# Biomedical Signal Processing
# Bubble Entropy

Riccardo Conforto Galli

# Entropy

# Sample Entropy

$$SampEn = -ln\frac{A}{B}$$

$$A = \text{n vectors where } d[X_{m+1}(i), X_{m+1}(j)] < r$$

$$B = \text{n vectors where } d[X_m(i), X_m(j)] < r$$

# Embedding

$$X = X_1, X_2, \ldots, X_{N_m}$$

$$X_i = (x_i, x_{i+1}, \ldots, x_{i+m})$$

- Starting from a time serie

  $x_1, x_2, \ldots, x_N$

  and the size of the embedding space m.

- $N_m = N-m+1$

# Timeserie of indices

$$J = J_1, J_2, \ldots, J_{N_m}$$

Where each $J_i$ is composed in the following way:

- each $X_i$ is sorted
- $J_i$ is composed by the positions which the elements of $X_i$ had before sorting

# Permutation Entropy

$$peEn = -\sum_{i=1}^{n} p(J_i)log(p(J_i))$$

where $p(J_i)$is the probability function of each pattern which appears in $J_i$

# Conditional Permutation Entropy

$$cPE(m) = PE(m+1) - PE(m)$$

# Rényi Permutation Entropy

$$RpEn_\alpha = \frac{1}{1-\alpha} log\left(\sum_{i=1}^{n} p_i^\alpha\right)$$

$$H_2(X) = RpEn_2 = -log\left(\sum_{i=1}^{n} p_i^2\right)$$

# Conditional Rényi Permutation Entropy

$$cRpeN = \frac{H_2^{m+1} - H_2^{m}}{log(m+1)}$$

$$\frac{1}{log(m+1)} = log[(m+1)!] - log(m!)$$

# Bubble Entropy

$$bEn = \frac{(H_{swaps}^{m+1} - H_{swaps}^{m})}{log(\frac{m+1}{m-1})}$$

$$B_i = \text{number of swaps to sort } i$$

$$H_{swaps}^{m} = RpEn(B)$$

# Implementation

# Conditional Rényi Permutation Entropy

```
###### RENYI PERMUTATION ENTROPY
def _rpEn(J, m=0):
    """return an unnormalized rpEn (of order 2) if m = 0, a normalized rpEn otherwise"""
    p = _probabilities(J)
    if m == 0:
        return -log(sum([p[j_i] ** 2 for j_i in p]))
    else:
        return -(log(sum([p[j_i] ** 2 for j_i in p])) / log(m))


def rpEn(timeserie, m):
    """return a normalized rpEn of order 2"""
    return _rpEn(_compute_toi(_embed(timeserie, m)), m)


def cRpEn(timeserie, m):
    return (rpEn(timeserie, m + 1) - rpEn(timeserie, m)) / log(m + 1)
```

# Bubble Entropy

```
def _fast_sort(to_remove, to_add, prev_sorted):
    pos = prev_sorted.index(to_remove)
    prev_sorted = prev_sorted[:pos] + prev_sorted[pos + 1 :]
    for index, element in enumerate(prev_sorted):
        if element > to_add:
            prev_sorted.insert(index, to_add)
            return len(prev_sorted) - 1 - index - pos, prev_sorted
    return -pos, prev_sorted + [to_add]

def _bbEn(X):
    to_remove = X[0][0]
    i, prev_sorted = _bubble_sort(X[0])
    J = [i]
    sorted = [prev_sorted]
    for x_i in X[1:]:
        r, prev_sorted = _fast_sort(to_remove, x_i[-1], prev_sorted)
        sorted.append(prev_sorted)
        to_remove = x_i[0]
        J.append(r + J[-1])
    return _rpEn(J)
```

# Bubble Entropy

```
def bbEn(timeserie, m):
    """return bubble entropy of timeserie X using embedding dimension m"""
    return (_bbEn(_embed(timeserie, m + 1)) - _bbEn(_embed(timeserie, m))) / log(
        (m + 1) / (m - 1)
    )
```

# Experiments

# Data used in the experiments

## nsr2db

- 54 long term ECG recordings of subjects in normal sinus rhythm

## chf2db

- 29 long-term ECG recordings of subjects with congestive heart failure.

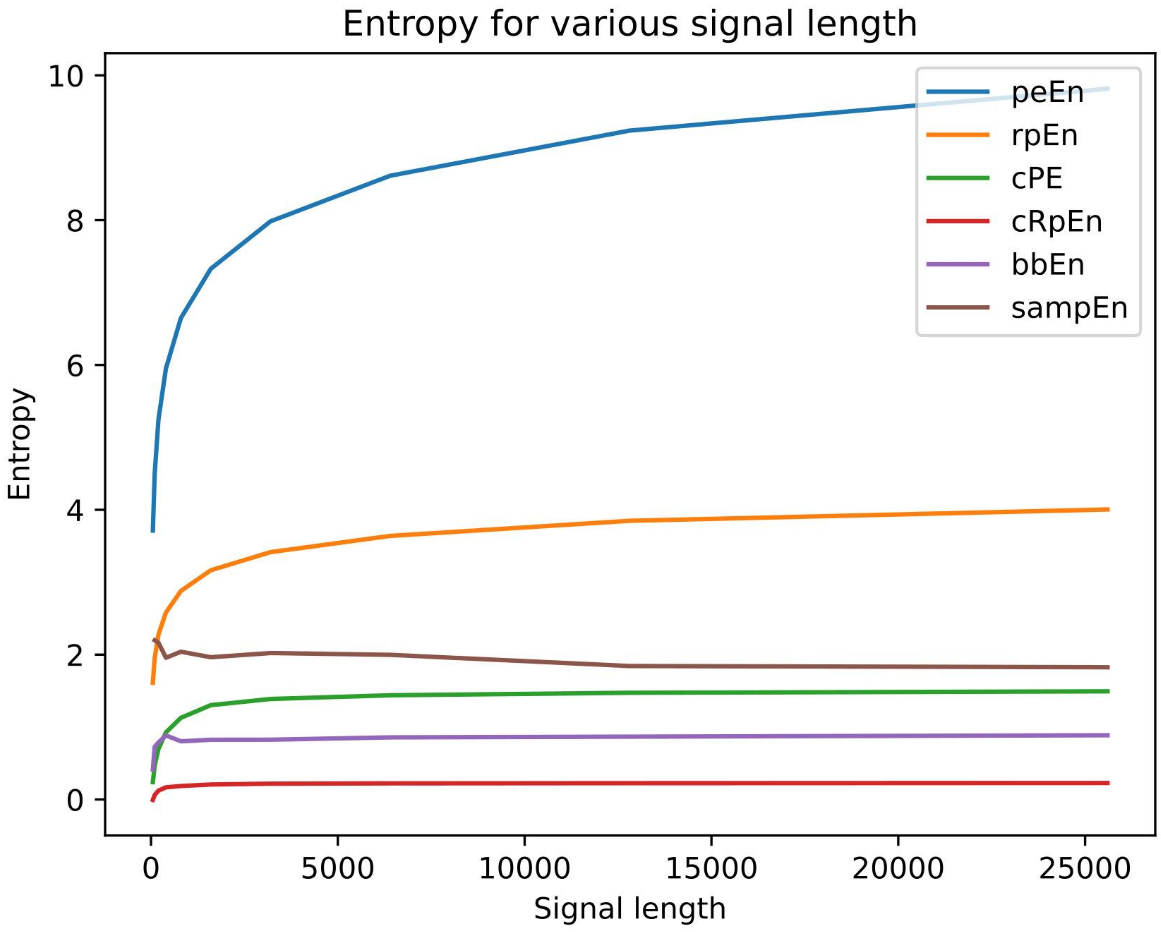
# Stability test with various signal lengths

First with all the algorithms:

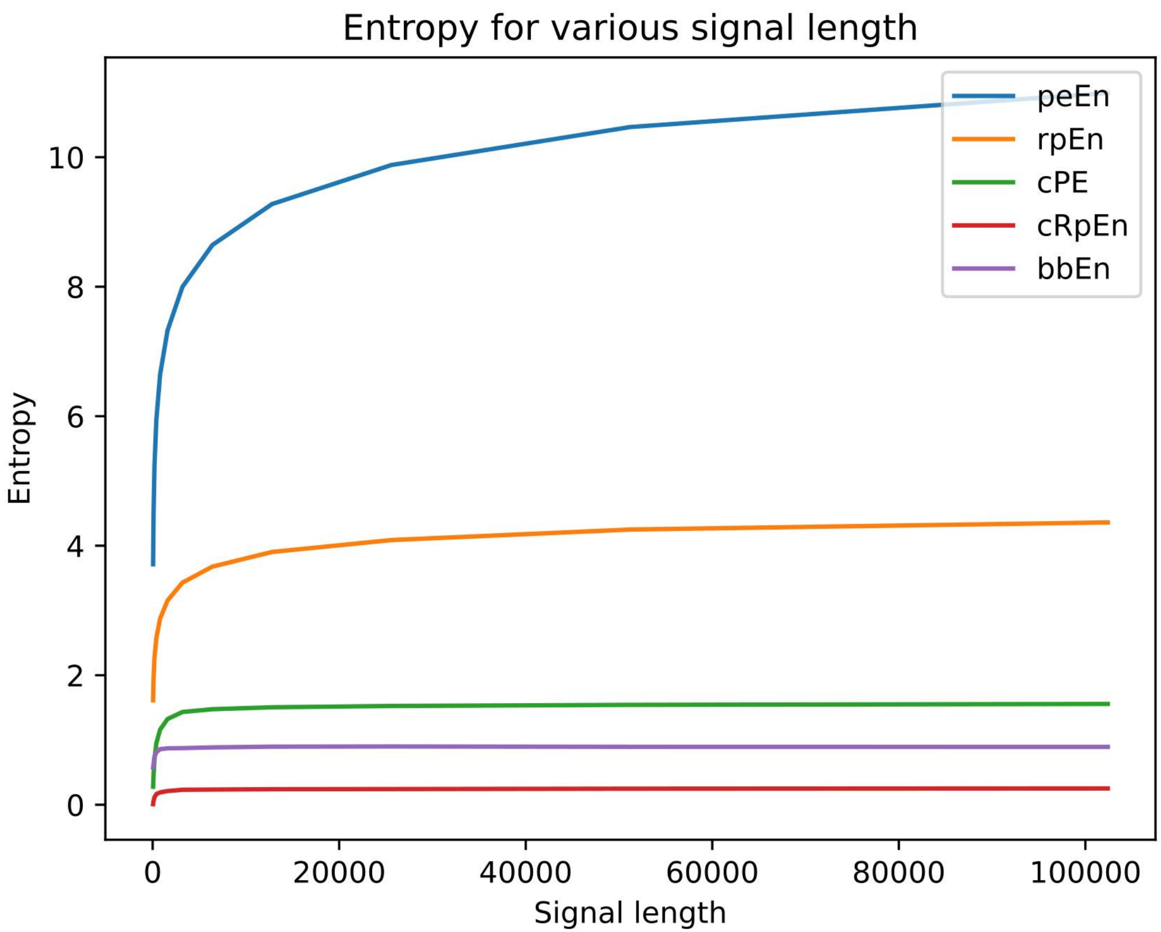
- 5 signals averaged
- N = 50 and doubled until 12800

Then without sampEn:

- 20 signals averaged
- N = 50 and doubled until end of signal
- embedding size depend on the algorithm



Entropy for various signal length
peEn
rpEn
cPE
cRpEn
bbEn
sampEn
Entropy
Signal length
0
2
4
6
8
10
0
5000
10000
15000
20000
25000

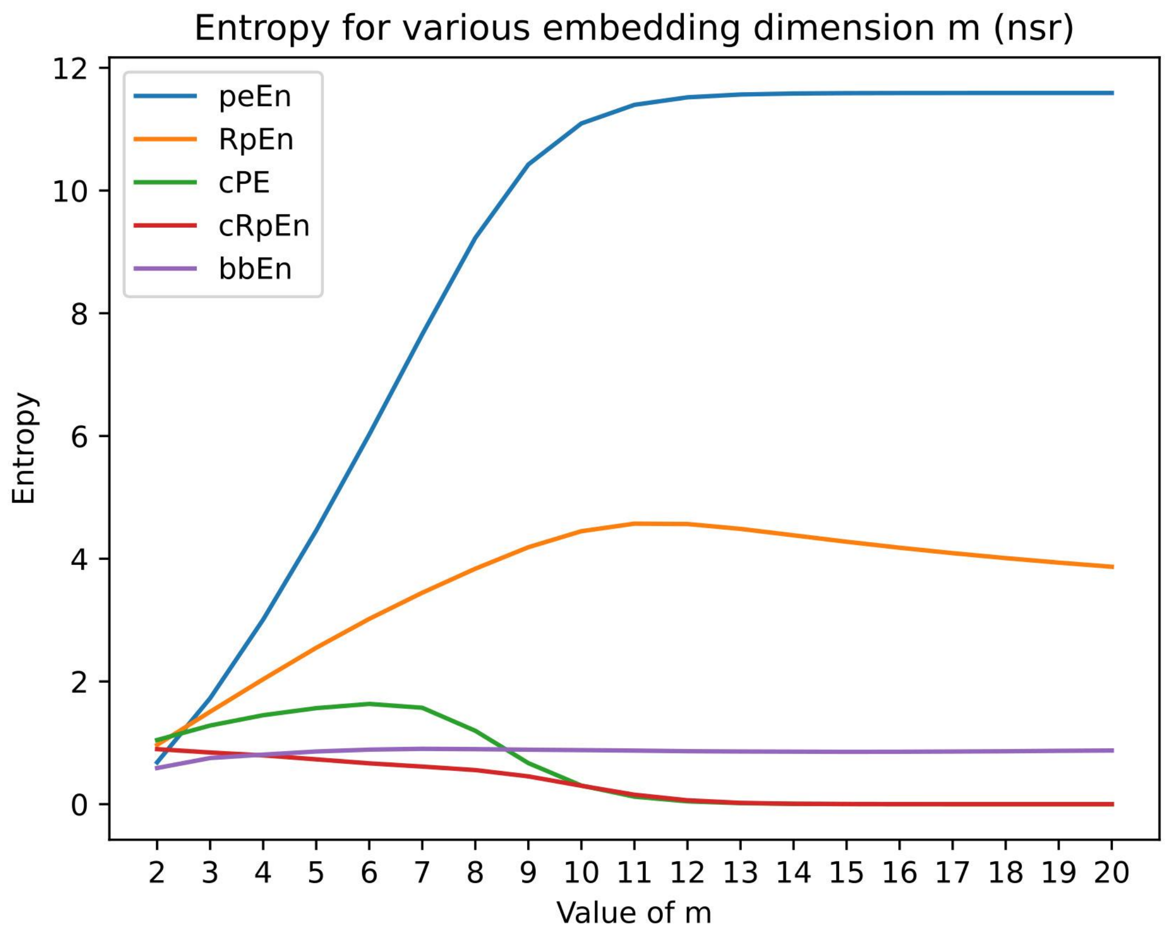


Entropy for various signal length
peEn
rpEn
cPE
cRpEn
bbEn
Entropy
Signal length
0
2
4
6
8
10
0
20000
40000
60000
80000
100000

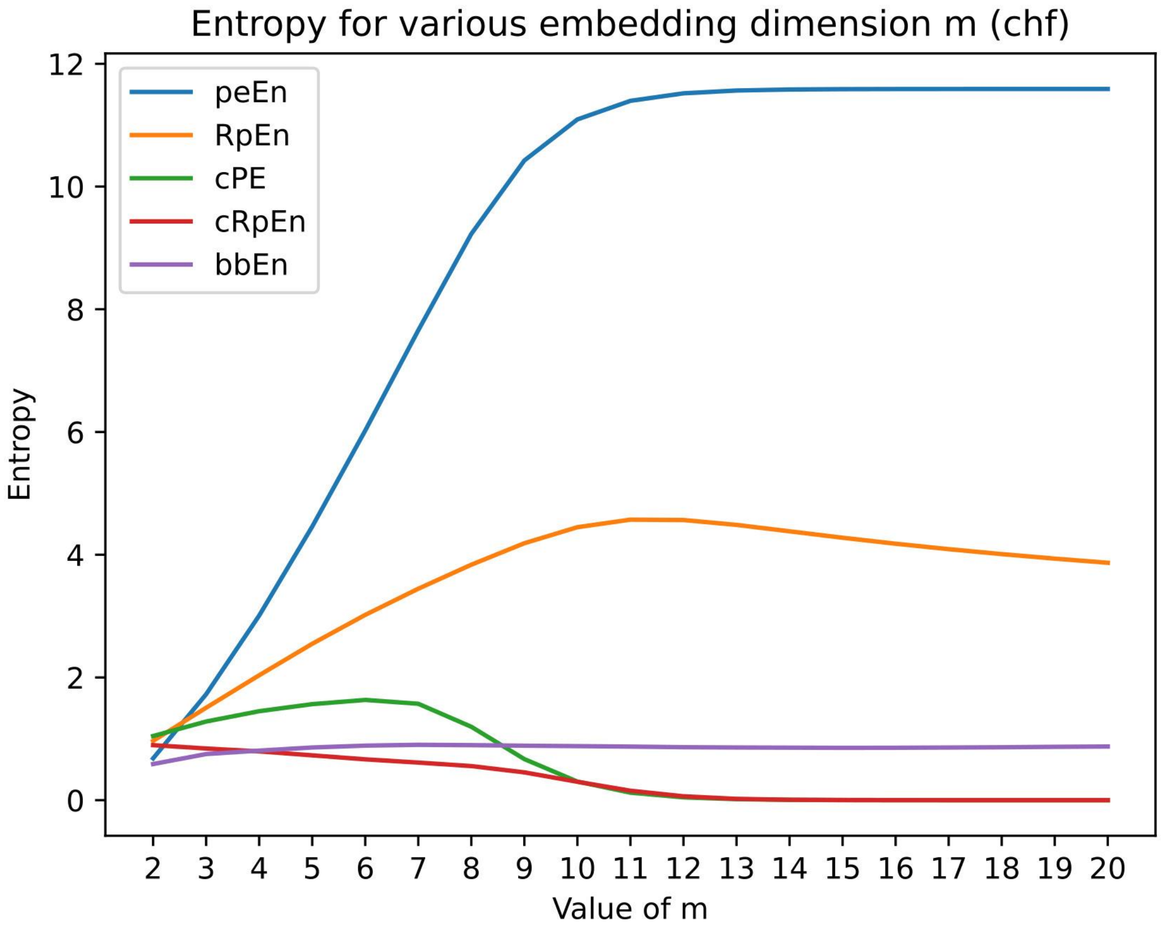
# Stability test with various embedding dimensions

Tested on all algorithms except for sampEn:

- 20 signals averaged
- m = range(2,21)
- Executed for each dataset



Entropy for various embedding dimension m (nsr)
peEn
RpEn
cPE
cRpEn
bbEn
Entropy
Value of m
0
2
4
6
8
10
12
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20



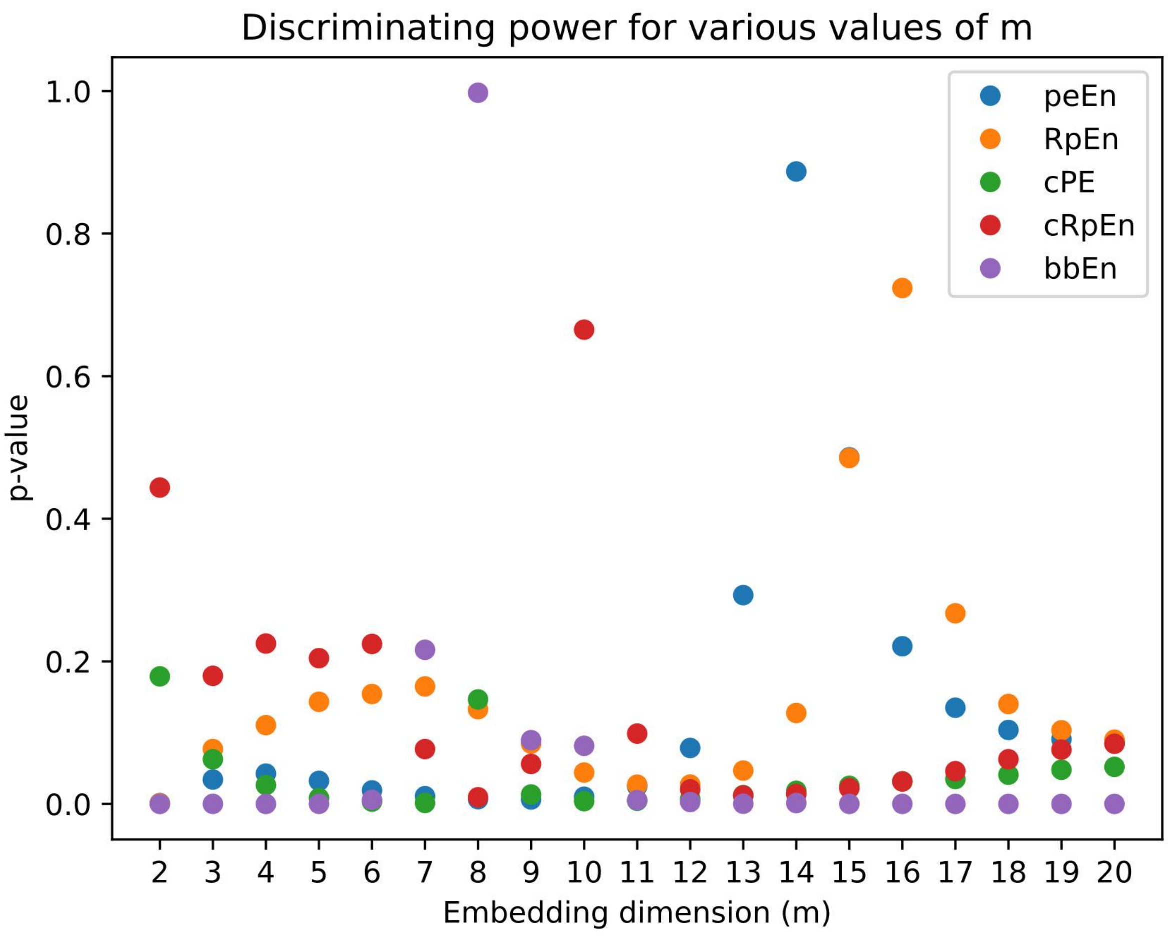
Entropy for various embedding dimension m (chf)
peEn
RpEn
cPE
cRpEn
bbEn
Entropy
Value of m
0
2
4
6
8
10
12
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

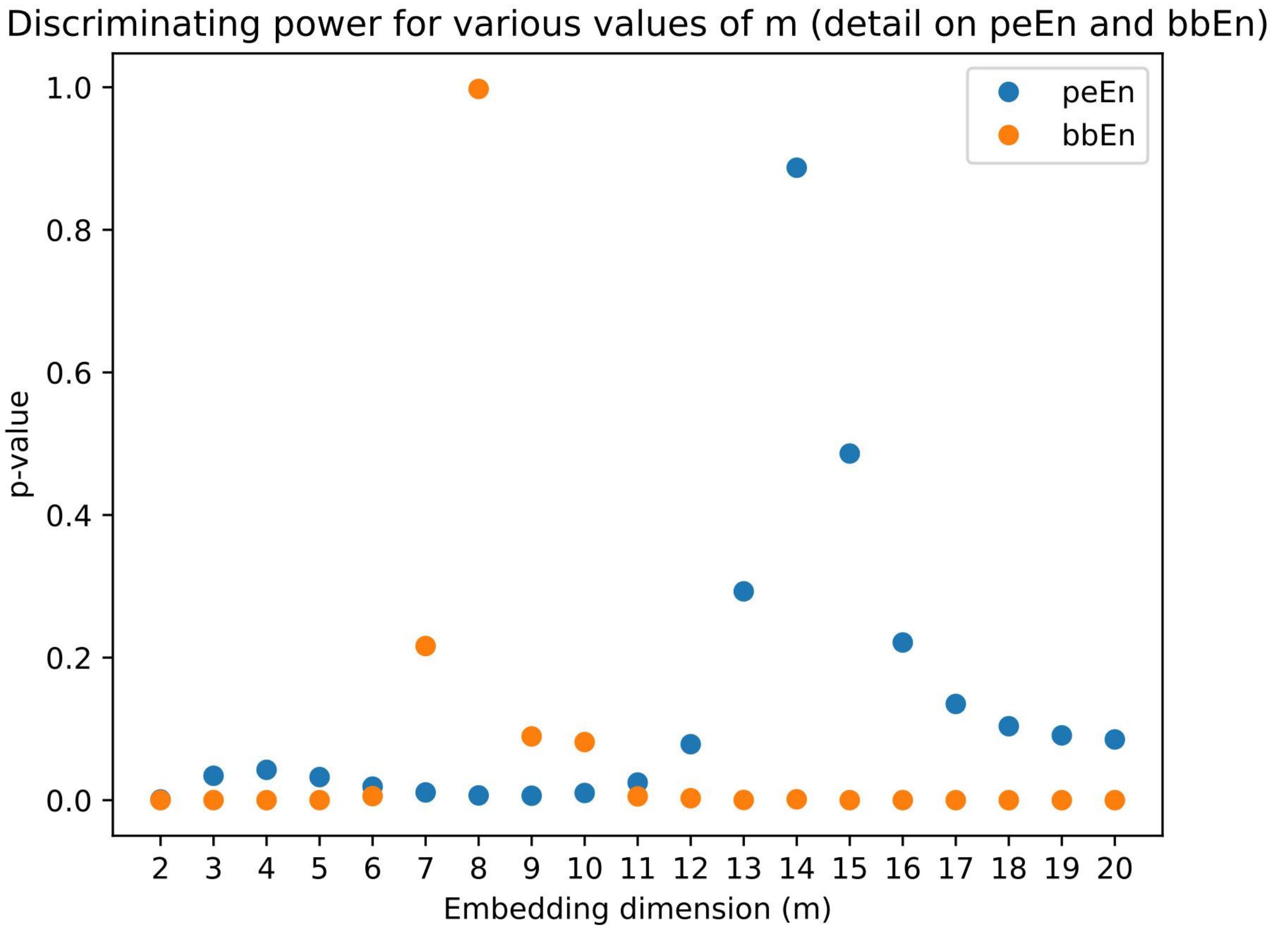
# Discriminating power with various embedding dimensions

Tested on all algorithms except for sampEn:

- 20 signals averaged
- m = range(2,21)
- Executed for each dataset

The resulting entropies are used to compute the p-value, using entropies from different datasets as different populations.


Discriminating power for various values of m
p-value
Embedding dimension (m)
1.0
0.8
0.6
0.4
0.2
0.0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
peEn
RpEn
cPE
cRpEn
bbEn


Discriminating power for various values of m (detail on peEn and bbEn)
peEn
bbEn
p-value
1.0
0.8
0.6
0.4
0.2
0.0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Embedding dimension (m)

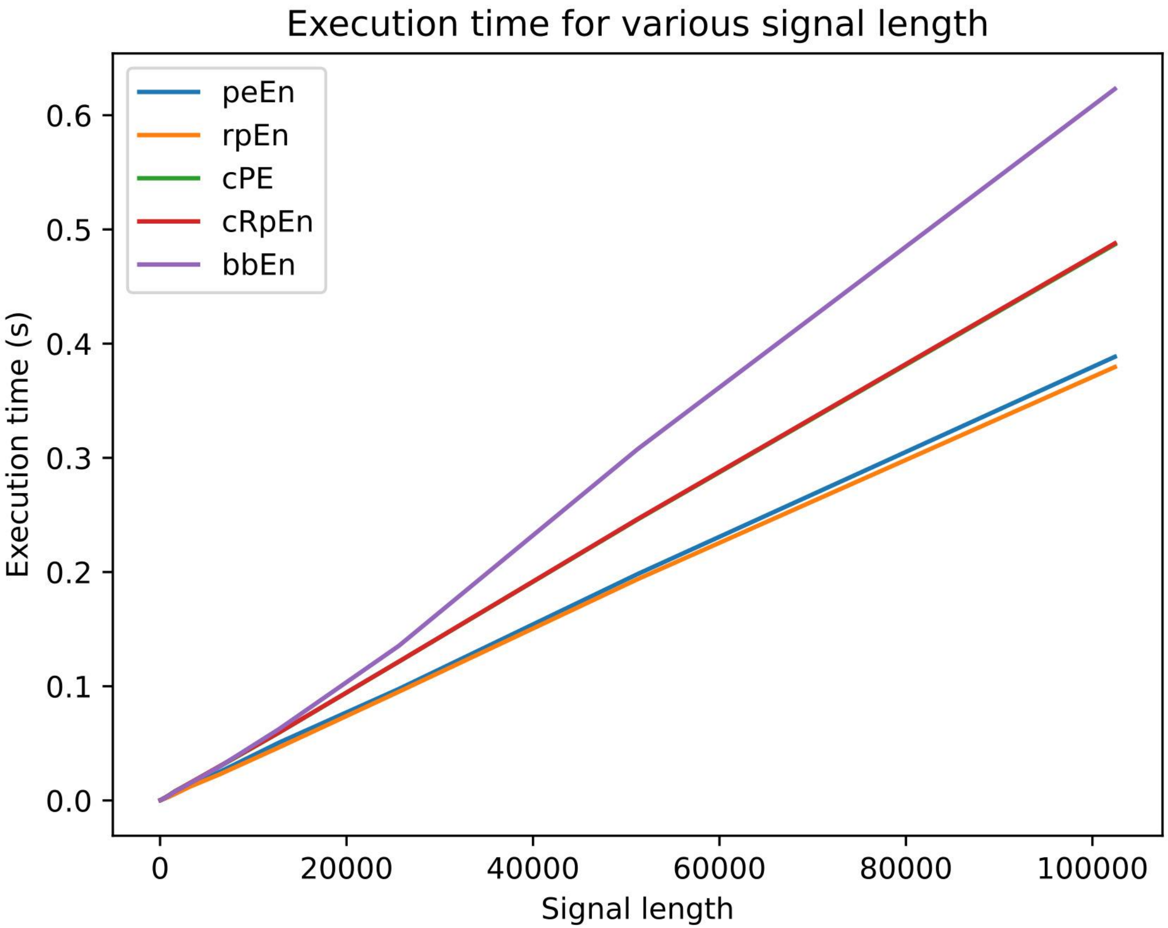
# Execution time for various signal lengths

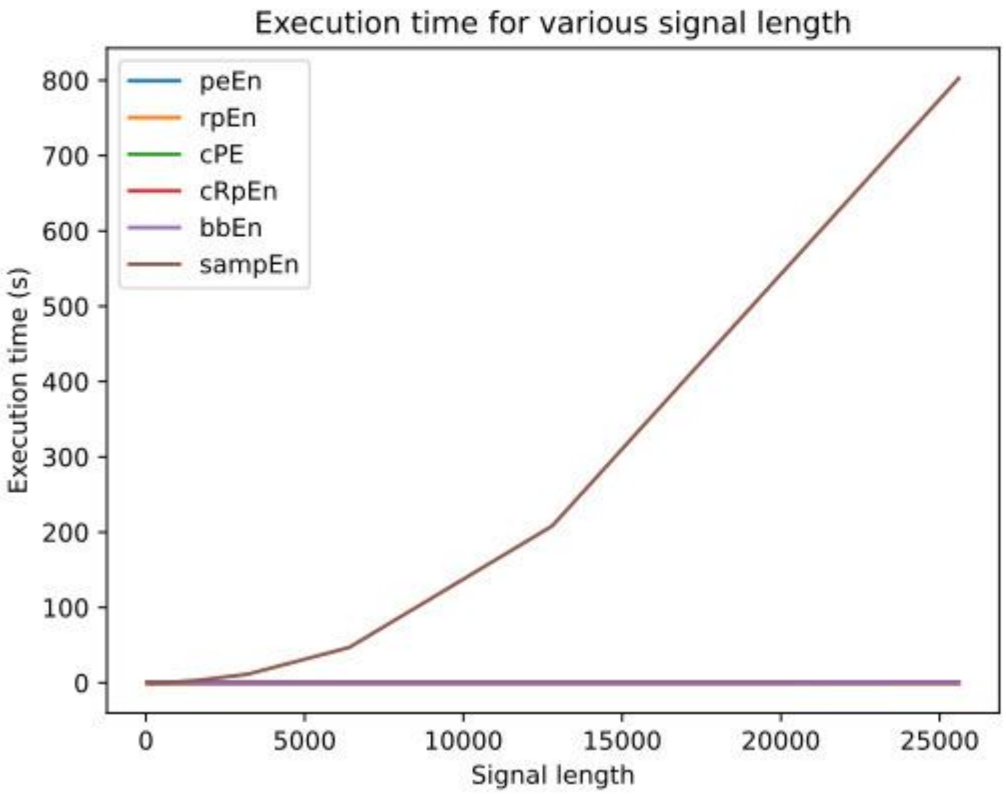
First with all the algorithms:

- 5 signals averaged
- N = 50 and doubled until 12800

Then without sampEn:

- 20 signals averaged
- N = 50 and doubled until end of signal
- embedding size depend on the algorithm


Execution time for various signal length
peEn
rpEn
cPE
cRpEn
bbEn
Execution time (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
20000
40000
60000
80000
100000
Signal length


Execution time for various signal length
peEn
rpEn
cPE
cRpEn
bbEn
sampEn
Execution time (s)
800
700
600
500
400
300
200
100
0
0
5000
10000
15000
20000
25000
Signal length

# Conclusions

1. No need to define r
2. Limited impact of the parameter m
3. Stable behavior
4. Computational complexity: $O(mN)$

# Thanks