libCCE

A C implementation (with a python wrapper) of a library used to calculate Corrected Conditional Entropy (CCE) for a given string of values.

Build requirements

A Python3 install with dev headers

A C compiler

```
sudo apt-get install python-dev build-essential clang
```

Build Process

In place build (local)

```
python3 build.py build_ext --inplace
```

System-wide build

This has the potental to require administrator priviliges:

```
python3 build.py install
```

If admin is required, use `sudo`'s `-H` flag:

```
sudo -H python3 build.py install
```

Development Reqirements

```
pip install Cython
```

In `build.py` uncomment the following lines so that any changes made to `.pyx` files will be added to the generated C code:

```
from Cython.Build import cythonize
ext_modules = cythonize(ext_modules)
```

Development Suggestions

If you are doing development work on this, there are a few things to keep in mind:

- If you are working on the C code, compiling with `-fsanitize=address` is **highly suggested** during development. It catches issues that you might not be even aware of and saves a lot of headaches.
- If you are working on the C code, ompiling with `-g -fno-omit-frame-pointer` is also recomended.
- If the editor or compiler you are using does not give you line numbers, using IDA or Cutter can be helpful for figuring out where exactly you error is (just jump to the address and poke around).
- If you are working on the Cython code, building via `cython <inputfile>.pyx -a` every once in a while might be useful to see what you can do to minimize interaction with python objects (lines highlighted in yellow).

Also, keep in mind that anything you write in a `cdef` function is effectivey straight C with all that comes with it.

- Adding the following lines to they `.pyx` files will help with debugging by adding line tracing code:
  ```
  # cython: linetrace=True
  # distutils: define_macros=CYTHON_TRACE_NOGIL=1
  ```

API

The `libCCE` API consists of the `CCE` class and the `localMinCCE` function. All functions and methods are well commented in the source code. The high level docstrings and their associated methods are placed here for ease of finding.

Class CCE:

This class is a wrapper for the C CCE implementation which is based on the work done by S. Gianvecchio and H. Wang's article on detecting covert timing channels using an entropy-based approach. In particular, please refer to section 3.4 (this section describes CCE implementation using a q-ary tree of height m) in the following publication: S. Gianvecchio, H. Wang, "Detecting Covert Timing Channels: An entropy-based Approach", *Proceedings of CCS'07*, Oct.29-Nov.2, 2007. The article can be found here.

This class calculates Corrected Conditional Entropy (CCE) of a given sequence. The inital body of samples can be provided to `__init__` and they will be populated into the tree using a faster pure C method. Any new data can be provided to `insertSequence` which will insert it into the tree.

As a general rule this implementation will try and fail via assertion before returining compromised information to the user.

```
def __cinit__(self, unsigned int branchingFactor, initalSequences = None, int subSeqLen = 50):
    Called on creation of object, if initalSequences is supplied, the sequences will be
    populated to the tree via a faster version of insertSequence.

    Args:
        branchingFactor (int):
            The branching factor of the tree, this is equivalent to the number of bins you have.
        initalSequences (iterable):
            A python iterable which contains Inter-Packet Delays(IPDs) for insertion.
        subSeqLen (int):
            The length you would like the subsequences devided into.

    Raises:
        ValueError:
            if subSeqLen is less than the length of the original
            sequence or if the branching factor is less than or equal to zero.

def __init__(self, branchingFactor, initalSequence, subSeqLen):
    All real work is done in __cinit__

def __dealloc__(self):
    Used to cleanup memory on destruction of object. This method is automatically
    called. It should NOT be manually called it as it WILL break everything.

def insertSequence(self, pySeq):
    Wrapper for insertSequence, converts a supplied python
    iterable which supports indexing and len() into an integer
    buffer of the same length.

    Args:
        pySeq(list or tuple):
            A python iterable that supports indexing and len()

def calculateCCE(self):
```

```
    Calculate the CCE for all inserted sequences and return the
    CCE for all sub-sequence lengths upto the maximum sub-sequence length.

    Returns:
        CCEs(list):
            A list of floats containing the CCE for each layer
            of the tree. This list will only be as long as the
            number of unique sequences.

def resetTree(self):
    This function can be used to reset the tree and reuse it
    by freeing all the memory assocated with the tree.

    This can be very useful when you want to calculate new data
    that is of a similar size to the previously-analized data.
```

Function localMinCCE:

This is a faster (although most likely less accurate) implementation of CCE, which only computes a local minimum.

It is based on the following publication: A. Porta, G. Baselli, D. Liberati, N. Montano, C. Cogliati, T. Gnecchi-Ruscone, A. Malliani, and S. Cerutti, “Measuring Regularity by Means of a Corrected Conditional Entropy in Sympathetic Outflow,” *Biological Cybernetics*, vol. 78, no. 1, pp. 71-78, Jan. 1998. The article can be found here.

This function calculates the CCE of a sequence using the local minimum as an early stopping metric. It still uses the same C library and the same tree approach to calculating CCE, but stops when it finds a local minimum.

```
def localMinCCE(seq, int maxLen):

    Args:
        seq (iterable):
            An iterable containing the pre-binned values of your IPD sequence.
        maxLen (int):
            The maximum subsequence length to be tried.  This is
            more of an upper bound as the local minimum is usually
            found before this is reached.

    Returns:
        minCCE (float):
            The local minimum CCE of the provided sequence.
```