A Faster Algorithm for Calculating the Sample Entropy

A faster algorithm for calculating the sample entropy of physiological signals

Gašper Fele-Žorž
Faculty of Computer and Information Science
Tržaška cesta 25
Ljubljana, Slovenia
polz@fri.uni-lj.si

ABSTRACT

In this paper, we present an overview of different methods for calculating a measure of signal complexity called sample entropy. We then present a way to improve the most widely used algorithm by using a skip-list.

Categories and Subject Descriptors

I.5.4 [**Applications**]: Signal processing

Keywords

sample entropy, skip list

1. INTRODUCTION

In nature, many processes are quite unpredictable. The level of unpredictability can sometimes tell us a lot about the system under observation. Research has shown, for example, that the heart-beats of people with some heart problems tend to be more regular than the heart-beats of healthy individuals.

Unfortunately, the most widely used method currently for calculating the sample entropy of signals is too slow to allow anything but offline processing of signals.

2. METHODS

Sample entropy is a measure of predictability of a time series. Apart from the time series itself, it also depends on two parameters, template length m and tolerance r. The less predictable time series, the higher it's sample entropy will be.

Suppose we have a finite time series $y(0, \ldots, N-1)$ of length N. Given a constant $m; m << N$, $y(t)$ can be divided into $N-m+1$ pattern templates $x_t(0, \ldots m-1)$ of length m so that $x_t(i) = y(t+i)$ for each $i = 0 \ldots m-1$ and for each $t = 0 \ldots N-m$.

Given a positive tolerance r, we can consider x_{t2} to match x_{t1} if $|x_{t1}(i) - x_{t2}(i)| \leq r$ for each $i = 0 \ldots m-1$.

The predictability of a time series can then be estimated by counting the number of matching templates for each template within a time series. An example time series with

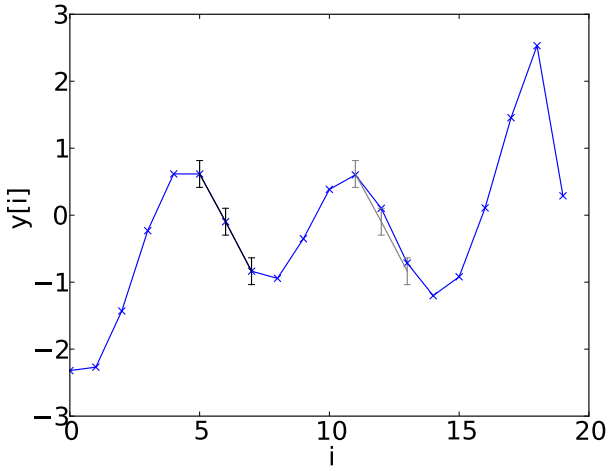

Figure 1: A time series with a template of length 3 at $i = 5$ and it's match within $r = 0.2$

a template m=3 samples long and it's matching template within $r = 0.2$ are shown in Figure 1.

Sample entropy depends on the probability that if for some template, a matching template of length m within a margin of r is found within a time series, then a matching template of length $m+1$ within a margin of r is also found. If no matching templates are found in the whole signal, sample entropy depends only on the length of the time series N and the template length $m+1$. The templates of length m at the end of a time series $(y(N-m) \ldots y(N-1))$ are a special case since there can be no matching templates of length $m+1$ corresponding to these matches.

More precisely, if A_m is the number of matches for all templates of length m within a time series $y(t)$ and B_m is the number of matches for all templates of length m except the ones at the end of the time series $y(t)$, sample entropy is

```
sampEn(y, r, M):
    N=len(y);
    lastruns=zeros(N);
    runs=zeros(N);
    A=zeros(M);
    B=zeros(M);
    e=zeros(M);
    for i in 0...N-1:
        y1=y[i];
        for j in i+1...N-1:
            if |y[j]-y1| < r:
                runs[j]=lastruns[j-1]+1
                for m in 0...min(M, run[j])):
                    A[m]+=1
                    if j<N-1:
                        B[m] += 1
            else:
                run[j]=0;
        (runs, lastruns) = (lastruns, runs)
    B = concatenate([N*(N-1)/2], B[:-1])
    p=A/B;
    return -log(p)
```

Figure 2: The most widely used algorithm for calculating the sample entropy of a signal

defined [3] as:

$$sampEn_{m,r}(y) = \begin{cases} -log(\frac{A_m}{B_{(m-1)}}) & : A_m \neq 0 \wedge B_{m-1} \neq 0 \\ -log(\frac{(N-m)}{(N-m-1)}) & : A_m = 0 \vee B_{m-1} = 0 \end{cases} \quad (1)$$

For $m = 0$, B_m is set to $\frac{N \cdot (N-1)}{2}$.

2.1 Calculating the sample entropy in $O(N^2)$ time

The obvious naïve algorithm for calculating sample entropy is to count the number of matches for each template of length m. Assuming that m is much smaller than N, the time complexity of this algorithm is

$$O\left(\frac{N^2}{2} \cdot (1 - (1 - p_r)^m)\right)$$

where p_r represents the probability of two samples, $y(i)$ and $y(j)$, being within r of each other. For typical physiological time series, r will be chosen so that p_r is usually small.

The algorithm above can typically be improved by a factor of close to $\frac{1-(1-p_r)^m}{p_r}$ by storing the runs of matches in an array of length N. The improved algorithm also calculates the sample entropy for every $m \leq M$ at no extra cost:

The time complexity of this algorithm is $O(N \cdot \frac{N}{2} \cdot (1 + p_r \cdot m))$

An example calculation of sample entropy is shown in Figure 3. The time series used, y, is the same as in Figure 1. The table runs is shown for each iteration through y. This seems to be the standard algorithm most researchers use. An implementation in C is available on Physionet [1].

2.2 Calculating the sample entropy using K-D trees

Recently, an improved method has been presented for calculating sample entropy by [2]. The method works by using each template of length m starting at index i within a signal y as a point p_i in m-dimensional space. $N - m$ points are constructed from a signal of length N. All points are inserted into a K-D tree of dimension $d = m$. The number of matches within r for each point are then counted.

Since range counting in K-D trees can be done in $O(N^{1-(1/d)})$ time and a K-D tree can be constructed in $O(NlogN)$ time, this algorithm is noticeably faster than the most commonly used algorithm.

This algorithm has been improved further by [4]. Instead of inserting all points into a single K-D tree and then performing queries against the whole tree, the number of points in the tree during each query can be drastically reduced by first removing all points from the tree and sorting the points by the first dimension $p(0)$.

The sorted points are then inserted back into the tree one by one until $p_i(0) > min(p(0)) + 2r$. Matches are then counted for each point in order of $p(0)$. When $p_i(0)$ reaches $min(p(0)) + r$, the points with the lowest $p(0)$ are removed from the K-D tree and all the points with $p(0) < p_i(0) + r$ are inserted.

This way, the K-D tree is kept small so the counting of matches can be done faster.

Due to the curse of dimensionality, these methods should perform poorly for large values of m. Since m is usually low, this is unlikely to present a problem.

2.3 Calculating the sample entropy using a skip-list

As depicted in Figure 3, typically, a small number of samples in y are within r of each other. The standard algorithm for calculating the sample entropy can be improved by skipping over those parts of y where the begginings of patterns do not match.

```
sampEn(y, r, M):
    N = len(y)
    lastrun = zeros(N) with history of length M;
    run = zeros(N) with history of length M;
    A = zeros(M)
    B = zeros(M)
    L = OrderedStructure
    for i in 0...N-1:
        L[y[i]] = i
    for i in 0...N-1:
        y1 = y[i];
        runs.history.remove_oldest()
        for j in run.history:
            runs[j] = 0
        sums = zeros(M)
        for j in L[y1 - r]...L[y1 + r]:
            runs[j] = lastruns[j-1]+1
            runs.history.newest.add(j)
            sums[min(M, runs[j])] += 1
        m_sum = 0
```

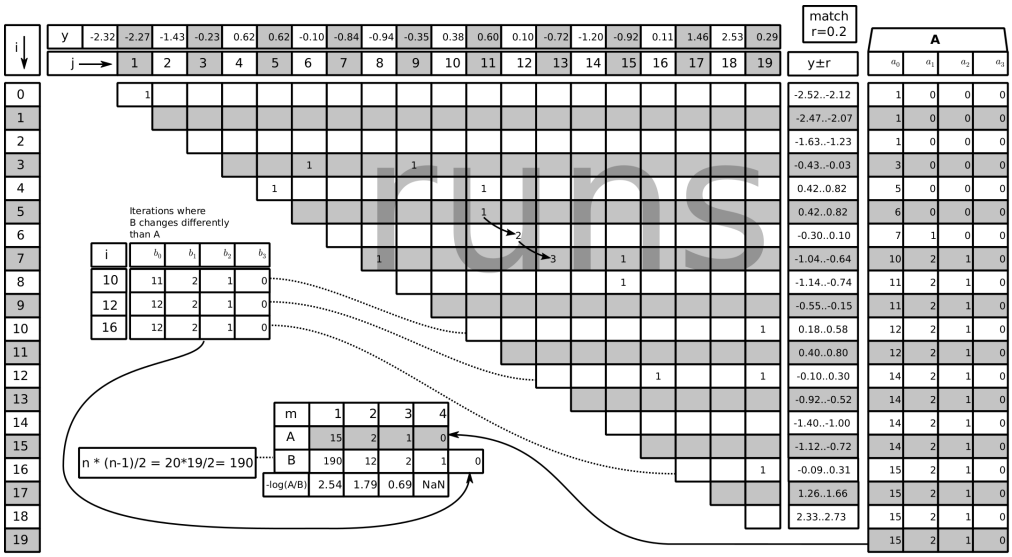

Figure 3: An example of sample entropy calculation.

```
    for m in M...0 step -1:
        m_sum += sums[0]
        A[m] += m_sum
        if j<N-1:
            B[m] += m_sum
    (runs, lastruns) = (lastruns, runs)
B = concatenate([N*(N-1)/2], B[:-1])
p=A/B;
return -log(p)
```

The time complexity thus becomes $O(N \cdot A_{L0} + N \cdot (A_{L1} + p_r \cdot (N/2 \cdot A_{L2} + m)))$ where A_{L0} denotes the time to add an element to L, A_{L1} denotes the time to access the first sample $y(j)$ which is greater than $y(i) - r$ and A_{L2} denotes the time to access each subsequent sample. The expected number of matches for each $y(i)$ is $p_r * N/2$.

For the algorithm to offer an improvement, the data structure L would have to offer certain performance guarantees. The value of the sample would serve as the key and it's position in the original time-series would serve as the value. In this data structure, the following operations would have to be fast ($O(\log(N))$ or better):

1. the storage of multiple instances with the same key (A_{L0}),
2. the retreival of the first instance whose key is above a certain limit (A_{L1}),
3. given an instance, the retreival of the next-higher instance (A_{L2}).

Preferably, the structure would also be simple to implement.

We decided to use a modified skip list. Normally, each node in a skip list contains a single value. Unfortunately, on modern computers, pointer lookups are relatively costly, especially when they lead to pages which are not yet in the processor's cache.

We therefore decided to implement a "fat skip list". A fat skip list is a skip list whose nodes contain sorted tables of elements. The size of each node is chosen according to the expected target microprocessor architecture during compile time. Our implementation is written in C.

When using a skip list, the time complexity of adding an element to L, A_{L0}, is $O(\log(N))$ and the expected time complexity of searching for the first matching element, A_{L1}, is $O(\log(N))$ in both cases with very high probability. The time complexity of finding the next match, A_{L2}, is $O(1)$.

Our algorithm would perform poorly if the value of r were high (i.e., p_r would be high), which would mean that for each starting sample in each pattern, there would be many matches within the time series. For finite, semi-random time, series and a small enough p_r, we can assume that the number of matching samples for each sample $y(i)$, $p_r \cdot N/2$ will be small. For a small enough r, this could be considered as a constant, C_{p_r}.

With these assumptions, the time complexity of our algorithm becomes $O(N \cdot \log(N) + N \cdot (\log(N) + C_{p_r} + m))$.

Unfortunately, if the signal is normalized prior to calculating sample entropy. The number of matching samples in this case can no longer be considered as small and the time complexity of our algorithm becomes $O(N \cdot \log(N) + N \cdot (\log(N) + N \cdot p_r + m))$. We expect our implementation to be slower than the algorithms using K-D trees and range counting.

3. RESULTS

We tested our algorithm on sections of two time series with various parameter settings. The first was the test data provided with the sampEn program. The second was pink noise.

We used pink noise as an approximation of a physiological signal [5]. We noted the running times of our algorithm and of the more widely used implementation with respect to the length of the signal subsection. Both our algorithm and the most widely used implementation were compiled using gcc 4.8.1 with the -O3 flag and run on a desktop computer running Ubuntu with Linux version 3.2 on an Intel(R) Core(TM) i5-3570K CPU @ 3.40GHz with 8G RAM.

3.1 Worst case

In the worst case, we normalized the signal and set the r parameter to 1.0 so that every pattern matched on each sample in the time series. To make the most widely used algorithm perform better, we set m to 2. Even in this case, our algorithm was faster than the most widely used implementation.

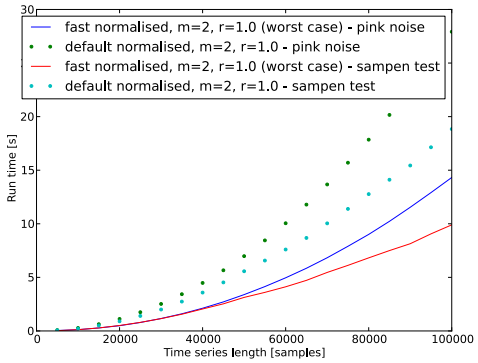

Figure 4: Time needed to calculate the sample entropy of the used signals (worst case). The most widely used implementation is marked as "default".

3.2 Best case

In the best case, the signal was not normalized, r was set to 0.02 and m to 100. In this case, the most widely used algorithm performed relatively poorly.

3.3 Typical case

Typically, sample entropy is calculated for physiological time series. The signals were normalized, m was set to 5 and r was set to 0.2 - the default settings for the most widely used implementation. The improvement in runtime when compared to the more widely used method was noticeable, especially for longer time series lengths at a low value of r.

4. DISCUSSION

The main idea behind the improvement of the algorithm used to calculate the sample entropy of a time series was to only count those pattern matches where the first samples of the patterns are within r of each other, skipping over the rest. To do this efficiently, we only needed to use a data structure which enabled us to search for an element greater than some specified value faster than in $O(N)$ time.

The first implementation of our algorithm used simple skip lists. The penalty incurred by the constant dereferencing of

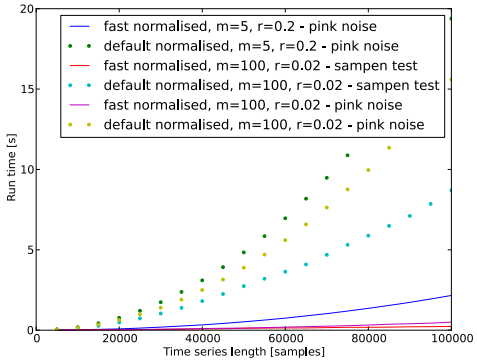

Figure 5: Time needed to calculate the sample entropy of a signal (best, typical cases). The most widely used implementation is marked as "default"

pointers turned out to be so great that our implementation of the algorithm was slower than the more popular sampEn on all but the longest time series. Using fat leaves, in the worst case, the current implementation is faster than the most widely used algorithm.

4.1 Further work

The algorithm presented in this article can easily be adapted for the calculation of sample entropy of consecutive subsections of a time series, using a sliding window.

5. REFERENCES

[1] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, June 13 2000. Circulation Electronic Pages: http://circ.ahajournals.org/cgi/content/full/101/23/e215.

[2] Y. Jiang, D. Mao, and Y. Xu. A fast algorithm for computing sample entropy, 2011.

[3] D. E. Lake, J. S. Richman, M. P. Griffin, and J. R. Moorman. Sample entropy analysis of neonatal heart rate variability. *American Journal of Physiology*, 283(3), 2002.

[4] Y.-H. Pan, Y.-H. Wang, S.-F. Liang, and K.-T. Lee. Fast computation of sample entropy and approximate entropy in biomedicine. *Computer Methods and Programs in Biomedicine*, 104:382–396, 2011.

[5] P. Szendro, G. Vincz, and A. Szasz. Pink-noise behaviour of biosystems. *European Biophysics Journal*, 30:227–231, July 2001.