

# Information Theory (UAI/500)



## Lecture Notebook

Lecturer: Ivo Bukovsky

## Information, Entropy

- Mutual Information

## Data Dependency

- Multidimensional, multivariate data dependency ( $\mathbf{y} = \mathbf{f}(\mathbf{x})$ ,  $\mathbf{x}$ ...always vector)
- Mutual Information (Section 8.1 in [1])

## Asside Topics

- Probability and Entropy Connotation to FFT (Fast Fourier Transform in 2-D)
- Design of Experiments [a] (... how to cover n-D space?)
- Multidimensional Probability Estimation ( simple RBF neurons, Parzen-Rozenblat density estimation [b])
- Pattern Feature Extraction - overview of methods [f] + automatic feature extraction by 2-D convolutions

## Notes and Notations

- $H$  ... Entropy
- $I(X; Y)$ ,  $MI(X; Y)$  ... Mutual Information , notice the semicolon ";" as separator
- notice the notation in [1] (section 8.1 and elsewhere):
  - $I(A, B; C, D|E, F)$  stands for  
 $I(A, B; C, D|E, F) = I(AB; CD|EF) = I(A \text{ and } B; C \text{ and } D | E \text{ and } F) = I(X; Y)$  where notationally
    - $X = AB$ , i.e., intersection of A and B
    - $Y = CD|EF$  i.e., intersection of C and D given the intersection E and F.

For example, you can also understand intersection  $AB$  via the table in Fig.2 below, where  $A$ ...a feature vector  $\mathbf{x}$ ,  $B$  ... a label  $y$ , and their intersections are actually defined by the rows of table of training data in Fig. 2.

## ✓ Some "Refreshment"

Let's recall conditional probability via the flying drone example, where the drone can physically fly only inside the area  $A$  and  $B$  and these two areas are overlapping.

The probability that the drone randomly crashes are as follows:

- Marginal probability is probability:  
 $P(A)$  ... probability that the drone crashed in area  $A$   
 $P(B)$  ... probability that the drone crashed in area  $B$
- Joint probability is probability:  
 $P(A, B) = P(B, A) = P(AB) = P(BA)$  ... probability that the drone crashed both in  $A$  and  $B$  , i.e. in the intersection of the two areas (if areas do overlap)
- Conditional probability is probability:  
 $P(A|B)$  ... probability that the drone crashed in area  $A$  while it is given it was flying only in area  $B$  (because the areas overlap)  $P(B|A)$   
... probability that the drone crashed in area  $B$  while it is given it was flying only in area  $A$  (because the areas overlap)

Notice, that for partical values of probability and for probability mass function (of a discrete random variable) a small  $p$  is usually used. Capital  $P$  is usually used to denote probability in general (such as for theoretical formulas etc).



Fig. 1: Flying drone example and its event space to recall the conditional probability concept.

Also, recall here the Bayesian rule as follows

$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A),$$

(1)

which can be demonstrated on the drone example above.

And further, let's recall the general formula for complete probability as follows

$$P(B) = \sum_{i=1}^{i=N_A} P(B|A_i) \cdot P(A_i), \quad (2)$$

which can be demonstrated, e.g., on tossing two dices, where  $A_i$  can be outcome that dice A turns number " $i$ " where  $i = 1, 2, 3, \dots, 6$ , and  $B$  can represent the outcome that dice B turns "6".

The principles in formulas (1) and (2) can be then combined for complete Bayes rule(s).

## Data Dependency

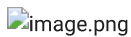
### Multidimensional, multivariate data dependency

Let's generally define a function  $f()$  of multidimensional feature vector as follows

$$y(k) = f(\mathbf{x}(k)), \quad (3)$$

where the input term  $\mathbf{x}(k)$  is the feature vector, the output  $y(k)$  remains scalar here,  $k$  is sample index or discrete time index, and  $\mathbf{X}$  would be then the matrix of all features as in Fig.2.

In this case,  $\mathbf{x}(k)$  may represent a feature vector that can serve as an input to a neural network, and  $y(k)$  may represent the corresponding label (target, or neural output, ...).



**Fig. 2:** See the file "Xy.csv" in subfolder "data" of this Jupyter Notebook. Here,  $\mathbf{X}$  denotes a matrix with row feature vectors, so we look at the 9-dimensional feature vector as at a 9-dimensional random variable  $\mathbf{x}$  (Recall:  $\mathbf{X}$ ,  $\mathbf{x}$ ,  $x$ ...matrix, vector, scalar)

For a multivariate mapping (3) and data example "Xy.csv" in Fig.2, there is a 9-dimensional feature vector  $\mathbf{x}(k)$  and a scalar label  $y(k)$ , then:

- the probability of  $k$ -th feature vector instance in sample-index domain is  $p_{\mathbf{x}}(\mathbf{x}(k)) = p_{\mathbf{x}}(x_1(k), x_2(k), \dots, x_9(k))$ , and for a histogram it is as  $p_{\mathbf{x}}(i_1, i_2, \dots, i_9)$ , so the probability of  $\mathbf{x}$ , is a hypersurface in  $(9 + 1)$ -dimensional space (for  $\mathbf{x}$  being 9-element long vector), for which

$$\sum_{i_1=1}^{n_{bins,1}} \sum_{i_2=1}^{n_{bins,2}} \dots \sum_{i_9=1}^{n_{bins,9}} p_{\mathbf{x}}(i_1, i_2, \dots, i_9) = 1, \quad (4)$$

where the number of bins  $n_{bins,j}$  for features  $x_j$  where  $j = 1, 2, \dots, 9$  can be estimated for each feature  $x_1, x_2, \dots, x_j, \dots, x_9$ , individually.

- the probability of  $k$ -th label in sample-index domain is  $p_y(y(k))$ , and for a histogram it is as  $p_y(i)$  where  $i = 1, 2, \dots, n_{bins,y}$ , so the probability of  $y$  is a curve in 2-D.

Now we see that the feature vector  $\mathbf{x}$  and the scalar label  $y$  **do not have the same probability space, so we can not use  $D_{KL}$**  to evaluate the similarity of their distributions, i.e., we can not evaluate the relationships (3) via distribution comparison of  $\mathbf{x}$  and  $y$  by  $D_{KL}$ .

(Obviously, we can not use Pearson's (linear) correlation coefficient  $r$  either, because  $r$  is defined on for scalar variables of the same length ("same sample domain").)

### ✓ Some remarks: What we can do to find relationship of multidimensional data data?

- Basically, we may try to (batch) train some multivariate regressor, such as linear or polynomial neuron or some other type of shallow neural architecture, preferably linear in parameters, and test its performance, that can give us the insight how well are the features correlated to the labels and if our configuration of training data is ok.
- Obviously, this can be suitable for problems **where deep networks and deep learning can not be done** or does not make sense, e.g., if we do not have enough of training data, or if we need to apply machine learning tasks where computation can not be done with sufficiently powerful HW.
- Notice, the more powerful neural architecture you use, the better you fit the training data, however, it does not necessarily mean, that such trained neural architecture grasped the dependency in data correctly, i.e., **be aware of overfitting (overtraining) vs. good generalization of the neural networks**.
- Notice, that we may use some feature selection method (e.g., BORUTA [ $\alpha$ ]) that can help us to identify the best features, but then we still might wish to double check the feature design, e.g., to validate whether using only features e.g.,  $x_6, x_8$  is better than combination of other two features to predict label  $y$ .
- We may use **Mutual Information** that is suitable for evaluation of nonlinear multidimensional relationships (3) on different probability spaces.

## Mutual Information (MI)

**Mutual Information (MI)** is suitable for evaluating the strength of nonlinear multidimensional (multivariate) relationships of data on different probability spaces.

The input (features) are vectors and the output (label, target) can be either scalar, such as in (3), Fig. 2, or vectors too, such as a neural network with multiple output neurons.

Recall mapping (3) is as follows

$$y(k) = f(\mathbf{x}(k)),$$

and the mapping would be for a fully multivariate problem

$$\mathbf{y}(k) = \mathbf{f}(\mathbf{x}(k)), \quad (5)$$

where  $f(\cdot)$  would yield a vector function  $\mathbf{f}(\cdot)$  as follows

$$\mathbf{y}(k) =$$

$$\begin{bmatrix} y_1(k) \\ y_2(k) \\ \vdots \\ y_{n_y}(k) \end{bmatrix}$$

$$\mathbf{f}(k) =$$

$$\begin{bmatrix} f_1(\mathbf{x}(k)) \\ f_2(\mathbf{x}(k)) \\ \vdots \\ f_{n_y}(\mathbf{x}(k)) \end{bmatrix}$$

$$\mathbf{x}(k) =$$

$$\begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_{n_x}(k) \end{bmatrix}$$

$$\mathbf{x}(k)$$

$$(6)$$

where, for example, we may generally think of  $\mathbf{f}(\cdot)$  as of a neural network with  $n_y$  output neurons.

Further for clarity, let's continue with scalar  $Y$ , i.e.  $y$  as in (3), i.e., for the example of data relationship as in Fig.2.

Recall that probabilities we may calculate for data as for (3), Fig.2, are:

- marginal probabilities  $p(\mathbf{x}), p(y)$ ,
- joint probabilities  $p(\mathbf{x}, y) = p(y, \mathbf{x})$ , and
- conditional probabilities  $p(y|\mathbf{x}), p(\mathbf{x}|y)$ .
  
- information marginal content  $h(X) = I(X), h(Y) = I(Y)$ ,
- information joint content  $h(X, Y) = I(X, Y)$  (notice the comma "," separator for joint events, recall that
  - $P(X, Y) = P(XY) = P(X \text{ and } Y)$  and that  $P(XY) = P(YX)$
- information conditional content  $h(X|Y) = I(X|Y), h(Y|X) = I(Y|X)$  and
  
- marginal entropies  $H(X), H(Y)$ ,
- joint entropy  $H(X, Y)$ ,
- conditional entropies  $H(X|Y), H(Y|X)$  and
- **Mutual Information**  $I(X; Y)$  (notice the semicolon ";" as separator for MI!, "H" is not used for "MI")

and once recalling the formula for information content  $h(x) = I(x) = \log\left(\frac{1}{p(x)}\right) = -\log(p(x))$  and for the (marginal) entropy as its average via probability mass function as  $H(x) = \sum_{\forall x} p(x)h(x)$ , the extensions to joint and conditional entropies should be obvious (find them in [1]).

**Important:** For MI, study section 8.1 in [1] and related parts, see also Fig. 8.1!

Notice, the caption of Fig. 8.1 in [1] on page 140 should read as "Figure 8.1. The relationship between joint *entropy*, marginal entropy, conditional entropy and mutual *information*."

The brief recall on derivation of MI can start with the fundamental Bayes rule for two random variables as follows

$$P(Y, X) = P(Y|X) \cdot P(X) = P(X|Y) \cdot P(Y),$$

(7)

where by applying a minus sign and logarithm to (7), i.e.,  $-\log(\textcolor{blue}{(7)})$ , we arrive to the **chain rule of information content**  $h$  as follows

$$\begin{aligned} -\log(P(Y, X)) &= -\log(P(Y|X)) - \log(P(X)) = -\log(P(X|Y)) - \log(P(Y)) = \\ &= \log\left(\frac{1}{P(Y, X)}\right) = \log\left(\frac{1}{P(Y|X)}\right) + \log\left(\frac{1}{P(X)}\right) = \log\left(\frac{1}{P(X|Y)}\right) + \log\left(\frac{1}{P(Y)}\right) = \\ &= h(Y, X) = h(Y|X) + h(X) = h(X|Y) + h(Y) \end{aligned}$$

(8)

where  $h(\cdot)$  stand for the Shannon information content.

The chain rule of  $h(\cdot)$  in (8) leads us to the **chain rule of entropy** as follows

$$H(Y, X) = H(Y|X) + H(X) = H(X|Y) + H(Y),$$

(9)

and if we subtract both conditional entropies  $H(X|Y)$  and  $H(Y|X)$  from every part of (9), we obtain theoretical formula for the Mutual Information as follows

$$\begin{aligned} I(Y; X) = I(X; Y) &= H(X, Y) - H(X|Y) - H(Y|X) = \\ &= H(X) - H(X|Y) = H(Y) - H(Y|X), \end{aligned}$$

(10)

i.e., Mutual Information is symetric  $I(X; Y) = I(Y; X)$  and non-negative  $I(X; Y) \geq 0$ , see Fig.8.1 in [1].

See (8.3) on p. 139 in [1] for illegality of three-term MI.

## Practical Computation of Mutual Information

- With some (well recognized) library of course, e.g.  $[\beta]$ ?

### MI in Probability Space

Notice, MI can be calculated when  $X, Y$  also have different probability spaces.

First, one-dimensional,  $n_x$ -dimensional (if random variable  $X$  is a vector  $\mathbf{x}$  of length  $n_x$ , e.g.,  $n_x = 9$  in Fig.2), and  $(n_x + 1)$ -dimensional probability mass functions are calculated from measured data (in the sample space), either via multidimensional histograms, or more often by clustering, or by some kernel technique estimation [b] (some rough estimations can be done via a simple RBF network too ... it can be a project for this class)

Then, the formula for computing the Mutual Information in the probability space is as follows

$$I(X; Y) = I(Y; X) = \sum_{\forall \mathbf{x} \in X} \sum_{\forall y \in Y} p_{\mathbf{x}, y}(\mathbf{x}, y) \cdot \log\left(\frac{p_{\mathbf{x}, y}(\mathbf{x}, y)}{p_{\mathbf{x}}(\mathbf{x}) \cdot p_y(y)}\right),$$

(11)

where, either multi-dimensional histograms and probabilities have to be calculated so that

$$\sum_{i_1=1}^{n_{bins,1}} \sum_{i_2=1}^{n_{bins,2}} \dots \sum_{i_n=1}^{n_{bins,n}} p_{\mathbf{x}}(i_1, i_2, \dots, i_n) = 1,$$

(12)

$$\sum_{i_1=1}^{n_{bins,1}} \sum_{i_2=1}^{n_{bins,2}} \dots \sum_{i_n=1}^{n_{bins,n}} \sum_{j=1}^{n_{bins,y}} p_{\mathbf{x}, y}(i_1, i_2, \dots, i_n, j) = 1,$$

(13)

### Practical Estimation of Nonlinear Relationship for Large Datasets (MI formula via clustering)

Practically, we may estimate the probability densities ([b] roughly by a RBF layer), and estimate the MI for our data, or more practically using existing libraries, MI can be estimated via clustering techniques that might be faster to calculate than multidimensional histograms, e.g., see [  $\beta$  ].

$$MI(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \frac{N|U_i \cap V_j|}{|U_i||V_j|}$$

where  $|U_i|$  is the number of the samples in cluster  $U_i$  and  $|V_j|$  is the number of the samples in cluster  $V_j$  (see [  $\beta$  ])

## Example L-11

1. Evaluate the strength of **linear** relationship between observed values of variables  $Y$  and  $X$  in data file `data\yx_linear.txt` and in data file `data\yx_nonlinear.txt`. In particular, implement in your own code (via Numpy) the Pearson's correlation coefficient

$$r(x, y) = \frac{1}{N} \sum_{k=1}^N \frac{(x(k) - \bar{x}) \cdot (y(k) - \bar{y})}{\sigma_x \cdot \sigma_y},$$

where  $N$  is the number of samples, and apply it to data in each file. Validate your result with some existing tool, e.g.

`numpy.corrcoef()`. Plot function  $y = y(x)$  for both data sets. Very briefly compare/explain your results in terms of linearity vs.

nonlinearity of data. **[0.2 Points]**

2. Evaluate the strength of **nonlinear** relationships between observed values of scalar variables  $Y$  and  $X$  in data file `data\yx_linear.txt` and in data file `data\yx_nonlinear.txt` Mutual Information:
  - Calculate MI with some existing library that directly returns MI and uses classical approach with probabilities (reference it) **[0.1 Points]**
  - Calculate MI with your own code in probability space as by formula (11) for both datasets, you can use 1-D and n-D histogram functions from relevant modules to get probabilities **[0.3 Points]**
  - Calculate the MI with some clustering-based method (with some existing library, e.g.  $[\beta]$ ) **[0.3 Points]**
3. Based on Fig.2 and for dataset `data\Xy.txt` Calculate Mutual Information  $I(X; Y) = I(\mathbf{x}; y)$  for a feature vector defined as  $\mathbf{x} = [x_6 \ x_8]$  and for the corresponding label  $y$ :
  - using some existing tool for MI **[0.2 Points]**
  - with your own code for MI in probability space as by formula (11), see Hint a) **[0.6 Points]**
  - Calculate the MI with some clustering-based method (with some existing library, e.g.  $[\beta]$ ) **[0.6 Points]**
4. Based on Fig.2 and for dataset `data\Xy.txt`, analyze which two features (which two columns  $[x_i \ x_j]$  are best for predicting label  $y = y(x_i, x_j)$ , i.e., find  $i, j$  for which  $I(y; X) = I(y; \mathbf{x}) = I(y; [x_i, x_j]) \stackrel{!}{=} \max$ :
  - using some existing tool for MI in probabilistic way **[0.6 Points]**
  - with your own code for MI in probability space as by formula (11), see Hint a) **[1.0 Points]**
  - with MI based on clustering and compare the results **[1.0 Points]**

### Hints:

a) you need some tool for 2-D and 3-D histograms or some estimation for 2-D and 3-D probability ( you do not have to code the multidimensional histograms yourself, but you can try ;)

```
1 """1)Evaluate the strength of linear relationship between observed values of variables Y and X in data file data\yx_linear.txt
2 and in data file data\yx_nonlinear.txt.In particular, implement in your own code (via Numpy) the Pearson's correlation coefficient
3 Validate your result with some existing tool, e.g. numpy.corrcoef().Plot function y=y(x) for both data sets.
4 Very briefly compare/explain your results in terms of linearity vs. nonlinearity of data.
5 """
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 data_linear = np.loadtxt('/content/xy_linear.txt', skiprows=1) # Skipping header row
10 data_nonlinear = np.loadtxt('/content/yx_nonlinear.txt', skiprows=1)
11
12 # Separate x and y values for both datasets
13 x_linear, y_linear = data_linear[:, 0], data_linear[:, 1]
14 x_nonlinear, y_nonlinear = data_nonlinear[:, 0], data_nonlinear[:, 1]
15
16 # Calculate Pearson correlation for linear dataset
17 N_linear = len(x_linear)
18 mean_x_linear, mean_y_linear = np.mean(x_linear), np.mean(y_linear)
19 std_x_linear, std_y_linear = np.std(x_linear), np.std(y_linear)
20 pearson_r_linear = np.sum((x_linear - mean_x_linear) * (y_linear - mean_y_linear)) / (N_linear * std_x_linear * std_y_linear)
21
22 # Calculate Pearson correlation for nonlinear dataset
23 N_nonlinear = len(x_nonlinear)
24 mean_x_nonlinear, mean_y_nonlinear = np.mean(x_nonlinear), np.mean(y_nonlinear)
25 std_x_nonlinear, std_y_nonlinear = np.std(x_nonlinear), np.std(y_nonlinear)
26 pearson_r_nonlinear = np.sum((x_nonlinear - mean_x_nonlinear) * (y_nonlinear - mean_y_nonlinear)) / (N_nonlinear * std_x_nonlinear *
27
28 # Validate with numpy's corrcoef
29 pearson_r_linear_np = np.corrcoef(x_linear, y_linear)[0, 1]
30 pearson_r_nonlinear_np = np.corrcoef(x_nonlinear, y_nonlinear)[0, 1]
31
32 print("Manual Pearson's r (Linear):", pearson_r_linear)
33 print("Numpy Pearson's r (Linear):", pearson_r_linear_np)
34 print("Manual Pearson's r (Nonlinear):", pearson_r_nonlinear)
35 print("Numpy Pearson's r (Nonlinear):", pearson_r_nonlinear_np)
36
37 # Plot y vs x for both datasets
38 plt.figure(figsize=(14, 6))
39
40 # Linear dataset plot
41 plt.subplot(1, 2, 1)
```

```

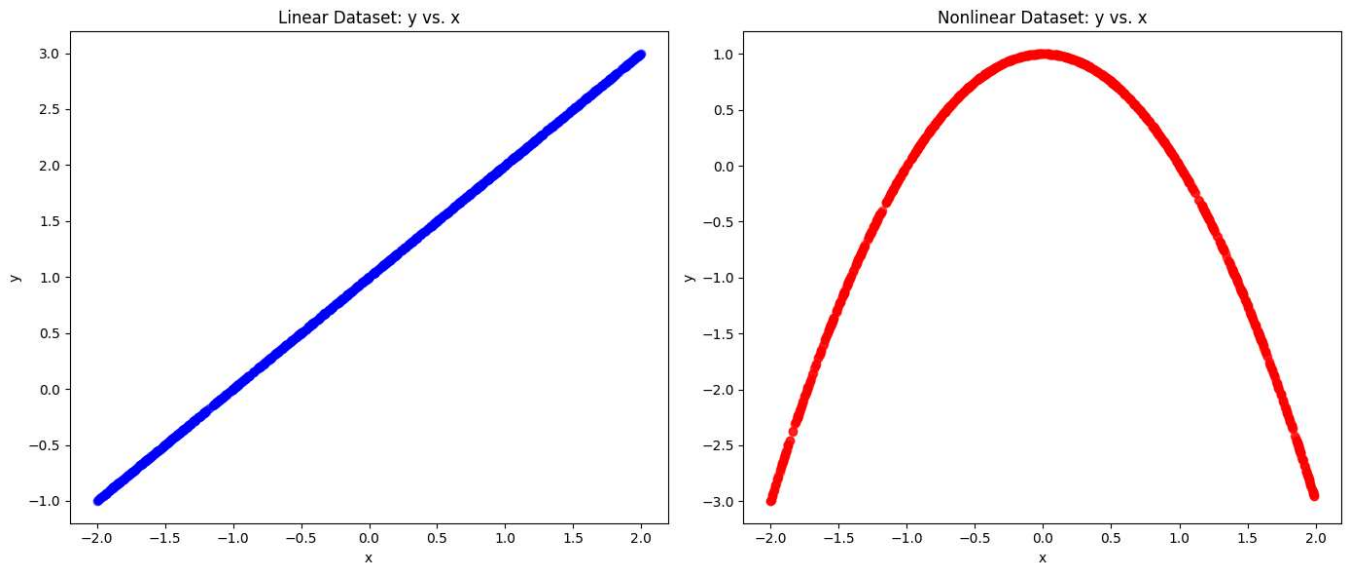
42 plt.scatter(x_linear, y_linear, color='blue', alpha=0.6)
43 plt.title("Linear Dataset: y vs. x")
44 plt.xlabel("x")
45 plt.ylabel("y")
46
47 # Nonlinear dataset plot
48 plt.subplot(1, 2, 2)
49 plt.scatter(x_nonlinear, y_nonlinear, color='red', alpha=0.6)
50 plt.title("Nonlinear Dataset: y vs. x")
51 plt.xlabel("x")
52 plt.ylabel("y")
53
54 plt.tight_layout()
55 plt.show()
56

```

```

Manual Pearson's r (Linear): 1.0000000000000002
Numpy Pearson's r (Linear): 1.0
Manual Pearson's r (Nonlinear): -0.05749190321712425
Numpy Pearson's r (Nonlinear): -0.05749190321712429

```



For the linear dataset, we expect a high absolute value for  $r$ , close to 1 or -1, indicating a strong linear relationship. The plot of  $Y$  against  $X$  for the linear dataset should exhibit a clear linear pattern.

For the nonlinear dataset,  $r$  is likely to be lower, showing that the linear correlation does not fully capture the relationship, as the data may exhibit a nonlinear pattern. The plot for the nonlinear dataset should reveal a more complex, curved pattern, demonstrating why the correlation coefficient is less meaningful in this case

```

1 """ 2) a) Evaluate the strength of nonlinear relationships between observed values of scalar variables Y and X
2 in data file data\yx_linear.txt and in data file data\yx_nonlinear.txt Mutual Information:
3 Calculate MI with some existing library that directly returns MI and uses classical approach with probabilities (reference it)'
4
5 from sklearn.feature_selection import mutual_info_regression
6 import numpy as np
7
8 # Separate x and y values for both datasets
9 x_linear1, y_linear1 = data_linear[:, 0].reshape(-1, 1), data_linear[:, 1]
10 x_nonlinear1, y_nonlinear1 = data_nonlinear[:, 0].reshape(-1, 1), data_nonlinear[:, 1]
11
12 mi_linear = mutual_info_regression(x_linear1, y_linear1)[0]
13 mi_nonlinear = mutual_info_regression(x_nonlinear1, y_nonlinear1)[0]
14
15 print("Mutual Information (Linear dataset):", mi_linear)
16 print("Mutual Information (Nonlinear dataset):", mi_nonlinear)
17
18 """ b) Calculate MI with your own code in probability space as by formula (11) for both datasets,
19 you can use 1-D and n-D histogram functions from relevant modules to get probabilities
20 """
21 import numpy as np
22
23 def mutual_information(x, y, bins=30):
24 # Calculate joint histogram
25 joint_hist, x_edges, y_edges = np.histogram2d(x, y, bins=bins)
26 joint_prob = joint_hist / np.sum(joint_hist) # Normalize to get joint probability distribution
27

```

```

28 # Calculate marginal histograms and probabilities
29 x_marginal = np.histogram(x, bins=x_edges)[0] / len(x) # P(X)
30 y_marginal = np.histogram(y, bins=y_edges)[0] / len(y) # P(Y)
31
32 # Broadcast the marginal distributions to match the joint distribution shape
33 x_marginal = x_marginal.reshape(-1, 1) # reshape for broadcasting
34 y_marginal = y_marginal.reshape(1, -1)
35
36 # Calculate Mutual Information
37 non_zero_joint = joint_prob > 0 # Avoid log(0) by excluding zero probabilities
38 mi = np.sum(joint_prob[non_zero_joint] *
39             np.log(joint_prob[non_zero_joint] / (x_marginal * y_marginal)[non_zero_joint]))
40
41 return mi
42
43 x_linear2, y_linear2 = data_linear[:, 0], data_linear[:, 1]
44 x_nonlinear2, y_nonlinear2 = data_nonlinear[:, 0], data_nonlinear[:, 1]
45
46 mi_linear1 = mutual_information(x_linear2, y_linear2, bins=30)
47 mi_nonlinear1 = mutual_information(x_nonlinear2, y_nonlinear2, bins=30)
48
49 print("\nMutual Information (Linear dataset) with own code:", mi_linear1)
50 print("Mutual Information (Nonlinear dataset) with own code:", mi_nonlinear1)
51
52 #c) Calculate the MI with some clustering-based method (with some existing library, e.g. [ β ] )
53
54 from sklearn.feature_selection import mutual_info_regression
55 import numpy as np
56
57 # Separate x and y values for both datasets
58 x_linear3, y_linear3 = data_linear[:, 0].reshape(-1, 1), data_linear[:, 1]
59 x_nonlinear3, y_nonlinear3 = data_nonlinear[:, 0].reshape(-1, 1), data_nonlinear[:, 1]
60
61 # Calculate Mutual Information using sklearn's mutual_info_regression with k-NN approach
62 mi_linear_knn = mutual_info_regression(x_linear3, y_linear3, n_neighbors=5)[0]
63 mi_nonlinear_knn = mutual_info_regression(x_nonlinear3, y_nonlinear3, n_neighbors=5)[0]
64
65 # Display the results
66 print("\nMutual Information (Linear dataset, k-NN based):", mi_linear_knn)
67 print("Mutual Information (Nonlinear dataset, k-NN based):", mi_nonlinear_knn)
68
69

```



```

Mutual Information (Linear dataset): 5.651137527217012
Mutual Information (Nonlinear dataset): 4.283952623672019

```

```

Mutual Information (Linear dataset) with own code: 3.3856900855250567
Mutual Information (Nonlinear dataset) with own code: 2.305513320628907

```

```

Mutual Information (Linear dataset, k-NN based): 5.2011375272170115
Mutual Information (Nonlinear dataset, k-NN based): 3.7769278250423994

```

```


1 """3) a) Based on Fig.2 and for dataset data\Xy.txt Calculate Mutual Information  $I(X;Y)=I(x;y)$  for a feature vector defined as
2  $x=[x_6 \ x_8]$  and for the corresponding label  $y$  : using some existing tool for MI"""
3
4 import warnings
5 warnings.filterwarnings("ignore")
6 from sklearn.feature_selection import mutual_info_classif
7 import numpy as np
8
9 data = np.loadtxt('/content/XY.txt', skiprows=1)
10
11 # Extract columns for features  $x_6$ ,  $x_8$ , and label  $y$  (last column)
12  $x_6$  = data[:, 5]
13  $x_8$  = data[:, 7]
14  $y$  = data[:, -1]
15
16 # Stack  $x_6$  and  $x_8$  to form a feature matrix
17  $X$  = np.column_stack(( $x_6$ ,  $x_8$ ))
18
19  $mi\_value$  = mutual_info_classif( $X$ ,  $y$ , discrete_features=True)[0]
20
21 print("Mutual Information  $I(X; Y)$  for  $x=[x_6, x_8]$  and label  $y$ :",  $mi\_value$ )
22
23
24 # b)with your own code for MI in probability space as by formula (11)
25
26 def mutual_information_probability_space(x, y, bins=10):
27     # Create 3D histogram for joint distribution  $P(x_6, x_8, y)$ 
28     hist_3d, edges = np.histogramdd(np.column_stack((x, y)), bins=(bins, bins, bins))
29     joint_prob = hist_3d / np.sum(hist_3d) # Normalize to get joint probabilities
30
31     # Marginal distributions

```

```

32 marginal_x = np.sum(joint_prob, axis=2)
33 marginal_y = np.sum(joint_prob, axis=(0, 1))
34
35 # Initialize MI
36 mi = 0.0
37
38 # Iterate over all non-zero joint probabilities
39 for i in range(joint_prob.shape[0]):
40     for j in range(joint_prob.shape[1]):
41         for k in range(joint_prob.shape[2]):
42             # Joint probability for specific (x6, x8, y) bin
43             p_xy = joint_prob[i, j, k]
44             if p_xy > 0: # Avoid log(0)
45                 # Marginal probabilities
46                 p_x = marginal_x[i, j]
47                 p_y = marginal_y[k]
48                 # Mutual Information increment
49                 mi += p_xy * np.log(p_xy / (p_x * p_y))
50
51     return mi
52
53 x_6 = data[:, 5]
54 x_8 = data[:, 7]
55 y_ = data[:, -1]
56
57 X_ = np.column_stack((x_6, x_8))
58
59 # Calculate Mutual Information
60 mi_value_ = mutual_information_probability_space(X_, y_, bins=10)
61
62 print("\nMutual Information with own code for I(X; Y) for x=[x6, x8] and label y:", mi_value_)
63
64 #c) Calculate the MI with some clustering-based method (with some existing library, e.g. [ β ] )
65
66 # Calculate Mutual Information using clustering-based method (k-NN)
67 mi_value1 = mutual_info_classif(X, y, discrete_features=False, n_neighbors=5)
68
69 # Summing MI for each feature
70 mi_total = np.sum(mi_value1)
71
72 print("\nMutual Information I(X; Y) for x=[x6, x8] and label y (k-NN based):", mi_total)
73

```

 Mutual Information I(X; Y) for x=[x6, x8] and label y: 0.32309897068538707

Mutual Information with own code for I(X; Y) for x=[x6, x8] and label y: 0.05242358248919792

Mutual Information I(X; Y) for x=[x6, x8] and label y (k-NN based): 0.0030414789623121052

```

1 # 4) a)
2 import warnings
3 warnings.filterwarnings("ignore")
4 from sklearn.feature_selection import mutual_info_classif
5 import numpy as np
6 from itertools import combinations
7
8 data = np.loadtxt('/content/XY.txt', skiprows=1)
9
10 # Separate features and label
11 X_all = data[:, :-1]
12 y = data[:, -1]
13
14 # Number of features
15 n_features = X_all.shape[1]
16
17 # Initialize variables to store the best feature pair and max MI
18 best_pair = None
19 max_mi = -np.inf
20
21 # Iterate over all pairs of features
22 for i, j in combinations(range(n_features), 2):
23     # Select the current pair of features
24     X_pair = X_all[:, [i, j]]
25
26     # Calculate MI between the pair of features and the label
27     mi_value = np.sum(mutual_info_classif(X_pair, y, discrete_features=False, n_neighbors=5))
28
29     # Update the best pair if this MI is the highest so far
30     if mi_value > max_mi:
31         max_mi = mi_value
32         best_pair = (i, j)
33

```



```

34 print(f"The best feature pair for predicting y is: x[{best_pair[0]}] and x[{best_pair[1]}]")
35 print(f"Maximum Mutual Information I(y; [x[{best_pair[0]}], x[{best_pair[1]}]]) =", max_mi)
36
37 # c)
38 import numpy as np
39 from itertools import combinations
40 from sklearn.cluster import KMeans
41 from sklearn.metrics import mutual_info_score
42 from sklearn.preprocessing import KBinsDiscretizer
43
44 # Function to compute MI based on clustering
45 def compute_mi_clustering(X, y, n_clusters=10):
46     # Apply KMeans clustering to the features X and the label y
47     kmeans_X = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
48     kmeans_y = KMeans(n_clusters=n_clusters, random_state=0).fit(y.reshape(-1, 1)) # Reshape y to fit
49
50     # Get the cluster assignments for the features and label
51     X_labels = kmeans_X.labels_
52     y_labels = kmeans_y.labels_
53
54     # Compute the MI between the feature clusters and the label clusters
55     mi_value = mutual_info_score(X_labels, y_labels)
56
57     return mi_value
58
59 # Initialize variables to store the best feature pair and max MI
60 best_pair = None
61 max_mi = -np.inf
62
63 # Iterate over all pairs of features
64 for i, j in combinations(range(n_features), 2):
65     # Select the current pair of features
66     X_pair = X_all[:, [i, j]]
67
68     # Calculate MI using clustering between the pair of features and the label
69     mi_value = compute_mi_clustering(X_pair, y, n_clusters=10)
70
71     # Update the best pair if this MI is the highest so far
72     if mi_value > max_mi:
73         max_mi = mi_value
74         best_pair = (i, j)
75
76 # Display the best feature pair and the corresponding MI value
77 print(f"\nThe best feature pair for predicting y using clustering is: x[{best_pair[0]}] and x[{best_pair[1]}]")
78 print(f"Maximum Mutual Information I(y; [x[{best_pair[0]}], x[{best_pair[1]}]]) (Clustering-based) =", max_mi)
79

```



The best feature pair for predicting y is: x[3] and x[6]  
Maximum Mutual Information I(y; [x[3], x[6]]) = 0.6252444849136383

The best feature pair for predicting y using clustering is: x[3] and x[6]  
Maximum Mutual Information I(y; [x[3], x[6]]) (Clustering-based) = 0.19831406656114117

## Points and Assignments (week 6):

If you wish to submit your solutions to collect your optional points, solve the problems in this notebook directly and upload to Moodle within 2 weeks.

In this notebook, you may collect maximum of 1.5 points (though the total sum of points available in this notebook is more).

## References

- 
- [α]  
A. Lee, "Boruta Feature Selection Example in Python," Medium, Jun. 01, 2021. <https://towardsdatascience.com/simple-example-using-boruta-feature-selection-in-python-8b96925d5d7a>
- [β]  
"sklearn.metrics.mutual\_info\_score," scikit-learn. [https://scikit-learn/stable/modules/generated/sklearn.metrics.mutual\\_info\\_score.html](https://scikit-learn/stable/modules/generated/sklearn.metrics.mutual_info_score.html)
- [γ]  
"Adjustment for chance in clustering performance evaluation," scikit-learn. [https://scikit-learn/stable/auto\\_examples/cluster/plot\\_adjusted\\_for\\_chance\\_measures.html](https://scikit-learn/stable/auto_examples/cluster/plot_adjusted_for_chance_measures.html)
- [a]  
"Randomized Designs — pyDOE 0.3.6 documentation." <https://pythonhosted.org/pyDOE/randomized.html>.
- [b]  
"Kernel density estimation via the Parzen-Rosenblatt window method," Dr. Sebastian Raschka, Jun. 19, 2014. [https://sebastianraschka.com/Articles/2014\\_kernel\\_density\\_est.html](https://sebastianraschka.com/Articles/2014_kernel_density_est.html).

[c]

S. Raschka and V. Mirjalili, Python machine learning: machine learning and deep learning with Python, scikit-learn, and TensorFlow 2, Third edition. Birmingham Mumbai: Packt, 2019.

[d]

B. Allen, B. Stacey, and Y. Bar-Yam, "Multiscale Information Theory and the Marginal Utility of Information," Entropy, vol. 19, no. 6, p. 273, Jun. 2017, doi: 10.3390/e19060273.

[e]

J. Brownlee, "Information Gain and Mutual Information for Machine Learning," Machine Learning Mastery, Oct. 15, 2019. <https://machinelearningmastery.com/information-gain-and-mutual-information/>

---

[f]

A. Humeau-Heurtier, "Texture Feature Extraction Methods: A Survey," IEEE Access, vol. 7, pp. 8975–9000, 2019, doi: 10.1109/ACCESS.2018.2890743.

---

[g]

"Statistical functions (scipy.stats) — SciPy v1.7.1 Manual." <https://docs.scipy.org/doc/scipy/reference/stats.html>.

---

### Basic Reading:

[1]

MACKAY, David J. C. Information theory, inference, and learning algorithms. Cambridge: Cambridge University Press, 2003. ISBN 978-0-521-64298-9. ([online for screen viewing, i.e., use as the pdf: https://www.inference.org.uk/itprnn/book.pdf](#)) and [video lectures by David MacKay](#).

### Recommended Reading:

[2] COVER, T. M. and Joy A. THOMAS. Elements of information theory. 2nd ed. Hoboken: Wiley-Interscience, c2006. ISBN 978-0-471-24195-9..

[3] HOST, S. Information and Communication Theory. Hoboken, NJ: Wiley-IEEE Press, 2019. ISBN 978-1119433781..

[4] EL-GAMAL, A. and YOUNG-HAN, K. Network information theory. Primera. Cambridge: Cambridge University Press, 2011. ISBN 978-1-107-00873-1..

