Hand in problem 1 in Information Theory (EITN45)

Spring 2020

Hand in Problem 1

In this problem you should implement two functions, the information measures entropy, $H(X)$, and mutual information, $I(X;Y)$. The idea is that by doing the implementation you will have to understand the derivation of the functions. The functions are also good to have during the rest of the course. The implementation should be done using either MATLAB or Python. Since we use scripts to check the functionality it is important that you follow the instructions of how to specify the functions.

Entropy

The first function to be implemented is the entropy function. The input to the function can be a vector of distributions (of the same size). In MATLAB, which is column vector oriented, the distributions are given as column vectors. That is, the input is a matrix where each column represents a probability distribution. In Python, which is more row vector oriented, the distributions are the rows of the input vector. In both cases, if only one value is given for a distribution (i.e. a row vector for MATLAB or a column vector for Python) it should be interpreted as the binary entropy function.

Mutual information

The mutual information should derive the mutual information, $I(X;Y)$. The input for this function is a matrix forming the joint probability distribution $P(X,Y)$.

Hand in details

To hand in the solution you upload the files at Canvas. For MATLAB you should upload two files, named Entropy.m and MutualInformation.m, and for Python one file, named HandIn1.py. For further specifications of these files, see below. Your files will be tested according to the specifications, and you will receive a reply with the result pass or fail.

File specifications

MATLAB

You should upload two files, Entropy.m and MutualInformation.m. It is important that you follow the function header below, that is, the files should start like this.

Entropy.m

```
function H=Entropy(P)
% The Entropy function H(X)
%
% P column vector: the vector is the probability distribution.
% P matrix: Each column vector is a probability distribution
% P scalar: The binary entropy function of [P; 1-P]
% P row vector: Each position gives binary entropy function
<Your code>
```

That is, the input is a matrix P, where the size determines the interpretation[1]. The basic idea is that each column in P corresponds to a probability distribution. If the column contains only one value, the script should derive the binary entropy function for this value. In this way the entropy for several distributions can be derived in a single call.

You can test your function with e.g.

```
>> Entropy([0.3 0.4 0.8; 0.1 0.3 0; 0.3 0.2 0.2; 0.3 0.1 0])
ans =
    1.8955    1.8464    0.7219
>> Entropy(1/8.*ones(8,1))
ans =
     3
>> plot(Entropy([0:0.1:1]));
```

where the last command should plot the binary entropy function. Compare your plot with e.g. the slides from lecture one. Note that you need to take care of the case $0 \log 0 = 0$ separately. (Details on error message handling is optional.)

MutualInformation.m

```
function I=MutualInformation(P)
% The mutual information I(X;Y)
%
% P=P(X,Y) is the joint probability of X and Y.
<Your code>
```

For the mutual information the input matrix P is the joint probability distribution, $p(x, y)$. The MATLAB function `reshape` might come in handy, but it can be solved in other ways also.

Test your function with distributions from the course, e.g.

[1]The MATLAB command size can be used to find the size of a matrix. The binary logarithm, $\log_2$ is called log2 in MATLAB.

```
>> MutualInformation([0 3/4; 1/8 1/8])
ans =
    0.2936
>> MutualInformation([1/12 1/6 1/3; 1/4 0 1/6])
ans =
    0.2503
```

The second example tests the case when P is not square, which should also work.

Python

If you choose to implement in Python you should write it as a class InfoTheory containing the two functions Entropy(P) and MutualInformation(P). You can of course also add other functions that you feel can be useful for the derivation. The file should have the following structure.

handIn1.py

```
import numpy as np
class InfoTheory:
    def Entropy(self,P):
        # Input P:
        #   Matrix (2-dim array): Each row is a probability
        #       distribution, calculate its entropy,
        #   Row vector (1Xm matrix): The row is a probability
        #       distribution, calculate its entropy,
        #   Column vector (nX1 matrix): Derive the binary entropy
        #       function for each entry,
        #   Single value (1X1 matrix): Derive the binary entropy
        #       function
        # Output:
        #   array with entropies
        <Your code, derive the entropy in H>
        return H

    def MutualInformation(self,P):
        # Derive the mutual information I(X;Y)
        #   Input P: P(X,Y)
        #   Output: I(X;Y)
        <Your code, derive the mutual information in I>
        return I

if __name__=='__main__':
    IT = InfoTheory()
    <Tests of your code when running from prompt>
```

By importing numpy you get access to a lot of mathematical functionality, e.g. vector and matrix handling, that is very useful in the implementation. It also gives the logarithm base 2 by np.log2(P).

In the part at the end you can write tests while doing your implementation. For example the following

```
    ### init
    IT = InfoTheory()
    ### 1st test
    P1 = np.transpose(np.array([np.arange(0.0,1.1,0.25)]))# row
   vector
    H1 = IT.Entropy(P1)
    print('H1 =',H1)
    ### 2nd test
    P2 = np.array([[0.3, 0.1, 0.3, 0.3],
                    [0.4, 0.3, 0.2, 0.1],
                    [0.8, 0.0, 0.2, 0.0]])
    H2 = IT.Entropy(P2)
    print('H2 =',H2)
    ### 3rd test
    P3 = np.array([[0, 3/4],[1/8, 1/8]])
    I3 = IT.MutualInformation(P3)
    print('I3 =',I3)
    ### 4th test
    P4 = np.array([[1/12, 1/6, 1/3],
                   [1/4,  0,   1/6]])
    I4 = IT.MutualInformation(P4)
    print('I4 =',I4)
```

This should give an output like the following.

```
H1 = [-0.         0.81127812  1.         0.81127812 -0.        ]
H2 = [1.89546184 1.84643934 0.72192809]
I3 = [0.29356444]
I4 = [0.2502948]
```

Note, in some of the tests there are entries that becomes $0 \log 0$ for the entropy derivation, that has to be dealt with by the function.