signal, _ = resample_sig(record_signal,␣
↪hz_frq_in, hz_frq_out)
                try:
                    if physical:
                        wfdb.wrsamp(file, fs=hz_frq_out,␣
↪p_signal=resampled_signal, sig_name=file_infos['signal_channel_names'],␣
↪units=file_infos['signal_channel_units'],␣
↪comments=file_infos['record_comments'], write_dir=output_filepath)
                    else:
                        wfdb.wrsamp(file, fs=hz_frq_out,␣
↪d_signal=resampled_signal, sig_name=file_infos['signal_channel_names'],␣
↪units=file_infos['signal_channel_units'],␣
↪comments=file_infos['record_comments'], write_dir=output_filepath)
                    print('finished', output_filepath)
                except (IndexError, ValueError) as ive: #wfdb = bug
                  print(ive)
            except AssertionError: #https://stackoverflow.com/questions/
↪11587223/
↪how-to-handle-assertionerror-in-python-and-find-out-which-line-or-statement-it-o
                _, _, tb = sys.exc_info()
                traceback.print_tb(tb) # Fixed format
                tb_info = traceback.extract_tb(tb)
                filename, line, func, text = tb_info[-1]

                print('An error occurred on line {} in statement {}'.
↪format(line, text))

        else:
          print("skipping", file)
```

0.0.1 Download the database

[3]:
```
db_dir = 'ptb-xl' #The Database to download (see get_dbs())
temp_directory = 'out'
save_to_drive = False
download_annotation = False #If the annotation/labels should be downloaded
hz_frq_out = 1000
#wfdb.dl_database(db_dir, temp_directory, records='all', annotators='all' if␣
↪download_annotation else None)
```

0.0.2 Resample all downloaded files

[ ]:
```
record_files = get_file_path_list('/media/julian/Volume/data/ECG/
↪ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
↪records500')
```

```
resample_frequency_all(record_files, hz_frq_out=hz_frq_out, output_filepath=os.
 ↪path.join('/media/julian/Volume/data/ECG/
 ↪ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
 ↪generated', str(hz_frq_out)), resample_annotation=download_annotation)
```

[ ]:
```
def generate_RECORDS_file(record_file_root_dir):
    record_files = get_file_path_list(record_file_root_dir, file_endings=['.
 ↪dat', '.hea'])
    output_path = record_file_root_dir
    with open(os.path.join(output_path, "RECORDS"), 'w') as f:
        f.write('\n'.join([os.path.basename(p) for p in record_files]))
generate_RECORDS_file('/media/julian/Volume/data/ECG/
 ↪ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
 ↪generated/1000')
```

[5]:
```
import numpy as np
from scipy import signal
import pandas as pd
def resample_sig(x, fs, fs_target): #https://github.com/MIT-LCP/wfdb-python/
 ↪blob/master/wfdb/processing/basic.py
    """
    Resample a signal to a different frequency.
    Parameters
    ----------
    x : ndarray
        Array containing the signal.
    fs : int, float
        The original sampling frequency.
    fs_target : int, float
        The target frequency.
    Returns
    -------
    resampled_x : ndarray
        Array of the resampled signal values.
    resampled_t : ndarray
        Array of the resampled signal locations.
    """
    t = np.arange(x.shape[0]).astype('float64')

    if fs == fs_target:
        return x, t

    new_length = int(x.shape[0]*fs_target/fs)
    # Resample the array if NaN values are present
    if np.isnan(x).any():
        x = pd.Series(x.reshape((-1,))).interpolate().values
    resampled_x, resampled_t = signal.resample(x, num=new_length, t=t)
```

```
    assert np.all(np.diff(resampled_t) > 0)
    #deleted assert to make it work again
    return resampled_x, resampled_t
```

[24]:
```
p = os.path.join('/media/julian/Volume/data/ECG/
 ↪ptb-xl-a-large-publicly-available-electrocardiography-dataset-1.0.1/
 ↪generated', str(hz_frq_out))
dat_files = set([os.path.splitext(d)[0] for d in glob.glob(os.path.join(p, '*.
 ↪dat'))])
hea_files = set([os.path.splitext(d)[0] for d in glob.glob(os.path.join(p, '*.
 ↪hea'))])
```

[22]:
```
for f in (dat_files - hea_files) | (hea_files - dat_files):
    for ff in glob.glob(f+'.hea') + glob.glob(f+'.dat'):
        print('removing:', ff)
        os.remove(ff)
```

[ ]:
```
```