Tor vergata
Master degree in Computer Science
Information Theory and Data Mining
A.A. 2019-2020

Assignment VI

Leonardo Tamiano

January 31, 2020

This document contains the sixth assignment and it's structured as follows: in the first section we briefly describe how to setup the environment to run the code that is sent along side this document; from there on each section deals with a single assignment by first describing some theory and then proceeding to describe the result obtained from the code written for the assignment.

Contents

0 Environment setup

In order to properly execute the developed code one must satisfy the following requirements:

(i) Have **python** with version ≥ 3.7 installed on the system.

(ii) Set the environmental variable PYTHONPATH to point to the folder sent through the e-mail. For example, if we save and unzip our code in the directory /home/leo/Downloads, then the PYTHONPATH should have the following value

```
PYTHONPATH=/home/leo/Downloads/ITDM_1920_Leonardo_Tamiano
```

(iii) The following python libraries have to be installed:
matplotlib, **sklearn**.

1 Assignment I

1.1 Discrete Random Variables, Information and Entropy

This assignment deals with the definition of the entropy function. **Entropy** is a function defined on a pmf (probability-mass-function) and it is mainly used as a *measure of uncertainty*.

A given **pmf**, that is a sequence of probabilities $p_1, p_2, ...p_{N_X}$ that add up to 1 ($\sum p_i = 1$), with $N_x \in \mathbb{N}$, can be used to describe a *random variable*, which can be thought of as a random experiment in which the result is not known deterministically, and all the possible results have an associated probability, that is the probability that the random experiment will have exactly that specific result.

For example the sequence of numbers $[0.2, 0.7, 0.1]$ can be described as the pmf of a random variable that assumes three distinct values and in which the first variable has a 20% probability of occuring, the second value 70% has 10% probability of occuring, and the third value has 10% probability of occuring.

Given a pmf of a random variable X then, we are interested in asking the following: "how much 'uncertainty' is there in X?". That is, how likely am I to predict the next value X will assume? This is where entropy comes in: *the higher the entropy, the higher the uncertainty, and the less likely I'm to predict its next value.* To actually compute its value, the entropy of a r.v. X with pmf $[p_1, p_2, ..., p_{N_X}]$ is defined as

$$H(X) := \sum_{i=1}^{N_X} p_i \cdot \log_2 \frac{1}{p_i}$$

The concept of uncertainty is linked with the concept of *information* in the following way: the more uncertain an event is, in probability terms, and the more information that events gives us, in case it happens. Using this intuition, given an event A, one can define the *self-information* of A, denoted by $I(A)$, in the following way

$$I(A) := \log_2 \frac{1}{p_i}$$

It now becomes clear that the entropy $H(X)$ of a random variable X can be described as follows

$$H(X) = \sum_{i=1}^{N_X} p_i \cdot \log_2 \frac{1}{p_i} = \mathbb{E}[I(X)]$$

which can be interpreted as stating that the entropy gives us the *average information content produced by a given random experiment.*

1.2 Entropy Plots

In the assignment we are asked to compute the graph of the entropy for a generic binary random variable as a function of p_0. The graph is reported below

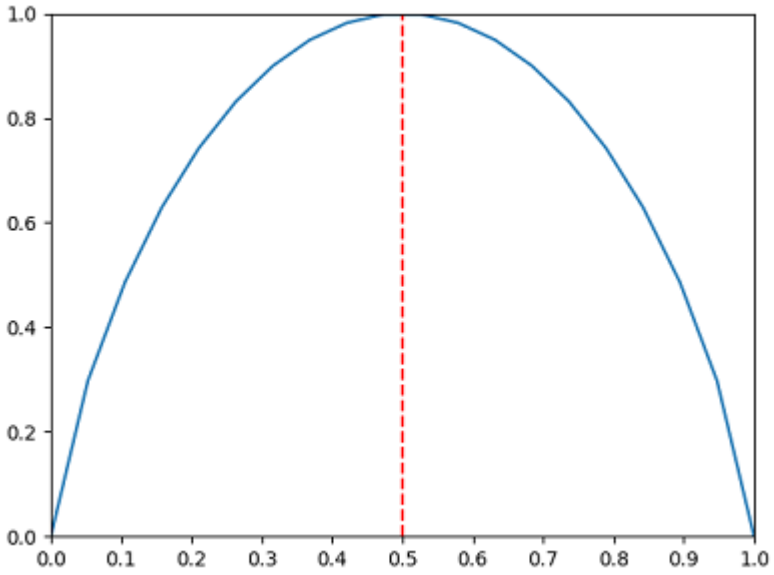

Figure 1: Graph of entropy for generic binary r.v.

Notice that the obtaiend graph makes sense: the entropy is maximized for the pmf $[0.5, 0.5]$, that is, when we have maximum uncertainty (all events have the same probability), while it is minimized for the edge cases $[1, 0]$ and $[0, 1]$, that is, when we have no uncertainty (either always happens the first event, or always happens the second event).

We are then asked to do the same, but for a generic ternary binary random variable as a function of p_0 and p_1. The graph computed is reported below from different perspectives

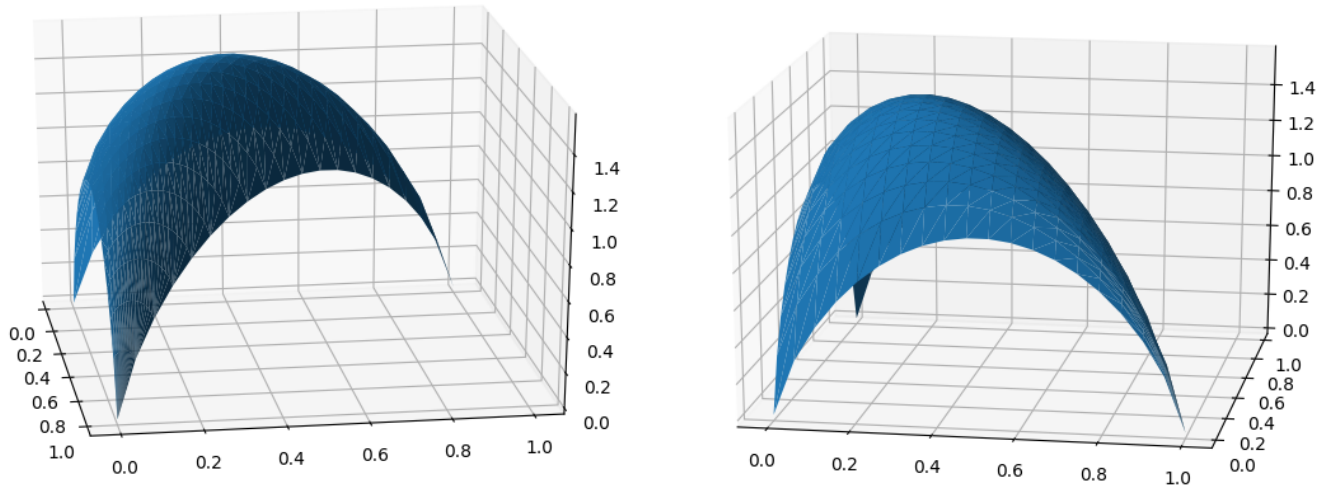

Figure 2: Graph of entropy of ternary pmf

Once again, the obtained result is coherent with that we would have expected: the entropy is maximized for the pmf $[1/3, 1/3, 1/3]$, which is the case of maximum uncertainty, while it is minimized for the three pmf cases $[1, 0, 0]$, $[0, 1, 0]$ e $[0, 0, 1]$, that is for all the cases with minimum uncertainty.

2 Assignment II

2.1 Multiple Discrete Random Variables

Consider two random experiments. As we have discussed in the previous chapter, we can model these random experiments with two random variables, X e Y, each of which is characterized by a specific pmf. Suppose that the pmf of X is the vector $[p_{x1}, p_{x2}, ..., p_{xN_x}]$, while the pmf of Y is the vector $[p_{y1}, p_{y2}, ..., p_{yN_y}]$. We can now ask the following question: are these two random experiments *independent* from eachothers, or do they influence eachothers in some way?

To construct a mathematical model that is able to answer this question, we first have to introduce some basic building blocks regarding single random variables. Consider then only X, with its pmf $[p_{x1}, p_{x2}, ..., p_{xN_x}]$. We define the following quantities

- The *mean of* X is defined as $\mathbb{E}[X] := \sum_{i=0}^{N_x} p_{xi} \cdot x_i$ and it measures a possible center value of X.
- The *variance of* X is defined as $Var[X] := \mathbb{E}[(X - \mathbb{E}[X])^2]$ and it measures how much X is spread with respect to the mean $\mathbb{E}[X]$.

  A low variance means that most likely X will assume a value around the mean $\mathbb{E}[X]$, while high variance means X can assume values even far away from the mean $\mathbb{E}[X]$.

We can now define the concept of *co-variance*, which is defined as

$$Cov(X, Y) := \mathbb{E}[(X - \mathbb{E}[X]) \cdot (Y - \mathbb{E}[Y])]$$

and it can be interpreted as a measure of the *joint variability* of the two random variables. The lower the covariance, and the lower the two random experiments are linked together, i.e., the more independent the two experiments are from eachothers. If $Cov(X, Y) = 0$, then the r.v. are said to be *independent*, and this fact can be denoted by writing $X \perp\!\!\!\perp Y$.

When we have multiple discrete random variables, we can define their *joint-pmf* as follows

$$P_{xy}(x_i, y_j) = P(X = x_i, Y = y_j)$$

The joint-pmf can be used to describe the probability of a joint outcome, made by combining the outcome of the single random experiments. Now, if the two random variables are independent, that is, if $Cov(X, Y) = 0$, then we can compute their joint pmf as follows

$$P_{xy}(x_i, y_j) = P(X = x_i, Y = y_j) = P(X = x_i) \cdot P(Y = y_j) = p_{xi} \cdot p_{yj}$$

2.2 Measures of Information

In this assignemtn we were asked to compute some of the most important measures that can be associated with two discrete random variables and which are used heavily in the context of data mining problems. It was also asked to compute a particular class of measures, which are called *normalized measures* and which are characterized from the fact that they take assume value in the range $[0, 1]$.

Let us consider two r.v. X and Y, and let $P_{XY}(\bullet, \bullet)$ be their joint pmf and $P_X(\bullet)$, $P_Y(\bullet)$ their respective marginal pmfs. The standard measure are reported below:

(i) **Joint entropy**: Denoted by $H(X, Y)$, it is defined as follows

$$H(X,Y) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} P(x_i, y_j) \cdot \log_2 \frac{1}{P_{XY}(x_i, y_j)}$$

The joint entropy $H(X, Y)$ is a measure of the uncertainty associated with the random variables X and Y. It has the following properties:

- $H(X, Y) = H(Y, X)$, that is, it is *symmetric*.
- $\max\{H(X), H(Y)\} \leq H(X, Y) \leq H(X) + H(Y)$.
- $H(X, Y) = H(X) + H(Y) \iff X \perp\!\!\!\perp Y$.

In the code (file assignment_2/assignment_2.py), it is computed in the following way

```
#
# ARGUMENTS:
#
# joint_pmf is a numpy matrix describing the joint probabily mass
# functions of X and Y. In particular
#
#           joint_pmf[i][j] = P(X = x_i, Y = y_j)
#
def joint_entropy(joint_pmf):
    N_x, N_y = joint_pmf.shape
    result = 0

    for i in range(0, N_x):
        for j in range(0, N_y):
            if joint_pmf[i][j] != 0:
                result += joint_pmf[i][j] * math.log(joint_pmf[i][j], 2)
    return -result

# ------------------------------------------------
```

(ii) **Conditional entropy**: Denoted by $H(X|Y)$, it is defined as follows

$$H(X|Y) := \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} P_{XY}(x_i, y_j) \cdot \log_2 \frac{1}{P(x_i|y_i)}$$

The conditional entropy $H(X|Y)$ is used to measure the uncertainty of experiment X, knowing the value produced by experiment Y. It has the following properties:

- $H(X|Y) \neq H(Y|X)$
- $0 \leq H(X|Y) \leq H(X)$
- $H(X|Y) = H(X) \iff X \perp\!\!\!\perp Y$
- $H(X|X) = 0$

In the code (file assignment_2/assignment_2.py), it is computed in the following way

```
# ARGUMENTS:
#
# 1) joint_pmf is a numpy matrix describing the joint probabily mass
#    functions of X and Y. In particular,
#
#            joint_pmf[i][j] = P(X = x_i, Y = y_j)
#
# 2) Y_pmf is a numpy vector describing the probabily mass function of
#    Y. In particular,
#
#            Y_pmf[j] = P(Y = y_j)
#
def conditional_entropy(joint_pmf, Y_pmf):
    N_x, N_y = joint_pmf.shape
    result = 0

    for i in range(0, N_x):
        for j in range(0, N_y):
            # Compute the conditional probability of X given Y using
            # the formula
            #
            # P(X = x_i | Y = Y_i) = P(X = x_i, Y = y_j)/P(Y = y_j)
            #
            cond_prob = joint_pmf[i][j] / Y_pmf[j]
            if cond_prob != 0:
                result += joint_pmf[i][j] * math.log(cond_prob, 2)

    return -result
```

(iii) **Mutual information**: Denoted by $I(X,Y)$, it is defined as follows

$$I(X,Y) := \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} P_{XY}(x_i, y_j) \cdot \log_2 \frac{P_{XY}(x_i, y_j)}{P_X(x_i) \cdot P_Y(y_j)}$$

The mutual information $I(X,Y)$ is used to measure the information that is common both to X and Y. It has the following properties

- $X \perp\!\!\!\perp Y \implies I(X,Y) = 0$.
- $0 \leq I(X,Y) \leq \min\{H(X), H(Y)\}$
- $I(X,X) = H(X)$
- $I(X,Y) = I(Y,X)$

In the code (file assignment_2/assignment_2.py), it is computed in the following way

```
# ARGUMENTS:
#
# 1) joint_pmf is a numpy matrix describing the joint probabily mass
#    functions of X and Y. In particular,
#
#           joint_pmf[i][j] = P(X = x_i, Y = y_j)
#
# 2) X_pmf is a numpy vector describing the probabily mass function of
#    X. In particular,
#
#           X_pmf[i] = P(X = x_i)
#
# 3) Y_pmf is a numpy vector describing the probabily mass function of
#    Y. In particular,
#
#           Y_pmf[j] = P(Y = y_j)
#
def mutual_information(joint_pmf, X_pmf, Y_pmf):
    N_x, N_y = joint_pmf.shape
    result = 0

    for i in range(0, N_x):
        for j in range(0, N_y):
            arg = joint_pmf[i][j] / (X_pmf[i] * Y_pmf[j])
            if arg != 0:
                result += joint_pmf[i][j] * math.log(arg, 2)

    return result
```

Here are the normalized versions of the previously introduced measures.

(i) **Normalized joint entropy**: Denotata con $\mu_{JE}(X,Y)$, e definita come segue

$$\mu_{JE}(X,Y) = 1 - \frac{I(X,Y)}{H(X)+H(Y)} \in \left[\frac{1}{2}, 1\right]$$

(ii) **Normalized conditional entropy**: Denotata con $\mu_{CE}(X,Y)$, e definita come segue

$$\mu_{CE}(X,Y) = \frac{I(X,Y)}{H(X,Y)} \in [0,1]$$

(iii) [1]**Normalized mutual information** Denotata con $\mu_{MI}(X,Y)$, e definita come segue

$$\mu_{MI}(X,Y) := \frac{I(X,Y)}{H(X,Y)} \in [0,1]$$

Finally, here are some other measure that was discussed during lecture.

(i) **Entropy correlation coefficient**: Denoted by $\mu_{CC}(X,Y)$, it is defined as follows

$$\mu_{CC}(X,Y) := \sqrt{\frac{2 \cdot I(X,Y)}{H(X)+H(Y)}} \in [0,1]$$

(ii) **Symmetric uncertainty**: Denoted by $\mu_{SU}(X,Y)$, it is a normalized measure of uncertainty, and its defined as follows

$$\mu_{SU}(X,Y) := \frac{2 \cdot I(X,Y)}{H(X)+H(Y)} \in [0,1]$$

(iii) **Variation of information**: Denoted by $V(X,Y)$, it is defined as follows

$$\begin{aligned} V(X,Y) &:= H(X|Y) + H(Y|X) \in [0,1] \\ &= H(X,Y) - I(X,Y) \\ &= H(X) + H(Y) - 2 \cdot I(X,Y) \end{aligned}$$

Notice that $V(X,Y)$ has the following properties

- $V(X,Y) = V(Y,X)$
- $V(X,X) = 0$
- $V(X,Y) \leq V(X,Z) + V(Z,Y)$

This means that $V(X,Y)$ can be used to measure distances between pmfs, or, in other words, that $V(X,Y)$ is a measure of *dissimilarity*.

[1]There other other types of normalized mutual information, here we have described only one.

(iv) **Normalized Variation of Information**: Denoted by $\eta_{VI}(X, Y)$, it represents the normalized version of $V(X, Y)$ and it is defined as follows

$$\eta_{VI}(X, Y) = \frac{V(X, Y)}{H(X, Y)} \in [0, 1]$$

3 Assignment III

In this assignment we were asked to compare the difference between the entropy of a discrete random variable given its pmf and the entropy of the *estimated* pmf obtained from a set of sample generated through the same pmf used to compute the first entropy. We were then asked the same thing, but instead of doing it with pmf (discrete case), we were asked to do it with a pdf (continuous case).

3.1 Differential entropy

So far we have only define the concept of entropy for discrete random variables. We will now extend it to the case of a continuous random variable, that is a random variable that has an associated *probability density function* $f(x)$ such that $f(x) > 0$ for all x in the range of the r.v. and $\int_{-\infty}^{\infty} f(x) = 1$. The association between a continuous random variable X and its pdf $f(x)$ is expressed through the *cumulative distribution function*, which, for a continuous random variable, is defined as

$$F_x(t) := P(X \leq t) = \int_{-\infty}^{t} f(x)dx$$

Let X be a continuous random variable with pdf $f(x)$. The *differential entropy* of X is defined as

$$h(x) : = \int_a^b f_x(x) \cdot \log_2 \frac{1}{f_x(x)} dx = -\int_a^b f_x(x) \cdot \log_2 f_x(x) dx$$

where $S = [a, b] \subset \mathbb{R}$ is the *support set*[2].

At this point we have to make the following important distinction between entropy and differential entropy: while entropy, which is defined for discrete random variables, is an *absolute* measure of information, that is a measure that has a meaning in and of itself, the differential entropy, which is defined for continuous random variables, is only a *relative* measure of information whose meaning has to be interpreted with other data. This big difference comes from the fact that the value of differential entropy depends on the possible values of the continuous random variable, and not only on their respective probabilities; the value of entropy, instead, only depends on the pmf, and not on the particular possible values.

To make this idea more concrete, consider X to be a Gaussian variable with mean μ and variance σ^2, $X \sim \mathcal{N}(\mu, \sigma^2)$ its differential entropy can be

[2]The support set for a continuous r.v. X is a subset obtaiend by taking all elements in the domain of $f(x)$ who get mapped to positive values.

computed as follows

$$\begin{aligned}
h(x) &= -\int_{-\infty}^{\infty} f_x(x) \cdot \log_2\left(f_x(x)\right) dx \\
&= -\int_{-\infty}^{\infty} f_x(x) \cdot \log_2\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\cdot\left(\frac{x-\mu}{\sigma}\right)^2}\right) dx \\
&= -\log_2\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \cdot \int_{-\infty}^{\infty} f(x)dx + \int_{-\infty}^{\infty} \frac{(x-\mu)^2}{2\sigma^2} f(x)dx \\
&= \frac{1}{2} \cdot \log_2\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2} \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx \\
&= \frac{1}{2} \cdot \log_2\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2} \cdot \sigma^2 \\
&= \frac{1}{2} \cdot \left(\log_2\left(2\pi\sigma^2\right) + 1\right)
\end{aligned}$$

3.2 Estimation using samples

Consider to study a random experiment, represented by the discrete r.v. X with with f_x as the pmf of X. We are interested in knowing what is the pmf of X, that is, we want to know the probability associated with each possible outcome of X. How can we do this?

The basic idea is to use a *sample*, that is a set of outcomes generated by repeating the experiment X in order to compute an *estimation* of f_x. Notice that by estimating we are not guaranteed to compute the correct probabilities; we are only guaranteed that the values we compute will be within a certain *measurable* error away from the correct values.

One of the simplest and most effective way to estimate a pmf using a set of sample is to use compute the frequency of occurence of each different term in the sample. This is exactly what we did in the assignment: to estimate the probability of occurence of a certain value v we simply computed

$$\hat{f}_f(v) = \frac{\text{occurences of } v \text{ in the sample}}{\text{sample size}}$$

The same idea of sampling and estimating can be extended to the continuous case. This time, however, the techniques that compute the estimations are a bit more complicated. There are various techniques that allows for pdf estimations, such as:

- **Histogram**: Based on *discretizing* the real line into a specified number of *bins* and counting how many samples fall inside each bin.

  Formally, we have the following: let h be the bins-width, and let x_0 be the starting position of the first bin, then the *histogram density estimator* is defined as

$$\hat{f}_x(x) := \frac{1}{n \cdot h} \sum_{k=1}^{n} \sum_{i=-\infty}^{\infty} \mathbb{1}(x \in B_i) \cdot \mathbb{1}(s_k \in B_i)$$
$$= \hat{f}_x(x) = \frac{1}{n \cdot h} \sum_{k=1}^{n} I\left(\frac{x - s_k}{h}\right)$$

where $B_i := [x_0 + i \cdot h \ , \ x_0 + (i+1) \cdot h]$ is the interval associated with the i-th bin.

Depending on the parameters x_0 and h we choose, we can have different estimations. In particular it is worth noting that a small bin width suffers from *large variance*, while a large bin width suffers from *large bias*.

- **Kernel Function Method**: Based on using *kernel functions*, that is functions $k(x)$ such that $(i) \int_{-\infty}^{\infty} k(x)dx = 1$ and $(ii) \int_{-\infty}^{\infty} x \cdot k(x)dx = 0$ to obtain a smoother estimation of the pdf.

  Formally, let h be the bandwidth, and let $k(\cdot)$ be a kernel function, then the *kernel density estimator* is defined as

$$\hat{f}_x(x) := \frac{1}{n \cdot h} \sum_{k=1}^{n} k\left(\frac{x - s_k}{h}\right)$$

  Notice that the estimation obtained this way is smoother than the one obtained using the histogram method, since $\hat{f}_x(x)$ is a continuous function. It can be proved that as $n \to \infty$ the kernel density estimator converges to the true density.

  There are different well known kernel functions. For example, the machine learning python library Scikit-learn offers the following kernel function to do pdf estimation

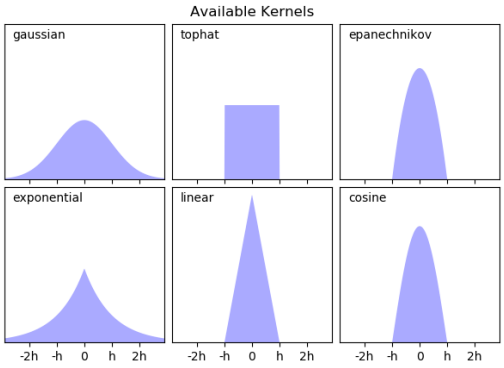

Figure 3: Kernel functions in Scikit-learn

3.3 Tuning pdf estimation using kernel method

In this paragraph we will consider the impact of bandwith on the difference between the differential entropy computed by knowing the mean and the variance of a guassian pdf and the differential entropy computed by estimating the gaussian pdf using the kernel method on a set of samples generated through the same gaussian r.v. (same mean and variance as before).

Let $X_1, ..., X_n$ be an indipendent sample generated from a gaussian model, i.e., $X_i \sim \mathcal{N}(\mu = 33, \sigma^2 = 5^2)$. Let B_{opt} be the "optimal bandwidth", computed as follows

$$B_{opt} = 1.06 \cdot \hat{\sigma} \cdot n^{-\frac{1}{5}}$$

The following graph shows the impact of the bandwidth as stated before. In particular on the x-axis we have the bandwidth, where on the Y-axis we have the difference between the two differential entropies.

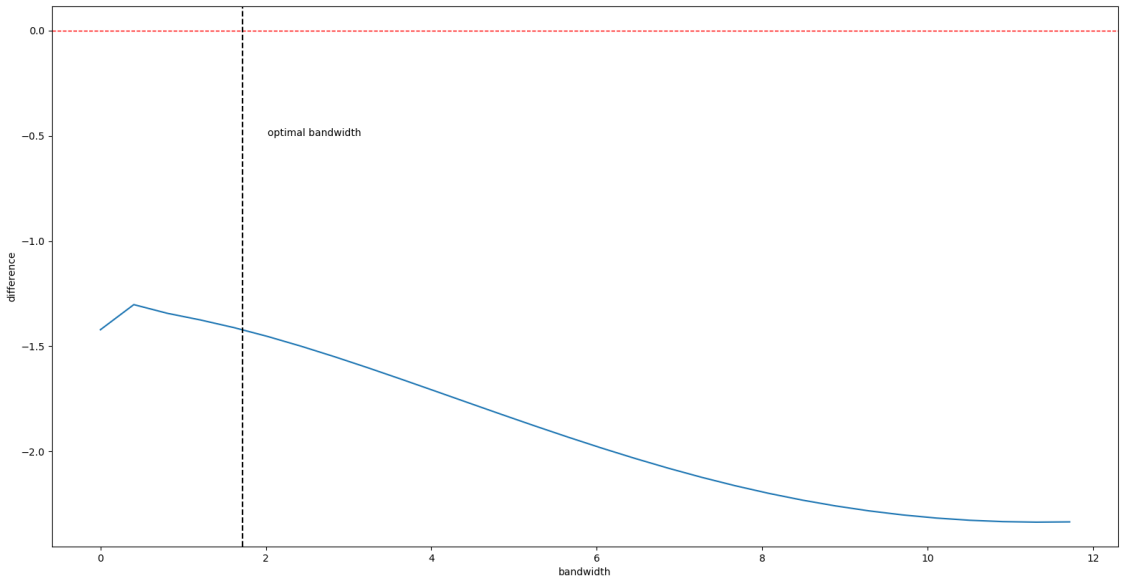

Figure 4: Bandwidth impact on difference 1

As we can see the choosen bandwidth value is not too far from the most optimal one in the range $[0, 2]$. Also, as we move further and further, the difference between the differential entropies of the theoretical pdf and the estimated pdf keeps increasing. The following graph however shows that this trend does not continue monotonically.

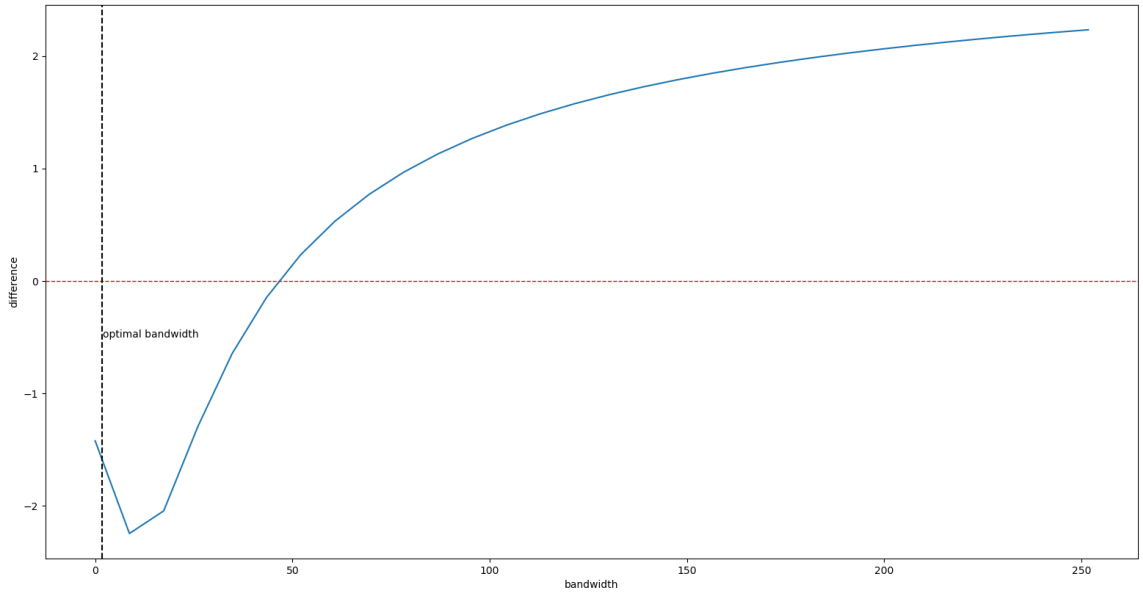

Figure 5: Bandwidth impact on difference 2

4 Assignment IV

4.1 Datasets

Once we have observed a bunch of data, we can structure it to form what are called *datasets*. A dataset can be represented with a data-matrix made up of n rows and d columns. The d columns are called the *features* of our dataset, while the n rows are called the *instances* of our dataset. Features and instances can be described as follows:

- A feature represents a particular characteristic about the objects we want to analyze. Put together, the d features form the characteristics we can analyze about each object in the world we have observed.
- An instance (i.e. a row in the dataset) represent an object in the real world that we have observed. Each object is characterized by having particular values for each feature.

Depending on the nature of the feature, it can represent various kinds of data. In particular we have can have either *numerical data*, which is data that rapresents quantitative measures, like calculating the length of a petal, or *categorial data*, which comes from qualitative measures, like observing the color of a pair of eyes.

Different data can be processed in different ways: for example categorical data can be further subdivided into *ordinal data* and *regular data*. To regular data you can only apply the $=$ operator, while to ordinal data you can also apply the $\leq$ operator to order the various possible values.

One famous dataset, used extensively as a learning tool, is the iris dataset, which consists of 50 samples from each of three species of Iris, which is a type of plant. For each sample the dataset contains the measurements of four different features: the length and width of sepals and petals, in centimeters. Here follows the first 10 rows (instances) of the iris dataset.

```
iris.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

(a) Iris dataset

(b) Iris setosa

4.2 Data mining

Once we have a dataset, we are interested in turning this data into information which can then be used to make plans and decisions. To do this we have to process and fliter the data by removing the parts of it which are of no use to our particular objective and by making the useful parts stand out. This task of processing and filtering the data is called *data mining*.

In a data mining we may, or may not do the following tasks in order to pre-process the data for later processing. The particular tasks done depend on the particular task we want to solve.

(i) **Data transformation**,

Consist of changing the type of certain data present in our dataset, as well as re-organizing the dataset in a different way as to facilitate the later processing.

(ii) **Data normalization**,

Consists of changing the range of values of certain features. For example, in case of numerical data, by normalizing we can have that all features range in the interval $[0, 1] \in \mathbb{R}$.

The impact that this normalization step has depends on the initial ranges of the various features.

(iii) **Management of outlier and missing value**,

Consists of discovering and choosing how to dela with outliers and missing value. An outlier is considered to be an observation with very low probability. To discover outliers it is therefore necessary to define a *threshold* value.

If the task we want to solve is not the discovering of outlier values, like for example in alarm systems, we may want to discard outliers to clean our dataset.

As for missing values, we can deal with them in two different ways: either discard the entire row, or put the sample mean (with respect to the dataset) of the respective feature instead of the missing value.

(iv) **Data discretization**

Consists of turning continuous numerical data (formalized using continuous r.v.) into discrete numerical data (formalized using discrete r.v.) in order to use information theory tools such as entropy, mutual information, and so on.

This step can also be considered as a **noise reduction step** in the preprocessing chain.

(v) **Dimensionality reduction**

Consists of reducing the number of rows and columns of our dataset.

There are various tecniques for doing dimensionality reduction such as SVD (Singular Value Decomposition) and PCA (Principal Component Analysis).

(vi) **Feature selection**

Consists of discarding features which are not useful to the processing phase. The significance of a given feature is computed according to a particular *metric*.

For example, we might use the mutual information to compute the correlation between the various features, and discard a feature if it is highly correlated with some other in the dataset.

4.3 Mutual information in iris dataset

In the assignment we were asked to compute the mutual information between the features of the iris dataset. To do so i discretized the dataset by multiplying each entry by 10, estimated the various pdfs, joint and marginals, and went on to compute the mutual information between each couple of features. The results are reported in the table below.

	Feature 1 (sepal-length)	Feature 2 (sepal-width)	Feature 3 (petal-length)	Feature 4 (petal-width)
Feature 1	4.822	2.089	3.003	2.240
Feature 2		4.822	2.227	1.676
Feature 3			5.034	2.695
Feature 4				4.050

Notice that in the table only the upper diagonal portion of the table is filled, since the mutual information is symmetric, i.e. $I(X, Y) = I(Y, X)$.

5 Assignment V

In this assignment we were asked to implement three different versions of a Bayes classifier in order to classify iris plants using a well known dataset. We were then asked to compute the accuracy

5.1 Classification

Classification is a typical problem in the field of machine learning. It can be describe informally as follows: we have a dataset in which to every row (instance) is assigned a particular value, called the class of the instance, taken from a well-defined set of class-values. The classification problem consists of building an algorithm that is able to assign, to every possible instance, even those outside of our dataset, a particular class label, such that the class label assigned is the one that describes optimally the given instance.

A more formal definition of the problem of classification would be the following: Given a fixed set of classes $\mathcal{C} = \{c_1, c_2, ..., c_J\}$, a dataset, represented by an $n \times d$ matrix D, where n is the number of instances, and d is the number of features of the dataset, and a class column vector C which assigns to every row in the dataset a class value taken from $\mathcal{C}$, we want to construct a function γ that maps instances of features, even those not contained in the original dataset, to classes.

There are various approaches to solve this problem. In particular we have seen the following:

- **Decision trees** A decision tree is a finite, oriented tree, with a root. A decision tree takes in input the instance to classify and uses the values of the intermediate nodes, called the *splitting features*, with certain *spltiting criterions* to traverse the tree and arrive at one of the leafes, which represent class labels. The leaf reached is then the class label assigned to the instance given in input.

  One type of decision tree classifier is named ID3 (Iterative Dichotomiser). This decision tree is built by computing the mutual information between the various features F_j and the class set C. Originally ID3 could only be applied to discrete data, in which the splitting criterion was easily to decide: branch for every distinct value; in the continuous case choosing how to split the dataset to construct the tree may be time consuming.

- **Bayes Classifiers** Based on the well-known Bayes' theorem, which states that, if A and B are events in a given probability space, then

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

  Bayes classifiers will be discussed in more detail in the following section.

5.2 Bayes Classifiers

Let $\bar{x}$ be an instance we want to classify. Which class should we assign to instance $\bar{x}$? We would like to assign to $\bar{x}$ the class that maximizes the probability $P(c_t|\bar{x})$, that is the probability that, given instance $\bar{x}$, the class of $\bar{x}$ is exactly c_t. Formally we have

$$c(\bar{x}) := \underset{c \in \mathcal{C}}{\text{argmax}} \; P(c_t|\bar{x})$$

All Bayes classifiers then make use of the following observation: we know from probability theory that

$$P(c_t|\bar{x}) = \frac{P(\bar{x}|c_t) \cdot P(c_t)}{P(\bar{x})}$$

Consider now how the formula changes as the class value changes: if instead of c_t we put a different class value, like c_l, only the nominator of the right-hand side changes, while the denominator stays the same. This is true for all class values. This means that, if we are interested only in the value c_t which maximizes the formula, we do not have to compute the denomintor, since it has the same values for all class values. This, then, allows us to make the following big leap

$$c(\bar{x}) = \underset{c \in \mathcal{C}}{\text{argmax}} \; P(c_t|\bar{x}) = \underset{c \in \mathcal{C}}{\text{argmax}} \; P(\bar{x}|c_t) \cdot P(c_t)$$

To compute this we then have to estimate, for each class c_t, the probabilities $P(\bar{x}|c_t)$ and $P(c_t)$. The estimation for $P(c_t)$ is simply obtained by counting in the original dataset how many instances are assigned to class c_t and diving the obtained number by the number of instances. In formula we obtain:

$$\hat{P(c_t)} = \frac{\text{\# of instances in dataset assigned to class } c_t}{\text{\# of instances in dataset}}$$

The way in which we estimate the probability $P(\bar{x}|c_t)$ depends on the particular assumptions we make and defines a particular type of Bayes classifier. Here we list a few well known Bayes classifier models:

- **Bayes classifier:** In this model we do not make any simplifying assumptions. To estimate $P(\bar{x}|c_t)$ we can either compute the frequency of instances like $\bar{x}$ which get assigned class c_t in the dataset, if our data is discrete, or we can use the multivariate kernel method if our data is continuous.

- **Naïve Bayes classifier:** In this model we assume that the features that characterize the instances are independent from eachothers conditional on the class c_t . This allows us to make the following simplification:

$$P(\bar{x}|c_t) = \prod_{i=1}^{d} P(x_i|c_t)$$

We can then estimate the various probabilities $P(x_i|c_t)$ by either computing the frequence of occurence of the value x_i in the i-th feature which gets assigned to class c_t, if our data is discrete, or we can use the kernel method on the values of the i-th feature which get mapped to class c_t.

- **Gaussian naïve Bayes Classifier:** In this model we make the same independent assumption as before. This time however we assume that the features are gaussian distributed. That is, the j-th feature follows a normal distribution, $F_j \sim \mathcal{N}(\mu_j, \sigma_j^2)$.

  In this model we can estimate the various probabilities $P(x_i|c_t)$ by estimating the mean μ_j and the variance σ_j^2 of each feature F_j depending on the class c_t and then using the gaussian pdf as follows

$$\hat{P}(x_i|c_t) = \frac{1}{\sqrt{2\pi\hat{\sigma}_j^2}} \cdot e^{-\frac{1}{2}\left(\frac{x_i-\hat{\mu}_j}{\hat{\sigma}_j}\right)^2}$$

  where $\hat{\mu}_j$ and $\hat{\sigma}_j^2$ are the estimated mean and variance.

5.3 Computed accuracy

To measure the performance of a classification algorithm we can proceed as follows: first we split the original dataset into two smaller datasets, one called the *training set*, and the other called the *test set*. Then we use the training set to, as the name suggests, train our classification algorithm, i.e. to estimate the various probabilities involved in our statistical model. Finally, we use the model we trained on the test set, and we compare the result obtained with the correct ones.
One of the most important measures in this case is called *accuracy*, and it is computed as follows: let N_c be the number of instances of the test set which were correctly identified using our classification algorithm, and let N be the total number of instances of the test set. Then the accuracy obtained is:

$$\text{accuracy} := \frac{N_c}{N}$$

Finally follows the results obtained by the various classifiers implemented for assignment V.

Classifier model	**Accuracy**
Bayes Classifier	0.9333
Naïve Bayes classifier	0.9466
Gaussian naïve Bayes Classifier	0.9200