PRUEBA 1

por JAIME DE LOS RIOS MOUVET

Fecha de entrega: 25-sep-2018 05:30a.m. (UTC+0200)
Identificador de la entrega: 1007879918
Nombre del archivo: 220804_JAIME_DE_LOS_RIOS_MOUVET_PRUEBA_1_1934992_1127068208.pdf
Total de palabras: 22821
Total de caracteres: 126036

uc3m | Universidad Carlos III de Madrid

Bachelor´s degree in Telecommunications Technologies Engineering (2017-2018)

Bachelor Thesis

"Python Module Development for Machine Learning Evaluation"

Jaime de los Ríos Mouvet

Tutor: Francisco J. Valverde Albacete

Co-Tutor: Carmen Peláez Moreno

(Leganés, Madrid, September 2018)

ABSTRACT

This technical report focuses on the study and creation of a software library in the programming language "Python" where a series of information-theoretic tools described by Valverde-Albacete and Peláez Moreno are implemented in papers [1, 2, 3]. The bases and concepts defined in the cited articles, delineate the necessary skills to be able to evaluate the effectiveness of techniques and methods used in the generation of machine learning models. Therefore, this document will introduce a visual interpretation of the performance within a multi-class classification scheme. These functionalities will be contained into a Python package, that will be distributed to the whole community.

The library created will cater users the ability of plotting diagrams known as Entropy Triangles. It is used as an exploratory analysis method for calculating the balance equation of entropies for the estimated distributions of the input and the output of classifiers. In this particular case, entropy calculations can be made in two different situations that will be seen in depth later on. To sum up, this tool will endow Python users with the capabilities to measuring and evaluating the performance of their multi-class classifications tasks in a visual way never used before, as well as evaluating the quality of the databases used in the mentioned processes.

Python is a powerful object-oriented open-source scripting language created in the late 80's by Guido van Rossum in the Netherlands, it has a syntax that encourages readable code, is multiplatform and being an open source language, benefits users by having a larger amount of resources and code available, in this case, for data processing and the application of machine learning techniques. Therefore, it is expected that the development of the library will provide a functionality that has not been yet available in the community.

In this document we describe the design and development of the software package. Before that, we set the theoretical backdrop reviewing the implemented tools and their mathematical background. To illustrate the software features and the utility of the tools, we present some examples with a multi-class dataset in which we unbalance the class distribution in different ways. Additionally, we introduce how to use the package using some vignettes representing real cases scenarios. Finally, we accomplish a critical project analysis and recommend future work.

Keywords: Machine Learning, Information Entropy, Classification algorithms, Information-Theory

ACKNOWLEDGEMENTS

My gratitude goes out to the people who have supported me along the road, not only during the preparation of this project, but throughout my entire career, particularly to my parents and family, to whom I will always be grateful. Finally, I would like to thank the teachers who have accompanied me during the development, as I have learned so much in a subject that I especially enjoy and to which I hope to devote myself in the future.

INDEX

FIGURE INDEX

TABLE INDEX

1. INTRODUCTION

This paper presents a complete Python Package for the performance evaluation of machine learning tasks such as data transformation, multi-class classifiers and data bases characterization based on information-theoretic tools. It will be rooted in a series of extensive learnings extracted from [3, 2]

1.1 Motivation

Technology has helped industries to have a descriptive and analytical perspective of the company´s activities they carry out based on data to analyse situations and define strategies. However, since the use machine learning systems, perspective has advanced and adopted a more predictive trend. So instead of using technology to, for example, analyse and describe customer needs, technology is now oriented to detecting data patterns and predicting customer needs based on their characteristics.

For that reason, a sort of different algorithms were developed to deploy this pattern recognition tasks. In particular, one of the most used techniques is the supervised classification, which are algorithms that use a set of classified examples (training data) to assign a classification to a second set of examples.

Although this is one of the most widely used techniques, and its use is widespread, we are faced with a shortage of technical tools that allow us to check how these machine models used to perform. For that reason, Valverde-Albacete and Peláez Moreno created papers displaying a series of information-theory tools that precisely solve this problem and let us evaluate the performance of multiclass classification.

Furthermore, the tools mentioned dealt as the basis for a generalization of the information theory tools provided for the multiclass classification [4], in order to analyze other different machine learning tasks like the evaluation of data feature transformations, or the assessment of information within a database used in a machine learning duty.

The conceptualization of the theoretical concepts was followed by the creation of software in platforms such as Matlab, Weka or R. However, there was not software implemented for one of the most used programming languages nowadays which is Python. In addition, it can be pointed out that Python is one of the most used language when talking about scientific computing, and in special, for Data Science.

For that reason, the aim of this project is the creation of a tool for the evaluation of machine learning methods, by means of tools from information theory, to extend the goal of providing a distributable package in Python, for extending these concepts and functionalities to researchers but also for daily community users.

1.2 Objective

The main study objectives will be to develop a Python package, which will allow us to successfully evaluate the performance of machine learning processes, and at the same time make the package accessible to the user community through repositories such as GitHub or PyPI. For this reason, we will provide a series of manuals for both use and installation so that, without a prior tool´s knowledge, users can easily manage the functionalities we offer.

1.3 Outline

The paper is going to introduce and present all the concepts necessary to understand the code presented to meet the objectives mentioned above, therefore, it will be structured in three parts. A first theoretical section 2 where all the theory behind the tools used are reviewed, from the information measures needed for the calculation, to the software used for the code creation.

In Chapter 3, we will present the architecture of the package implementation system, deepening in the tools used for the project creation and distribution. In addition, we will explain in detail the programming process and the options and difficulties we have had at the time of implementation. Lastly in this section, we will provide a user and installation manual, to provide new users with documentation with which they can take their first steps with the package.

Finally, in the last section 4 we will focus on the software conclusions and the next application steps.

2. THEORETICAL BACKGROUND

Our code development is going to be based on the theories and the tools provided by Valverde-Albacete and Peláez Moreno in papers [3, 2, 1]. For that reason, this section will be based on the reviewing and the explanation of the mathematical background for providing a reference within the document.

2.1 Entropy Measures

Information-theoretic measures can be used to provide tools for analyzing the flow of information from some discrete multivariate source $\bar{X}$ to a discrete multivariate sink of information $\bar{Y}$ joined by a distribution $P_{\overline{XY}}$. From [1] we will use the defined *balance equations* whose normalization can be represented in a ternary plot called *de Finetti* diagram. These balance equations will represent the decomposition of the joint entropy of two random variables for the analysis of the information flow.

During the theoretical analysis, we can distinguish two scenarios while talking from a data flow perspective, one where the information transmission is made by random variables $n = |\bar{X}| = |\bar{Y}| = p = 1$ called *single-input single-output* (SISO) which is described by the joint distribution P_{XY} and a second one where the transfer is established between random vectors, named as *multiple-input-multiple-output* (MIMO) which is described by the joint distribution $P_{\overline{XY}}$ in [3, 2].

In this section, concepts needed for the definition of the balance equations in both scenarios are going to be defined, as well as the information theoretic measures used to define the equations and the posterior ternary diagrams known as Entropy Triangles.

2.1.1 Entropic Measures of Random Variables

Let X and Y be two random variables with some joint distribution P_{XY}, then for the conceptualization of the information transfer between the variables mentioned, some information quantities can be defined from a better understanding of the final entropy equations proposed in [1]

2.1.1.1 Entropies of random variables

The *entropy* is a measure of information [5], which reflects the expected uncertainty of the outcome or the amount of information learnt on average from one instance of a random variable. From a random variable X with a probability mass function $p_x(x)$, the entropy is calculated as

$$H(X) = E\left[\log\frac{1}{P_X}\right] = -\sum_i p(x_i)\log(p(x_i)) \quad (2.1)$$

Note that the entropy only depends on the probability distribution of the variable $p(x)$.

Now consider two random variables X and Y, jointly distributed according to the probability mass function $p(x,y)$. We can now define some measures as the following:

The *joint entropy* of two variables [5], will measure how much uncertainty there is in the two joint events.

$$H(X,Y) = E\left[\log\frac{1}{p(x,y)}\right] = -\sum_{x,y} p(x,y)\log(p(x,y)) \quad (2.2)$$

Secondly, we can define the *conditional entropy* [5] of X given Y which measures the uncertainty remaining in r.v. X when known the value of

$$H(X|Y) = E\left[\log\frac{1}{p(x|y)}\right] = -\sum_{x,y} p(x,y)\log(p(x|y)) \quad (2.3)$$

Relations of the two random variables will be essential for the posterior entropy analysis, as will help us for instance to calculate in our case, the amount of information shared by two variables, or the variation of information between them.

2.1.1.1.1 Properties of Entropy

We can define several properties of the entropic measures seen in [6, 5]:

- *Non-Negativity*: $H(X) \geq 0$, entropy is always non-negative. If $H(X) = 0$, then X is deterministic
- *Chain rule*: We can decompose the joint entropy as follows:

$$H(Y|X) = H(X,Y) - H(X) \quad (2.4)$$

For two variables then the chain rule can will be

$$H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X) \quad (2.5)$$

Noticing that in general $H(X|Y) \neq H(Y|X)$

- *Monotonicity*: Conditioning always reduces entropy:

$$H(X|Y) \leq H(X) \quad (2.6)$$

- *Maximum entropy*: The maximum value is achieved when the random variable X is uniformly distributed, and that value will be

$$H(X) \leq \log(k) \quad (2.7)$$

Being k the number of values that the discrete random variable X could take.

- *Independent variables*: When X and Y are independents variables, then

$$H(X,Y) = H(X) + H(Y)$$
$$H(X|Y) = H(Y) \quad H(Y|X) = H(X) \quad (2.8)$$

Once we understand the basic measures and the properties of the entropies that can be measured when talking about random variables, we are going deeper into the decomposition of the joint entropy of two random variables, defining some regions from [1].

For this case we are going to use two random variables X and Y for the depiction of the joint entropy decomposition in a blueprint called *information-diagram* (i-diagram) [7, 8]. Figure 1 represents the entropy decomposition of the joint distribution P_{XY} where we can find the regions used to characterize the decomposition.

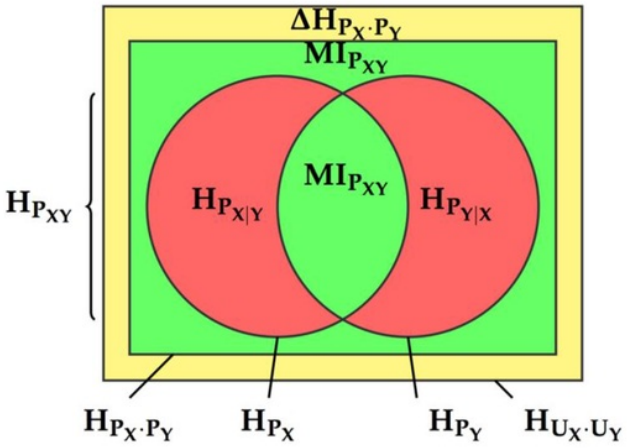


Figure 1. Information diagram of the joint distribution of two random variables joint distribution (from [1])

2.1.1.1.2 Mutual information

The *mutual information* [9] of two variables measures the amount of information that the variables share. This is the information, or uncertainty, that remains when subtracting from one variable the uncertainty of the same variable given the other

$$MI_{XY} = H_X - H_{X|Y} = H_Y - H_{Y|X} \quad (2.9)$$

In Figure 1, the *mutual Information* [3] $MI_{P_{XY}}$ is then represented as the force of the stochastic binding between P_X and P_Y, "towards the outside". It can be also considered the decrease in entropy when passing form distribution $Q_{XY} = P_X \cdot P_Y$ to P_{XY} (outside area)

$$MI_{P_{XY}} = H_{P_X \cdot P_Y} - H_{P_{XY}} \quad (2.10)$$

but it also might be considered "towards the inside"

$$MI_{P_{XY}} = H_{P_X} - H_{P_{X|Y}} = H_{P_Y} - H_{P_{Y|X}} \quad (2.11)$$

Notice that two independent variables do not have *mutual information* ($MI_{XY} = 0$) as $H_{X|Y} = H_X$

2.1.1.1.3 Variation of information

The *variation of information* [10] is the opposite concept to the *mutual information* as it is interpreted as the distance between two random variables. So, the variation of information can be calculated extracting the mutual information present in each variable. It is also called the *residual information*

$$VI_{XY} = H_X + H_Y - 2 \cdot MI_{XY} \tag{2. 12}$$

In Figure 1, the *Variation of information* $VI_{P_{XY}}$ embodies the residual entropy, not used in the binding of the variables, and it is represented by the red area (sum of the conditional entropies) [3]

$$VI_{P_{XY}} = H_{P_{X|Y}} + H_{P_{Y|X}} \tag{2. 13}$$

2.1.1.1.4 Divergence with respect to uniformity

Divergence with respect uniformity [11] $\Delta H_{P_X \cdot P_Y}$, between the joint distribution where P_X and P_Y are independent and the uniform distributions with the same cardinality of events as P_X and P_Y.

$$\Delta H_{P_X \cdot P_Y} = H_{U_X \cdot U_Y} - H_{P_X \cdot P_Y}$$
$$H_{P_X \cdot P_Y} \leq H_{U_X \cdot U_Y} \tag{2. 14}$$

It shows how imbalanced the data of the random variables are, represented in Figure 1 as the bordered yellow area [3]

2.1.1.2 Entropic Measures of Random Vectors

After defining the bivariate entropy measures used in later sections for the calculation of the *balance equation*, where the information is shared between the random variables X and Y, in this section, the extension of the entropies decomposition characterizing a joint distribution P_{XY} of a multivariate inputs and outputs (*MIMO*) is covered. For this reason, before reviewing different flavours of the information measures of multivariate sets for

the development of a general entropy balance equation, lets differentiate between the two scenarios to face as mentioned in [3]

Scenario 1) All the random variables form part of the same set $\bar{X}$ and we are looking at information transfer *within* this set. (Source-MET Section 2.1.1.2.2)

Scenario 2) Random variables are portioned in two different sets $\bar{X}$ and $\bar{Y}$ and we are looking at information transfer *between* the sets. (Channel-MET Section 2.1.1.3)

2.1.1.2.1 Random Vectors as Sources

We will try in this section to generalize the balance equations and entropy triangles to the case of multiple variables $\bar{X}$ taken as joint source of information with a multivariate distribution $\bar{X} \sim P_{\bar{X}}$. For the generalization, new information measures and concepts will be applied, hence, we will first introduce them from [3].

2.1.1.2.2 Entropy of a Multivariate Source

Papers [3, 2] reviews different flavours of information measures describing more than two variables. Since the *i-diagrams* are a powerful tool to visualize the interaction of distribution in the bivariate, we will try to extend them for sets of random variables. However, there may be some caveats for their multivariate use, for instance with the possibility of obtaining some negative "areas" when not considering signed measures of probabilities.

Furthermore, when increasing the number of variables in the analysis, some new areas defined in [8] will appear and affect directly on the behaviour of the joint entropy decomposition of the random vector. For the definitions lets recall the concepts mentioned in [12], where James et al. pointed out that we should call *information* those measures which involve amounts of entropy shared by multiple variables and *entropies* those that do not. This will simplify the way of calculating the multivariate generalizations in next sections

2.1.1.2.2.1 Random Vector variation of information

Following the line mentioned above, we must consider that every random variable has a residual entropy, which is the information not explained by other variables. For that reason we can define the *residual entropy* [2] $H_{P_{X_i|X_i^c}}$ –where $X_i^c = \bar{X} \setminus \{X_i\}$ – which can be grouped in a measure call *residual information* or *multivariate variation of information* $VI_{P_{\bar{X}}}$. This measure is the generalization of the same quantity of the bivariate case, and it is calculated by adding all the residual entropies of the data set.

$$VI_{P_{\bar{X}}} = \sum_{i=1}^{n} H_{P_{X_i|X_i^c}} \tag{2. 15}$$

2.1.1.2.2.2 Multivariate Source Mutual Information

Once defined the *variation of information*, [12] pointed out that some information measures stem from focusing in a particular property of the bivariate mutual information and generalize it to the multivariate setting, and the properties in question are:

1. $MI_{P_{XY}} = H_{P_X} + H_{P_Y} - H_{P_{XY}}$ (2. 16)
2. $MI_{P_{XY}} = H_{P_X} - H_{P_{X|Y}}$ (2. 17)
3. $MI_{P_{XY}} = \sum_{x,y} P_{XY}(x,y) \log \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)}$ (2. 18)

However, these generalizations do not translate inconditionallity to the multivariate setting. For that reason, we are going to define some quantities that will help us to explain the mutual information shared between multiple random variables, considering the next situation.

As mentioned in [3], we can consider the scenario where we are using a vector of random variables $\bar{X} \sim P_{\bar{X}}$, let $\prod_{\bar{X}} = \prod_{i=1}^{n} P_{X_i}$ be the (jointly) independent distribution with similar marginals to $P_{\bar{X}}$. We are going then to consider the Figure 2 as the distribution i-diagram of the random vector $\bar{X} = [X_1, X_2, X_3]$ with $\prod_{\bar{X}} = P_{X_1} \cdot P_{X_2} \cdot P_{X_3}$ as the inner rectangle containing both green areas in figure Figure 2. Then the *mutual information* is decomposed in different properties:

- The *total correlation* $C_{P_{\bar{X}}}$ [13], *integration* [14] or *multiinformation* [15], which is represented by the green area outside $H_{P_{\bar{X}}}$ in Figure 2 and it is a generalization of (2. 16)

$$C_{P_{\bar{X}}} = H_{\prod_{\bar{X}}} - H_{P_{\bar{X}}} \tag{2. 19}$$

- The *Dual total correlation* $D_{P_{\bar{X}}}$ [16, 17] or *interaction complexity* [18] , is a generalization of (2. 17), represented by the green area inside $H_{P_{\bar{X}}}$ in Figure 2

$$D_{P_{\bar{X}}} = H_{P_{\bar{X}}} - VI_{P_{\bar{X}}} \tag{2. 20}$$

- The *interaction information* [19] multivariate mutual information [20] or co information [21] the total amount of information to which all variables contribute. It

is represented by the inner convex area (within the *dual total correlation*) and is the generalization of (2. 18). Notice that this quantity can be negative for $n > 2$ [17]

$$MI_{P_{\bar{X}}} = \sum P_{\bar{X}}(\bar{x}) \log \frac{P_{\bar{X}}(\bar{x})}{\prod_{\bar{X}}(\bar{x})} \tag{2. 21}$$

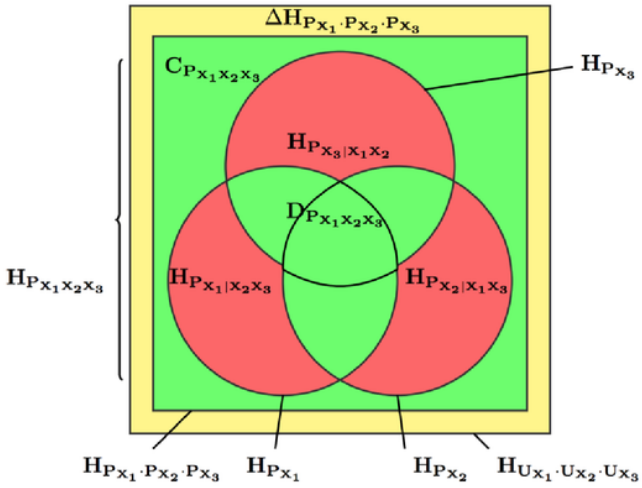


Figure 2. Extended information diagram of a trivariate distribution (from [2])

- The *local exogenous information* [12] or the bound information [22] is the addition of the total correlation and the dual total correlation.

$$M_{P_{\bar{X}}} = C_{P_{\bar{X}}} + D_{P_{\bar{X}}} \tag{2. 22}$$

These measures will be used in 2.2.2 for the representation and definition of the balance equations for the *source multivariate* case.

2.1.1.3 Random Vectors as Joint distributions

The characterization of the joint distribution $P_{\overline{XY}}$ between random vectors $\bar{X}$ and $\bar{Y}$ will be analogous to the bivariate case where we characterize the decomposition of the joint distribution of two random variables. In this we will work to define the same decomposition for multivariate random vectors [3]. So, considering the i-diagram from a random vector in Figure 2 we can visualize the updated diagram from Figure 3 differentiating two random vectors $\bar{X}$ and $\bar{Y}$ with a joint distribution $P_{\overline{XY}}$.

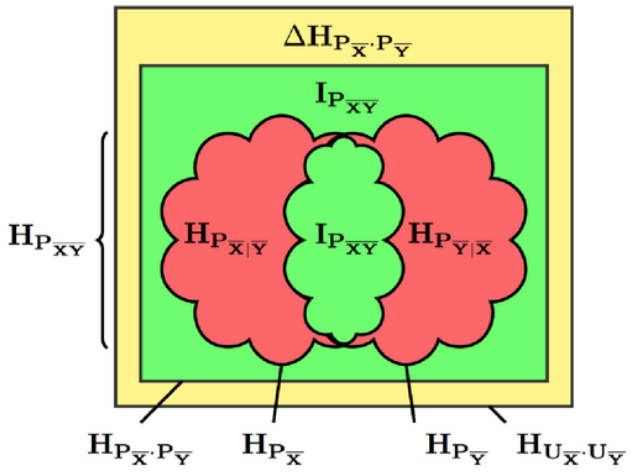


Figure 3. Partioned distribution entropy diagram (from [2])

From this point we can establish proper generalizations of the entropic measures defined in section 2.1.1.1 around $P_{\overline{XY}}$. As mentioned in [3] if we focus on the distinction between two random vectors $\bar{X}$ and $\bar{Y}$, a proper multivariate generalization of the *variation of information* in (2. 13) is

$$VI_{P_{\overline{XY}}} = H_{P_{\bar{X}|\bar{Y}}} + H_{P_{\bar{Y}|\bar{X}}} \tag{2. 23}$$

Which represents the addition of information in $\bar{X}$ not shared with $\bar{Y}$ and vice-versa, and besides it is a positive value as is a sum of entropies, it is captured in the Figure 3 as the red area.

Next, we can consider $U_{\overline{XY}}$, the uniform distribution over the supports of $\bar{X}$ and $\bar{Y}$, and $P_{\bar{X}} \times P_{\bar{Y}}$, the distribution created with the marginals of $P_{\overline{XY}}$ considered independent. Thus, we can define the multivariate divergence with respect uniformity [3] as an analogy of (2. 14)

$$\Delta H_{P_{\bar{X}} \times P_{\bar{Y}}} = H_{U_{\overline{XY}}} - H_{P_{\bar{X}} \times P_{\bar{Y}}} \tag{2. 24}$$

This can be represented as the yellow area in Figure 3. In addition, we can conclude from [3] that the multivariate divergence with respect to uniformity admits splitting as

$$\Delta H_{P_{\bar{X}} \times P_{\bar{Y}}} = \Delta H_{P_{\bar{X}}} + \Delta H_{P_{\bar{Y}}} \tag{2. 25}$$

Because $H_{U_{\overline{XY}}} = H_{U_{\overline{X}}} + H_{U_{\overline{Y}}}$ and $H_{P_{\overline{X}} \times P_{\overline{Y}}} = H_{P_{\overline{X}}} + H_{P_{\overline{Y}}}$. This generalization will help us to introduce the split balance equations in later sections.

2.1.1.3.1 Mutual Information for Pairs of Random Vectors

Although the above concepts have been defined through a generalization of the bivariate case, we now deal with the problem of defining the mutual information between 2 random vectors. For that reason, we can define from [3] a "remanent information" represented by the inner area in the i-diagram of Figure 3

$$I_{P_{\overline{XY}}} = H_{P_{\overline{XY}}} - VI_{P_{\overline{XY}}} \tag{2.26}$$

As mentioned in [3] this can be easily "refocused" on each of the subsets of the partition:

Lemma 1: Let $P_{\overline{XY}}$ be a discrete joint distribution. Then

$$H_{P_{\overline{X}}} - H_{P_{\overline{X}|\overline{Y}}} = H_{P_{\overline{Y}}} - H_{P_{\overline{Y}|\overline{X}}} = I_{P_{\overline{XY}}} \tag{2.27}$$

Proof. Recalling that the conditional entropies are easily related to the joint entropy by the chain rule $H_{P_{\overline{XY}}} = H_{P_{\overline{X}}} + H_{P_{\overline{Y}|\overline{X}}} = H_{P_{\overline{Y}}} + H_{P_{\overline{X}|\overline{Y}}}$, simply subtract $VI_{P_{\overline{XY}}}$

Lemma 2: Let $P_{\overline{XY}}$ be a discrete joint distribution [3]. Then

$$I_{P_{\overline{XY}}} = H_{P_{\overline{X}} \times P_{\overline{Y}}} - H_{P_{\overline{XY}}} \tag{2.28}$$

Proof. Considering the entropy decomposition of $P_{\overline{X}} \times P_{\overline{Y}}$

$$\begin{aligned} H_{P_{\overline{X}} \times P_{\overline{Y}}} - H_{P_{\overline{XY}}} &= H_{P_{\overline{X}}} + H_{P_{\overline{Y}}} - \left(H_{P_{\overline{Y}}} + H_{P_{\overline{X}|\overline{Y}}}\right) \\ &= H_{P_{\overline{X}}} - H_{P_{\overline{X}|\overline{Y}}} = I_{P_{\overline{XY}}} \end{aligned} \tag{2.29}$$

This is the quantity of information required to bind $P_{\overline{X}}$ and $P_{\overline{Y}}$, or in other words the amount of information lost from $P_{\overline{X}} \times P_{\overline{Y}}$ to achieve a binding in $P_{\overline{XY}}$. It is represented by the outermost green area [3] in Figure 3.

Lemma 3: Let $P_{\overline{XY}}$ be a discrete joint distribution. Then

$$I_{P_{\overline{XY}}} = \sum_{i,j} P_{\overline{XY}}(x_i, y_i) \log \frac{P_{\overline{XY}}(x_i, y_i)}{P_{\bar{X}}(x_i) P_{\bar{Y}}(y_i)} \tag{2.30}$$

Proof. With an easy manipulation we can conclude that

$$\sum_{i,j} P_{\overline{XY}}(x_i, y_i) \log \frac{P_{\overline{XY}}(x_i, y_i)}{P_{\bar{X}}(x_i) P_{\bar{Y}}(y_i)} = H_{P_{\bar{X}}} - H_{P_{\bar{X}|\bar{Y}}} = I_{P_{\overline{XY}}} \tag{2.31}$$

Multivariate mutual information [3] it can be represented as the inner green area in the Figure 3 also known as "*remanent information*" or the quantity of information required to bind $P_{\bar{X}}$ and $P_{\bar{Y}}$ but also represented by the outer green area of the same figure, presenting the amount of information lost from $P_{\bar{X}} \cdot P_{\bar{Y}}$ to $P_{\overline{XY}}$

$$I_{P_{\overline{XY}}} = H_{P_{\overline{XY}}} - VI_{P_{\overline{XY}}} = H_{P_{\bar{X}} \times P_{\bar{Y}}} - H_{P_{\overline{XY}}} \tag{2.32}$$

2.2 The Information-Theoretic Approach to Evaluating Supervised Classification Tasks

Following the studies presented by Valverde-Albacete and Peláez Moreno we have reviewed the theoretical basis and a set of tools for analyzing the flow of information in a classification task. As mentioned article [1], the use of information theory to assess classifiers is grounded on the analogy between a communication channel and a classification process since the input and output of a classifier can be described as a set of discrete random variables.

Figure 4. Conceptual representation of a supervised classification architecture as a communication channel (from [2])

Based on the initial scheme of [1], we will introduce the conceptualization of a supervised machine learning task called multi-classification, cast in an information theoretic setting as shown in Figure 4. First of all, there will be a set of m relations [2] of a random vector $\bar{X}$ of features or *observations* variables paired with as many realizations of a class variable K. The set of values $\{(k^i, \bar{x}^i)\}_{1 \le i \le m}$ will be called dataset.

The observations, then may be transformed in other instances of vector $\bar{Y}$ through a *transformation* function $f: \bar{X} \to \bar{Y}$ where $\bar{x}^i \to \bar{y}^i = f(\bar{x}^i)$ with the desired

characteristics. For the supervised classification [2], learning a classifier is the subtask of inducing a function $k: \bar{Y} \rightarrow K,\ \bar{y}^i \rightarrow \hat{k}^i = k(\bar{y}^i)$ that tries to estimate $\hat{K}$.

In this context we will use the defined theoretic tools to assess the performance of the classification. We can first notice in the end-to-end panorama, that we can measure de effectiveness of the estimation of $\hat{K}$ from K, using the *Channel Binary Entropy Triangle* (section 2.2.1). Then, we will move to evaluating the features information contained on the database by the *Source Multivariate Entropy Triangle* use (section 2.2.2). Finally, we will measure the quality and the information transferred within the transformation block $f: \bar{X} \rightarrow \bar{Y}$ with the *Channel Multivariate Entropy Triangle* (section 2.2.3).

2.2.1 The Channel Binary Entropy Triangle (CBET)

The *Channel Binary Entropy Triangle* is mainly [4] used as an exploratory data analysis tool for visual assessment, helping to visualize the performance of supervised classifiers in a forthright way. We can figure out the scenario in which we have the contingency matrix of a classifier on a supervised classification task for the random variables of true class $K \sim P_K$ and the predicted labels $\hat{K} \sim P_{\hat{K}}$. These variables now will play the role as shown in the binary case of section 2.1.1.1 of the distributions P_X and P_Y.

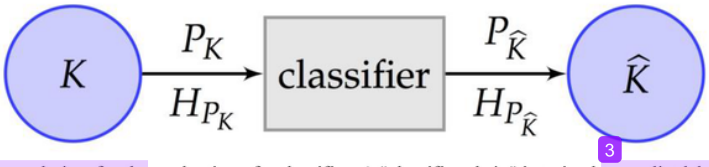


Figure 5. end-to-end view for the evaluation of a classifier. A "classifier chain" is trained to predict labels K from the true labels (from [3])

The transfer of information from single input to single output is going to be measured using the well-known measures dating back to Shannon [9] but considering an extended relation between them in the form of a balance equation for their joint entropy based in [4]. So using the previous concepts defined in section 2.1.1.1, let´s depict the disengaging of the joint entropy of two random variables shown on the i-diagram of Figure 1 as the 3 main regions represented:

- *Divergence with respect uniformity* [4]

$$\Delta H_{P_X \cdot P_Y} = H_{U_X \cdot U_Y} - H_{P_X \cdot P_Y}$$
$$H_{P_X \cdot P_Y} \leq H_{U_X \cdot U_Y} \qquad (2.33)$$

- *Mutual Information* [9]

$$MI_{P_{XY}} = H_{P_{X \cdot P_Y}} - H_{P_{XY}}$$
$$H_{P_X} - H_{P_{X|Y}} = H_{P_Y} - H_{P_{Y|X}} \quad (2.34)$$

- *Variation of information* [10]

$$VI_{P_{XY}} = H_{P_{X|Y}} - H_{P_{Y|X}} \quad (2.35)$$

2.2.1.1 Balance Equation

After defining the concepts needed to decompose the information measures for the joint distribution, we can now write the *entropy balance equation* defined in [4] of two discrete random variables X and Y:

$$H_{U_{X \cdot U_Y}} = \Delta H_{P_{X \cdot P_Y}} + 2 * MI_{P_{XY}} + VI_{P_{XY}}$$
$$0 \leq \Delta H_{P_{X \cdot P_Y}}, 2 * MI_{P_{XY}}, VI_{P_{XY}} \leq H_{U_{X \cdot U_Y}} \quad (2.36)$$

2.2.1.2 Split Balance Equations

The logarithmic nature of the entropy gives the possibility of decomposing the balance equation of a joint distribution for analyzing the behaviour of each variable distribution as shown in the modified i-diagram in Figure 6. For instance, we can split some joint entropy measurement into an individual values addition ($H_{U_{X \cdot U_Y}} = H_{U_X} + H_{U_Y}$) as mentioned in [3]. For that reason, we can expand the formula (2. 33) to

$$\Delta H_{P_{X \cdot P_Y}} = H_{U_{X \cdot U_Y}} - H_{P_{X \cdot P_Y}} = \left(H_{U_X} - H_{P_X}\right) + \left(H_{U_Y} - H_{P_Y}\right)$$
$$= \Delta H_{P_X} + \Delta H_{P_Y} \quad (2.37)$$

Extending it to the balance equation (2. 36) will generate the split version [3]

$$H_{U_X} = \Delta H_{P_X} + MI_{P_{XY}} + H_{P_{X|Y}}$$
$$0 \leq \Delta H_{P_{X \cdot}}, MI_{P_{XY}}, H_{P_{X|Y}} \leq H_{U_X} \quad (2.38)$$

$$H_{U_Y} = \Delta H_{P_Y} + MI_{P_{XY}} + H_{P_{Y|X}}$$
$$0 \leq \Delta H_{P_Y}, MI_{P_{XY}}, H_{P_{Y|X}} \leq H_{U_Y} \quad (2.39)$$

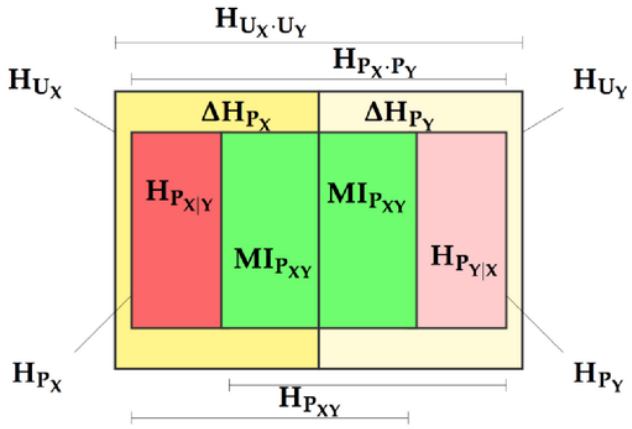


Figure 6. Schematic split entropy diagram (from [1])

2.2.1.3 Entropy Triangle

The entropy triangle is a representation of the balance equations on the *de Finetti*[1] diagram. Considered as a ternary chart, is a barycentric diagram used to represent 3 variables that combined, sum a given constant value. The variables of the ternary diagram always will sum a constant value $a + b + c = k$ $(k > 0)$ therefore, the existence of a relationship of dependence between two variables and the third variable can be noted, for instance, $a = k - (b + c)$.

Then, for computing the entropy triangle, some normalization in $H_{U_X \cdot U_Y}$ is going to be applied in the equation (2. 36)

$$1 = \Delta H'_{P_X \cdot P_Y} + 2 * MI'_{P_{XY}} + VI'_{P_{XY}} \tag{2. 40}$$

$$0 \leq \Delta H'_{P_X \cdot P_Y}, 2 * MI'_{P_{XY}}, VI'_{P_{XY}} \leq 1$$

Which represents the 2-simplex in normalized $\Delta H'_{P_X \cdot P_Y} \times 2 * MI'_{P_{XY}} \times VI'_{P_{XY}}$ space. At the end, each joint distribution P_{XY} can be characterized by its joint entropy coordinates, or entropic composition [23] $F(P_{XY}) = [\Delta H'_{P_X \cdot P_Y}, 2 * MI'_{P_{XY}}, VI'_{P_{XY}}]$ whose projection onto the $(1,1,1)$ direction vector is it´s de Finetti diagram. This particular case is called the *Channel Bivariate Entropy Triangle*, CBET and will be used for the evaluation of a multi-class classification task.

[1] https://en.wikipedia.org/wiki/De_Finetti_diagram

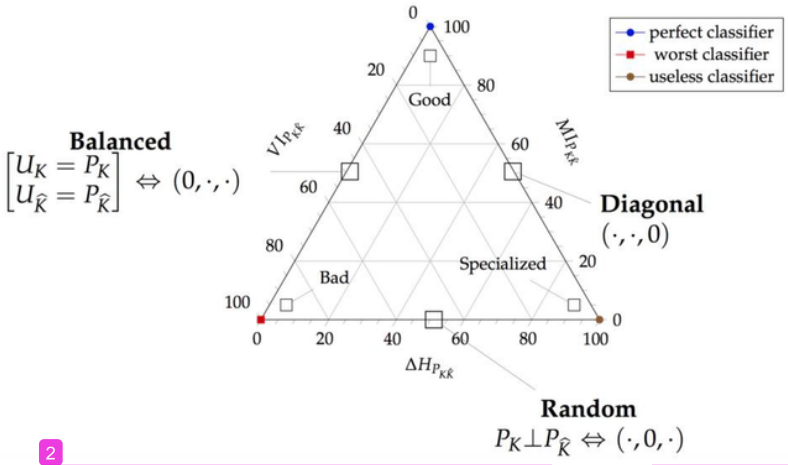


Figure 7. Schematic CBET as applied to supervised classifier assessment (From [3]). An actual triangle show dots for each classifier, and none of the callouts for specific classifiers. The callouts situated at the sides of the triangles apply to the whole side

The representation of the balance equation on the diagram provides a graphical assessment of joint distribution in terms of information theory. We can review some borderline cases from [4], defined by regions where one of the components of the balance equation (2. 40) are equal to zero. With X and Y with joint distribution P_{XY}, then at the ET we can differ some characteristic regions

- Left Side ($\Delta H'_{P_X \cdot P_Y} = 0$) [4] This region represents variables that has a uniform joint distribution for its input and output variables.

- Right Side ($VI'_{P_{XY}} = 0$) [4] portrays the regions in which there is not variation of information, meaning that all the information from a variable X to Y is shared or transferred while talking about a transmission.

- Bottom Side ($MI'_{P_{XY}} = 0$) [4] depict the scenario when the mutual information between 2 variables X and Y is equals to 0, meaning that X and Y are independents variable that not share any information.

Figure 7 presents the Entropy triangle of the mentioned cases points in the case we evaluate the performance of classifiers, that fulfils the conditions in the vertices of the triangle.

2.2.1.4 Split Entropy Triangle

The entropy triangle is going to represent the separated balance equations as well of each of the marginal distributions. We will just need to normalize the splitted equations [1] by the corresponding value H_{U_X} or H_{U_Y}

$$1 = \Delta' H_{P_X} + MI'_{P_{XY}} + H'_{P_{X|Y}}$$
$$0 \leq \Delta' H_{P_X}, MI'_{P_{XY}}, H'_{P_{X|Y}} \leq 1 \quad (2.41)$$

$$1 = \Delta' H_{P_Y} + MI'_{P_{XY}} + H'_{P_{Y|X}}$$
$$0 \leq \Delta H_{P_Y}, MI_{P_{XY}}, H'_{P_{Y|X}} \leq 1 \quad (2.42)$$

Since these are also equations on a 2-simplex, we can actually represent the coordinates $F_X(P_{XY}) = \left[\Delta H'_{P_X}, MI'_{P_{XY}}, H'_{P_{X|Y}}\right]$ and $F_Y(P_{XY}) = \left[\Delta H'_{P_Y}, MI'_{P_{XY}}, H'_{P_{Y|X}}\right]$ in the same triangle side by side the original $F(P_{XY})$ [1], whereby the representation seems to split in two.

2.2.1.5 Measurements: Confusion Matrix

We will need for the entropy balance equations definition on a multi-class classification task, some tools for determining the entropic measures defined in section 2.2.1.1. From [4] consider a set of k true classes $\{x_1, x_2, \ldots, x_k\}$ and a discrete random variable X following a prior class distribution P_x, and a set of N instance patterns each belonging to one of the true classes. Then a classification is a process where each of these instances are assigned to one of the predicted classes $\{y_1, y_2, \ldots, y_k\}$ generating a discrete random variable Y with a prior distribution P_y .

Thus, we can consider the joint event of a classification process consisting on the presentation of an instance of an input class $X = X_i$ and deciding a predicted output class $Y = Y_j$.

For the performance evaluation then, a *confusion matrix* C_{XY} is type of contingency table[2] that counts the occurrences of the joint events. This matrix tallies as mentioned before the true labels x_i of the entries (inputs of a classifier), against the predicted ones y_i (outputs of the classifiers) to check the predictors performance.

Each element of the table (i, j) is the number of instances in the test set which belonging to the class x_i are classified as y_j. $(i = j$ means a correct classification)

Then, the maximum likelihood estimate for the joint probability of every label-prediction pair [4] is the element of the matrix at such index divided by the total number of instances of the test set

$$P_{XY} = \frac{C_{XY}}{N} \quad (2.43)$$

[2] https://stattrek.com/statistics/dictionary.aspx?definition=contingency%20table

Hence, we can estimate the joint probabilities and their marginals. Since the calculation of entropy only depends on the distributions of the variables, through the matrixes of confusion we will easily obtain the desired measurements. Then, the contingency matrix is going to be used in our analysis for calculation of the entropic parameters defined in section 2.2.1.3

2.2.2 The Source Multivariate Entropy Triangle (SMET)

The Source Multivariate Entropy Triangle is a multivariate extension of the bivariate channel case, created for the representation for the entropic content of multivariate distributions where we can analyze, for instance, which features contribute the most to the total bound information in a classification dataset.

The Figure 8 extracted from [2] presents the case gives a schematic representation of a machine learning task for the case of supervised multi-class classification. We can use the framework developed in this paper to analyze the information content of any of the virtual data sources represented by the dashed rectangles as instances of those systems presented in Figure 4. Specially we may study statistical multivariate sources $\overline{K}$.

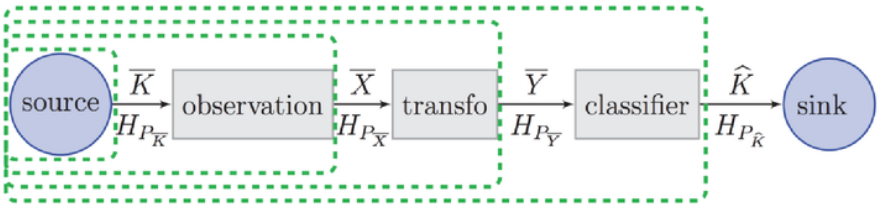


Figure 8. Schematic representation of a multi-class/multi-label classification task with points where the SMET is applicable (from [2]). The dotted boxes represent measuring points where the abstraction of a multivariate source could be of interest: for investigating the classification labels themselves $\overline{K}$, the features as issued from a process of observation $\overline{X}$, as observed after some feature transformation $\overline{Y}$, or even the result of classification $\hat{Y}$.

2.2.2.1 Balance Equations

Once we have the tools necessaries for understanding the composition of either the mutual information of random vectors or the variation of information shown in section (2.1.1.2.2), then we can represent as in the previous sections the measures needed for defining the general multivariate entropic source balance equation as the following:

- Divergence with the uniform multivariate distribution [2] $\Delta H_{\prod_{\overline{X}}}$ represented by the outer yellow zone of the Figure 2

$$\Delta H_{\prod_{\bar{X}}} = H_{U_{\bar{X}}} - H_{\prod_{\bar{X}}}$$

$$\Delta H_{\prod_{\bar{X}}} = \sum_{i=1}^{n} H_{U_{X_i}} - \sum_{i=1}^{n} H_{P_{X_i}} = \sum_{i=1}^{n} H_{U_{X_i}} - H_{P_{X_i}} = \sum_{i=1}^{n} \Delta H_{P_{X_i}} \quad (2.44)$$

- *Mutual information or Bound information* [2] $M_{P_{\bar{X}}}$ shown in the 2.1.1.2.2 is represented by the sum of all the red zones at Figure 2.

$$M_{P_{\bar{X}}} = C_{P_{\bar{X}}} + D_{P_{\bar{X}}} =$$
$$= H_{\prod_{\bar{X}}} - H_{P_{\bar{X}}} + H_{P_{\bar{X}}} - VI_{P_{\bar{X}}} = H_{\prod_{\bar{X}}} - VI_{P_{\bar{X}}} \quad (2.45)$$

- *Multivariate variation of information* [2] $VI_{P_{\bar{X}}}$ shown in the Figure 2 is represented by the sum of all the red zones (repeated from equation (2. 15))

$$VI_{P_{\bar{X}}} = \sum_{i=1}^{n} H_{P_{X_i|X_i^c}} \quad (2.46)$$

These are the necessary measures needed for extending the balance equation for a bivariate case into the multivariate one. Paper [2] presents some theorems to introduce and derive the aggregate and the split entropy balance equation for a multivariate source of data.

Theorem 1. [2] (Aggregate source multivariate balance equation). Let $P_{\bar{X}}$ be an arbitrary discrete distribution over the set of random variables $\bar{X}$. Then, the following balance equation holds

$$H_{U_{\bar{X}}} = \Delta H_{\prod_{\bar{X}}} + M_{P_{\bar{X}}} + VI_{P_{\bar{X}}}$$
$$0 \leq \Delta H_{\prod_{\bar{X}}}, M_{P_{\bar{X}}}, VI_{P_{\bar{X}}} \leq H_{U_{\bar{X}}} \quad (2.47)$$

This equation will be the generalization for the multivariate source case of the entropy balance equation.

2.2.2.2 Multi-Split Balance Equations

From the general equations, we're going to try to get the multi-split balance equations for evaluating independently the information measures of each variable. To do so, we are going to rely on the second theorem presented in paper [2] which reads as follows

Theorem 2 (Multi spit source multivariate balance equation [2]) Let $P_{\bar{X}}$ be an arbitrary discrete distribution over the set of random variables $\bar{X} = \{X_i\}_{i=1}^{n}$, then, with the definitions above, the balance equation holds for each variable individually:

$$H_{U_{X_i}} = \Delta H_{P_{X_i}} + M_{P_{X_i}} + H_{P_{X_i|X_i^c}} \quad 1 \leq i \leq n$$

$$0 \leq \Delta H_{P_{X_i}}, M_{P_{X_i}}, H_{P_{X_i|X_i^c}} \leq H_{U_{X_i}} \quad 1 \leq i \leq n \tag{2.48}$$

With both equations, the aggregated and the splitted now we can define the source entropy triangle for the multivariate case.

2.2.2.3 Entropy Triangle

A generalization from the balance equations and entropy triangles defined in section () to are made to the case of multiple variables $\bar{X}$ taken as a joint source of information. So, as it´s reported and derived in paper [2] we now may normalize the aggregate source multivariate balance equation by $H_{U_{\bar{X}}}$

$$1 = \Delta H'_{\prod_{\bar{X}}} + M'_{P_{\bar{X}}} + VI'_{P_{\bar{X}}}$$

$$0 \leq \Delta' H_{\prod_{\bar{X}}}, M'_{P_{\bar{X}}}, VI'_{P_{\bar{X}}} \leq 1 \tag{2.49}$$

Then, from [2] we can define the composition $F(P_{\bar{X}}) = \left[\Delta' H_{\prod_{\bar{X}}}, M'_{P_{\bar{X}}}, VI'_{P_{\bar{X}}}\right]$ for the representation of the ternary diagram on the aggregate entropy. As in the channel case, we can now define some regions [2] of the multivariate source representation:

• If $P_{\bar{X}} = \prod_{\bar{X}} = \prod_{i=1}^{n} P_{X_i}$, then $F(P_{\bar{X}}) = [\cdot, 0, \cdot]$ is the location of distributions with independent marginals.

• If $P_{X_i} = U_{X_i}, 1 \leq i \leq n$ then $F(P_{\bar{X}}) = [0, \cdot, \cdot]$ is the geometric location of distribution with uniform marginals. $\Delta H_{P_{X_i}} = 0$

• If $P_{X_i} = P_{X_j}$, $i \neq j$ then $F(P_{\bar{X}}) = [\cdot, \cdot, 0]$ is the locus of distribution with identical marginals and in general high bound information

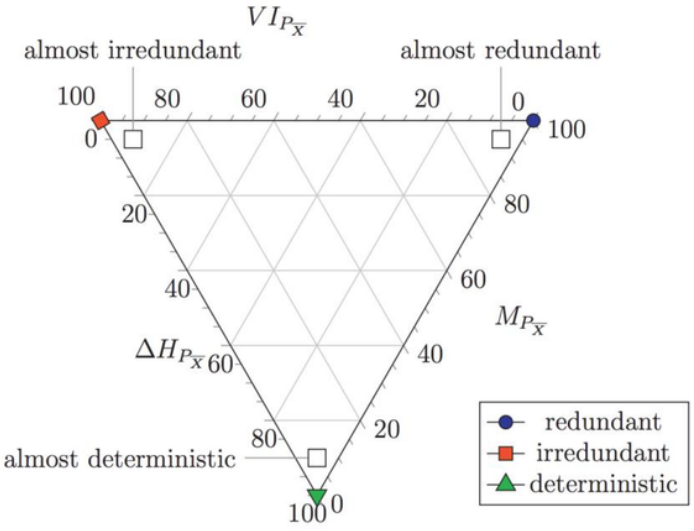


Figure 9. Conceptually annotated Source Multivariate Entropy Triangle (from [2])

In this case, the interpretations of the variables of the SMET are different as they were in the channel triangle [2]. The multivariate residual entropy $VI_{P_{\bar{X}}}$ is the sum of amounts of information singularly captured by each variable, then the total bound information $M_{P_{\bar{X}}}$ can be analyzed as the redundancy in every portion residing in (at least) two different variables. Finally, the divergence from uniformity is the maximal randomness of the source of information that has not been realized and it´s not available for latter processing.

Figure 9 represents the SMET [2], that seems a representation of the channel case, but rotated. That happens because of the divergence from uniformity is a deleterious quantity of information, that's why is represented as a down-growing quantity by rotating the triangle. It also shown the coordinates represented the boundary cases that represent the areas that will characterize the data sources analyzed.

2.2.2.4 Split Source Multivariate Entropy Triangle (split-SMET)

When plotting the aggregate visualization of a joint distribution of a sink of information $\bar{X}$, we are representing an average of the component decomposition [2] for different variables. However, we can disaggregate the visualization like in the previous section for a better unique variable analysis.

From (2. 49) we can normalize the equation for the split balance but on each of the individuals $H_{U_{X_i}}$ for obtaining:

$$1 = \Delta H'_{P_{X_i}} + M'_{P_{X_i}} + H'_{P_{X_i|X_i^c}} \quad 1 \leq i \leq n$$
$$0 \leq \Delta H'_{P_{X_i}}, M'_{P_{X_i}}, H'_{P_{X_i|X_i^c}} \leq 1 \tag{2. 50}$$

Then for each multivariate $\bar{X} = \{X_i\}_{i=1}^n$ we can write for each marginal P_{X_i}, the coordinates in a de Finetti diagram as $F(P_{X_i}) = \left[\Delta H'_{P_{X_i}}, M'_{P_{X_i}}, H'_{P_{X_i|X_i^c}}\right]$. They will be represented in the same entropy triangle as the multi-split Source Multivariate Entropy Triangle.

2.2.3 The Channel Multivariate Entropy Triangle (CMET)

Once introduced the measures needed to proceed with multivariate inputs and outputs, we will rely on theorems defined in [3] for extracting the final aggregated multivariate entropy balance equation and whether if it is possible to split the equation as in the bivariate case. From Figure 10, we are going to represent the transfer of information between two random vectors, casting the transformation scenario into a communication scenario.

In a communication channel, let $\bar{X} = \{X_i | \ 1 \leq i \leq n\}$ be a set of discrete random variables with joint multivariate distribution $P_{\bar{X}} = P_{X_1 \ldots X_n}$, and the corresponding marginals $P_{X_i}(X_i) = \sum_{j \neq i} P_{\bar{X}}(\bar{X})$ where $\bar{x} = x_1 \ldots x_n$ is a tuple of n elements. $\bar{Y} = \{Y_j \mid 1 \leq j \leq l\}$, with $P_{\bar{Y}} = P_{Y_1 \ldots X_l}$ and the marginals P_{Y_j}. Furthermore let $P_{\overline{XY}}$ be the joint distribution of the $(n + l)$-length tuples $\overline{XY}$.

Figure 10. Transformation block implementing $\bar{Y} = f(\bar{X})$). $\bar{X}$ is the data source and $\bar{Y}$ the transformed data (from [24])

Then, $\bar{X}$ carries a prior information and $\bar{Y} = f(\bar{X})$ is derived from it, as shown in Figure 10, where $\bar{X}$ represents the data source (real data) and $\bar{Y}$ represents the transformed data $\bar{Y} = f(\bar{X})$. In this scenario, the point of interest is in whether all of the relevant information extant in $\bar{X}$ is transferred to $\bar{Y}$, and for that reason, we are going to use the generalization of the balance equations for the decomposition of the joint distribution $P_{\overline{XY}}$ presented in [24].

2.2.3.1 Balance Equation

From [3, 24] we can extract another theorem that will allows us to define the balance equations for the joint multivariate case using the measures defined in section 2.1.1.3.

Theorem 1. [3] Let $P_{\overline{XY}}$ be a discrete joint distribution. Then the following decomposition holds:

$$H_{U_{\bar{X}} \times U_{\bar{Y}}} = \Delta H_{P_{\bar{X}} \times P_{\bar{Y}}} + 2 * I_{P_{\overline{XY}}} + VI_{P_{\overline{XY}}}$$
$$0 \leq \Delta H_{P_{\bar{X}} \times P_{\bar{Y}}}, 2 * I_{P_{\overline{XY}}}, VI_{P_{\overline{XY}}} \leq H_{U_{\bar{X}} \times U_{\bar{Y}}} \quad (2.51)$$

Proof. From [3] we have $H_{U_{\bar{X}} \times U_{\bar{Y}}} = \Delta H_{P_{\bar{X}} \times P_{\bar{Y}}} + H_{P_{\bar{X}} \times P_{\bar{Y}}}$ whence by introducing (2. 28) and (2. 26) we obtained:

$$H_{U_{\bar{X}} \times U_{\bar{Y}}} = \Delta H_{P_{\bar{X}} \times P_{\bar{Y}}} + I_{P_{\overline{XY}}} + H_{P_{\overline{XY}}} =$$
$$= \Delta H_{P_{\bar{X}} \times P_{\bar{Y}}} + I_{P_{\overline{XY}}} + I_{P_{\overline{XY}}} + VI_{P_{\overline{XY}}} \quad (2.52)$$

This theorem is followed by a series of conditions, which will fix the lower bounds on the entropy triangle [3].

1) $\bar{X}$ marginal uniformity when $H_{P_{\bar{X}}} = H_{U_{\bar{X}}}$, $\bar{Y}$ marginal uniformity when $H_{P_{\bar{Y}}} = H_{U_{\bar{Y}}}$ and *marginal uniformity* when both conditions cooccur.
2) Marginal independence, when $P_{\overline{XY}} = P_{\bar{X}} \times P_{\bar{Y}}$
3) $\bar{Y}$ determines $\bar{X}$ when $H_{P_{\bar{X}|\bar{Y}}} = 0$, $\bar{X}$ determines $\bar{Y}$ when $H_{P_{\bar{Y}|\bar{X}}} = 0$ and mutual determination, when $H_{P_{\bar{X}|\bar{Y}}} = H_{P_{\bar{Y}|\bar{X}}} = 0$

Note that the conditions mentioned above are independent of each other and in the other hand, they conditionate the values of the other variables in the balance equations.

2.2.3.2 Split Channel Multivariate Balance Equation

As we did in previous sections the equation (2. 51) can be decomposed in split equations. Following the previous theorem in the paper [3, 24] we can find other theorem to concrete whether the balance equation for multivariate random vectors also admits splitting.

Theorem 2. [3] Let $P_{\overline{XY}}$ be a discrete joint distribution. Then the Channel Multivariate Entropy Balance equation can be split as:

$$H_{U_{\overline{X}}} = \Delta H_{P_{\overline{X}}} + I_{P_{\overline{X}\overline{Y}}} + H_{P_{\overline{X}|\overline{Y}}}$$
$$0 \leq \Delta H_{P_{\overline{X}}}, I_{P_{\overline{X}\overline{Y}}}, H_{P_{\overline{X}|\overline{Y}}} \leq H_{U_{\overline{X}}} \quad (2.53)$$

$$H_{U_{\overline{Y}}} = \Delta H_{P_{\overline{Y}}} + I_{P_{\overline{X}\overline{Y}}} + H_{P_{\overline{Y}|\overline{X}}}$$
$$0 \leq \Delta H_{P_{\overline{Y}}}, I_{P_{\overline{X}\overline{Y}}}, H_{P_{\overline{Y}|\overline{X}}} \leq H_{U_{\overline{Y}}} \quad (2.54)$$

This can be proved in a similar way we did in equations (2. 41)(2. 42). These quantities will be *non-negative.*

2.2.3.3 Channel Multivariate Entropy Triangle

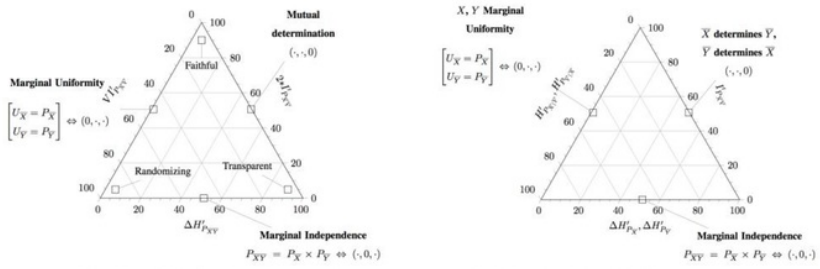
The generalized formula for representing the information balance of a multivariate transformation after the normalization of (2. 51) for the triangle representation will be

$$1 = \Delta H'_{P_{\overline{X}} \times P_{\overline{Y}}} + 2 * I'_{P_{\overline{X}\overline{Y}}} + VI'_{P_{\overline{X}\overline{Y}}}$$
$$0 \leq \Delta H'_{P_{\overline{X}} \times P_{\overline{Y}}}, 2 * I'_{P_{\overline{X}\overline{Y}}}, VI'_{P_{\overline{X}\overline{Y}}} \leq 1 \quad (2.55)$$

So definitely, this is the general *de Finetti* equation of the aggregated channel Multivariate Entropy Triangle, whose proportions for a joint distribution are:

$$F(P_{XY}) = \left[\Delta H'_{P_{\overline{X}} \times P_{\overline{Y}}}, 2 * I'_{P_{\overline{X}\overline{Y}}}, \quad VI'_{P_{\overline{X}\overline{Y}}}\right] \quad (2.56)$$

The tool basically is a graphical assessment of multivariate joint distribution. The Figure 11 shows the portrayal of this entropy triangle, and the representation of the *theorem 1* limits conditions mentioned in [24].



(a) Schematic CMET with a formal interpretation

(b) Schematic split CMET with a formal interpretation

Figure 11. Schematic Channel Multivariate Entropy Triangles (CMET) showing interpretable zones and extreme cases using formal conditions (from [24])

For the bounds determination we will list the bounding cases presented in [24] :

- The lower side of the triangle with $I'_{P_{\overline{XY}}} = 0$, are affected by the marginal independence of blocks $\bar{X}$ and $\bar{Y}$ meaning that $P_{\overline{XY}} = P_{\bar{X}} \times P_{\bar{Y}}$ the joint distribution does not share information. Then $F(P_{\overline{XY}}) = [\cdot\,,\ 0\,,\ \cdot]$
- The right side of the triangle with $VI'_{P_{\overline{XY}}} = 0$, described with mutual determination $H_{P_{\bar{X}|\bar{Y}}} = H_{P_{\bar{Y}|\bar{X}}} = 0$, is the locus of partitioned joint distributions whose groups do not carry supplementary information to that provided by the other group [24]. Being $F(P_{\overline{XY}}) = [\cdot\,,\ \cdot\,,\ 0]$
- The left side with $\Delta H'_{P_{\bar{X}} \times P_{\bar{Y}}} = 0$, describing distributions with uniform marginals $U_{\bar{X}} = P_{\bar{X}}$ and $U_{\bar{Y}} = P_{\bar{Y}}$, is the locus of partitioned joint distributions that offer as much potential information for transformations as possible [24]. $F(P_{\overline{XY}}) = [0,\ \cdot\,,\ \cdot]$

Some descriptions can be obtained from the preceding conclusions mentioned, evaluating the areas that correspond to the vertices of the triangle as shown in [3].

First, if we want a transformation from $\bar{X}$ to $\bar{Y}$ to be faithful, then we want to maximize the information used for mutual determination $I'_{P_{\overline{XY}}} \to 1$, equivalently, minimizing the divergence from uniformity $\Delta H'_{P_{\bar{X}} \times P_{\bar{Y}}} \to 0$ and the information that only belongs to each of the blocks [3] in the partition $VI'_{P_{\overline{XY}}} \to 0$. Then

However, if the coordinates of a distribution [3] lay close to the left vertex $VI'_{P_{\overline{XY}}} \to 1$, then it shows marginal uniformity $\Delta H'_{P_{\bar{X}} \times P_{\bar{Y}}} \to 0$ but shares little or no information between the blocks $I'_{P_{\overline{XY}}} \to 0$, hence it must be a randomizing transformation.

Finally, the distributions whose coordinates lay close to the right vertex $\Delta H'_{P_{\bar{X}} \times P_{\bar{Y}}} \to 1$ are essentially deterministic and in that sense carry no information $I'_{P_{\overline{XY}}} \to 0$, $VI'_{P_{\overline{XY}}} \to 0$. Indeed [3], in this instance there does not seem to exist a transformation, whence we call them rigid.

2.2.3.4 Split Channel Entropy Triangle

The purpose of the representation of the split balance equations in the same diagram, is to evaluate the formal conditions separately of each block of data $\bar{X}$ and $\bar{Y}$. Then, with the independently normalization of equations (2. 53)(2. 54) we will obtain the 2-simplex equations normalizing by $H_{U_{\bar{X}}}$ and $H_{U_{\bar{Y}}}$. [3, 24]

$$1 = \Delta' H_{P_{\bar{X}}} + I'_{P_{\bar{X}\bar{Y}}} + H'_{P_{\bar{X}|\bar{Y}}}$$
$$0 \leq \Delta' H_{P_{\bar{X}}}, I'_{P_{\bar{X}\bar{Y}}}, H'_{P_{\bar{X}|\bar{Y}}} \leq 1 \quad (2.57)$$

$$1 = \Delta' H_{P_{\bar{Y}}} + I'_{P_{\bar{X}\bar{Y}}} + H'_{P_{\bar{Y}|\bar{X}}}$$
$$0 \leq \Delta' H_{P_{\bar{Y}}}, I'_{P_{\bar{X}\bar{Y}}}, H'_{P_{\bar{Y}|\bar{X}}} \leq 1 \quad (2.58)$$

Note that we will overlap 2 different diagrams on the same triangle calling this representation as the *split Channel Multivariate Entropy Triangle* as represented in the Figure 11. As shown in [24] using the Theorem 1 mentioned, on the aggregated CMET but considering only one block at a time, for instance the block $\bar{X}$ (The $\bar{Y}$ case is the same scenario):

• The lower side of the triangle with $I'_{P_{\bar{X}\bar{Y}}} = 0$, is interpreted as before in the aggregated case (marginal independence). Then $F(P_{\bar{X}}) = [\cdot, 0, \cdot]$

• The right side is the location of the partitioned joint distribution whose $\bar{X}$ block is totally described by $\bar{Y}$ meaning $H_{P_{\bar{X}|\bar{Y}}} 0$ $F(P_{\bar{X}}) = [\cdot, \cdot, 0]$

• The left side with $\Delta H'_{P_{\bar{X}}} = 0$, is the locus of partitioned distribution with uniform marginals $F(P_{\bar{X}}) = [0, \cdot, \cdot]$

2.3 Use Cases

From [1] introduction, for the performance analysis of a supervised classification tasks, we can highlight some mechanisms as the Receiver Operating Characteristic (ROC) curve as a good visual characterization for binary classification, but it´s generalization to higher input and output set cardinals is not effective. The same case occurs with the Area under the curve (AUC) of a ROC, but we will face the same problem when dealing with a high number of classes.

Therefore, in this section we will demonstrate how the entropic triangle can help us evaluating the performance of the scenarios present in the supervised classification seen in section 2.2. Moreover, we will provide a graphic solution using the tools implemented in real cases scenarios using a set of classification datasets extracted from UCI[3] , whose number of classes, features and observations are listed in Table 1

[3] https://archive.ics.uci.edu/ml/datasets.html

2.3.1 Multiclass classification evaluation (CBET)

As mentioned in section 2.2.1.5, the contingency matrix can be perceived as the joint distribution $P_{K\hat{K}}$ between random variables. Then, using the balance equations defined for the Channel Bivariate case and the consequent *Channel Bivariate Entropy Triangle*, we can reach a more complete understating of how the classifier is performing.

From the representation of the *de Finetti* diagram, the sides and vertices of the triangle depicts classifier performance-related qualities. So, we can define the most relevant areas as mentioned in [4]:

- The lower side represent the geometric region where P_X and P_Y are independent, meaning that there is no mutual information between the distributions, *so the closer the classifier is to this zone, the more unreliable the classifier decision are.*
- The left side represents the region where the marginals of P_{XY} are uniform, which means that the classifiers are not trained with overrepresented classes and therefore cannot specialize in any of them. In other words, they operate on perfectly balanced distributions, so *the closer to the left side, the more generic the classifier is* (more difficult the classification problem).
- Finally, the right-side locus represents the situation where the marginals are equal $P_X = P_Y$, meaning there is not variation of information, hence perfect accuracy. This characterizes *classifiers which transfer as much information as they can*.

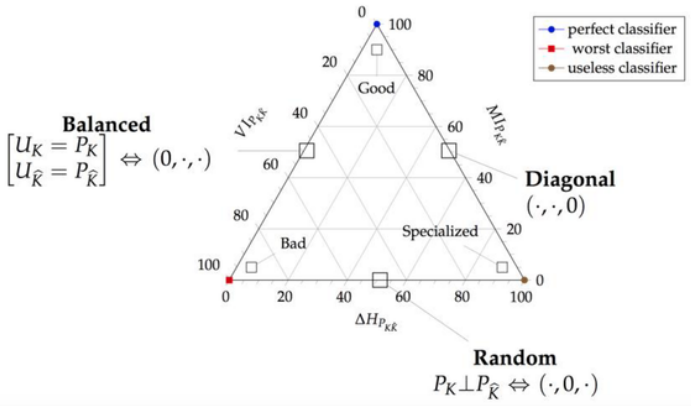


Figure 12. Schematic entropy triangle showing interpretable zones and extreme cases for classifiers. Reprinted from [1]

We can now move to the evaluation of the vertices of the triangle, which will represent the ideal, prototypical classifier [4]:

- The upper vertex represents *optimal classifiers* with the highest information transfer from input to output $F_{XY} = [0,1,0]$
- The left vertex exemplifies an *inaccurate classifier*, with low information-transfer $F_{XY} = [0,0,1]$
- The right vertex represents *underperforming classifiers*, with low information transfer $F_{XY} = [0,0,1]$

Once the main regions are defined, we are going to display a real example in which we can actually see the performance of some real classifiers using in this case as an experiment the *Knn-classifier* and the *Naïve-Bayes classifier.*

2.3.1.1 A Knn-Classifier example

We are going to evaluate, based on the previous regions we defined, the performance of the *knn* classifiers, observing the variation of the information measures for the joint distributions P_{XY} (as $P_{K\bar{K}}$) while varying the number of neighbours k, of both databases.

As we can see in Figure 13 , the classifier performance points as soon as we vary the number of neighbours, falls on the same left-axis parallel line, meaning that the classifiers not arbitrarily specializing. So, they are *general classifiers* which is a good characteristic because they hadn´t been over trained. In addition, the mutual information $MI_{P_{XY}}$ is between always in the *60% - 70%* range, so the classifier in general is performing over a good information transmission condition. Finally, the number of neighbours needed for the best classification performance of the *Breast Cancer* dataset is k = 5, and as we increase the number of neighbours k worsens the performance.

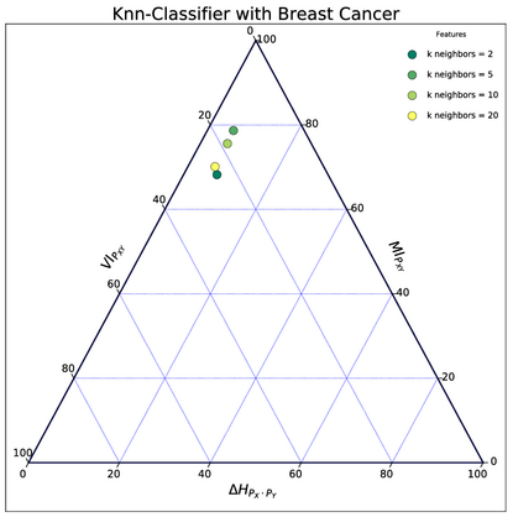


Figure 13. *Knn* CBET with Breast Cancer dataset

In the case of Figure 14 we can realize that the performance of the *knn* varying the number of neighbours is exactly the same. This actually happened because of the balanced

characteristic of the *Iris* dataset ($\Delta H_{P_{XY}} \approx 0\%$). The information transferred is around the maximum ($MI_{P_{XY}} \approx 95\%$) so we can realize that the classifiers performance is almost optimum ($F_{XY} = [0,1,0]$)

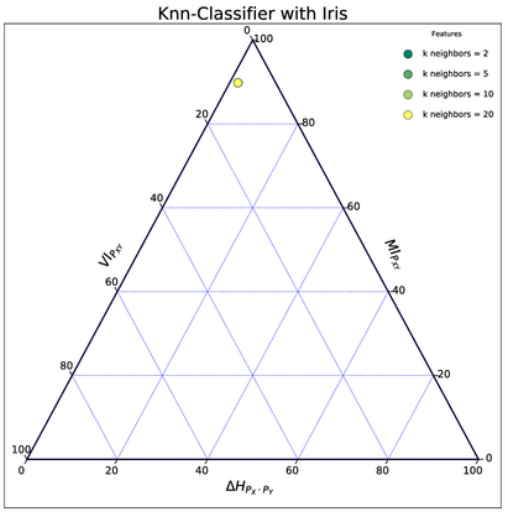


Figure 14. *Knn* CBET Iris dataset

Once evaluated the performance of the classifiers with all the data, we are going to check how the information varies depending on the size of the training set we use to model the classifier with $k = 5$. First, in Figure 15 we can appreciate with the Breast Cancer dataset that reducing the size of the training set worsens the classifier, as the mutual information shared by the X and Y variables is lower.

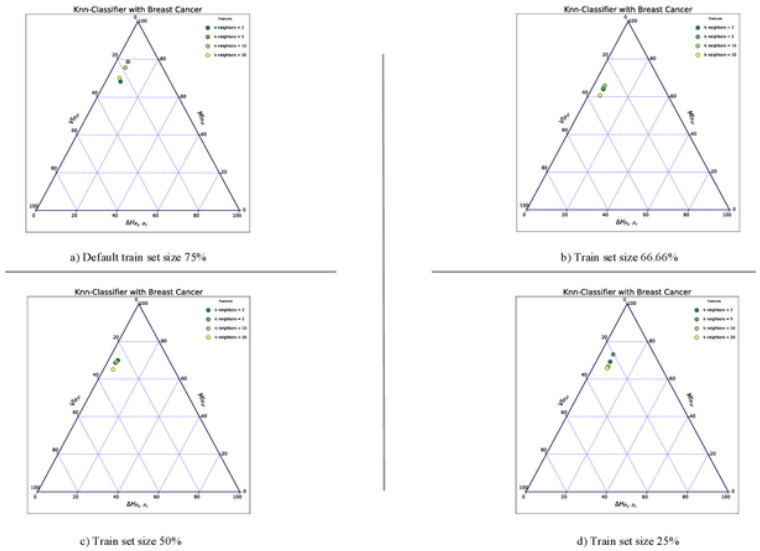


Figure 15. Knn CBETs varying the Breast cancer Train/Test size

The same scenario happens when we evaluate the performance of the Iris dataset in a knn-classification task shifting the train data set size shown in Figure 16 .The ideal case where the classifiers were working, is worsened as the mutual information is lower. This is straightforward, because of the fact that the *knn – classifier*, is directly affected by the train/set size, as the higher size for the train set, the distinction between the class areas will be higher.

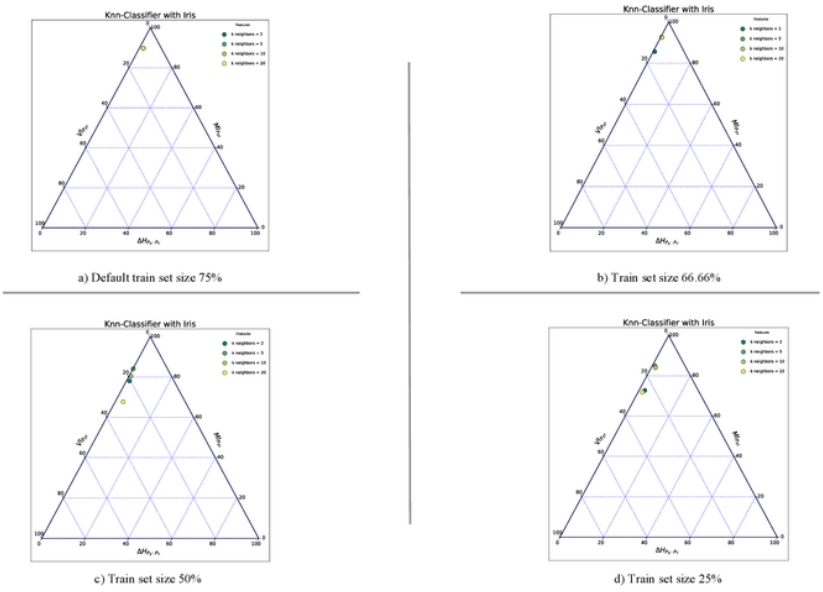


Figure 16. Knn CBETs varying the Iris Train/Test size

2.3.1.2 Naïve-Bayes Classifier

We will follow the same procedure as with the *Knn*, in the first scenario we are going to evaluate the performance of the Naïve-Bayes Classifier using the default train set size provided by *sklearn* [4] and then we will see how it performs decreasing the set size

[4] Scikit-learn package from Python: http://scikit-learn.org/stable/

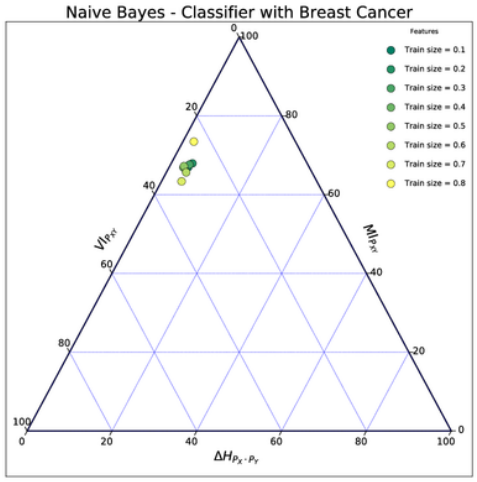


Figure 17. Naive Bayes Classifier CBET representation with Breast Cancer

From Figure 17 we can observe the slight decrease of the mutual information of the joint distributions with the decrease of the train set size, at the same time as the joint distribution of the variables shows a higher information variation. We can appreciate that the marginals of the joint distribution are mostly uniform, meaning that the classifier is mostly generic ($\Delta H_{P_{XY}} \approx 5\%$).

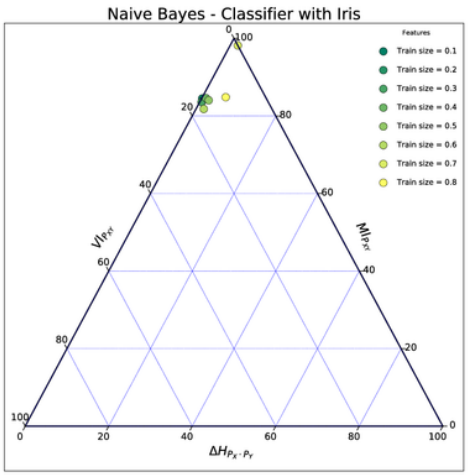


Figure 18. Naive Bayes Classifier CBET representation with Iris

Figure 18 display the representation of the Naïve Bayes classificatory but with the Iris dataset. We can see in this particular case that when we use *70 %* of the dataset for training the model, we obtain a perfect classifier as $MI_{P_{XY}} \approx 98\%$, $VI_{P_{XY}} = 0\%$. However, with other measures of the sets size, we obtain similar results when we varied the number k of the *knn classifier*. This means that the variation proposed of the train model, doesn´t affect the performance in nearly all the situations when using Iris, because it is a balanced database.

2.3.2 Using the SMET for Analyzing Data Sources

This tool can help us to analyse whether the data we are using in some supervised or unsupervised classification task is good enough for solving it. We can analyse how different dataset features contribute to the aggregate information, and in other way we can individually encode the information within the class using the equations used in section 2.2.2.

Analysing the feature´s individual scenario, we can use the split balance equation for the multivariate source case of a dataset class K from equation (2. 50) [2]:

$$1 = \Delta H'_{P_K} + M'_{P_K} + H'_{P_{K_i|K^C}} \quad 1 \leq i \leq n$$
$$0 \leq \Delta H'_{P_K}, M'_{P_K}, H'_{P_{K_i|K^C}} \leq 1 \tag{2. 59}$$

Then we can interpret some regions as in the previous case for characterizing the dataset features used in a machine learning task as mentioned in [2]:

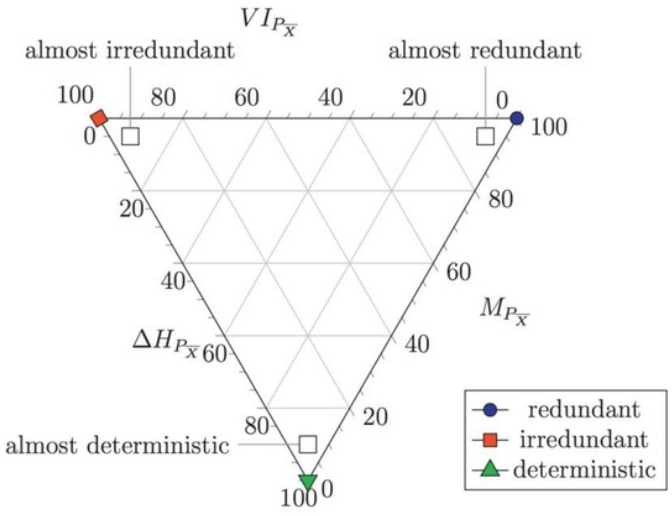


Figure 19. Representation of the SMET with the maximum conditions (from [2])

- ΔH_{P_K} quantifies how unbalanced the class variable is. From [2] we know that the greater this quantity is, the easier determining it with a possible encoding. In the other hand the lower the ΔH_{P_K}, the easier K is to determine from any feature.
- M_{P_K} quantifies the information of K provided by another variables K^C [2]. The higher M'_{P_K} the more quantity of information is captured by the features and the easier the classification should be.

- $H_{P_{K|K^C}}$ is the remanent entropy [2] in the class variable not capture by the features. The higher the more difficult the classification is.

The position in the triangle provides information as to the difficulty of the classification task, prior to any inference of a classifier. Figures 20, 21 and 22 shows the representation of the *source multivariate entropy triangle* of a series of databases for their characterization depending of their information measures. The datasets evaluated will be the presented in Table 1

Table 1. Data Sources used for the SMET evaluation

Dataset Name	Instances	$\lvert\bar{X}\rvert$	|Support(K)|
Arthritis	84	3	3
Iris	150	4	3
Breast Cancer	699	9	2

Using Table 1 datasets we are going to depict and analyse the information measures for each of the dataset's classes differentiating from their positions, some types of datasets that can be represented:

- The Figure 20 represents the Arthritis dataset. We can see that the features lay on a region which means that it is almost a balanced dataset $\Delta H_{P_K} \approx 0\%$ [2] with a lot of redundant information $H'_{P_{K_i|K^C}} \geq 60\%$) not captured by the features. This is one of the most difficult tasks for obtaining efficiency.

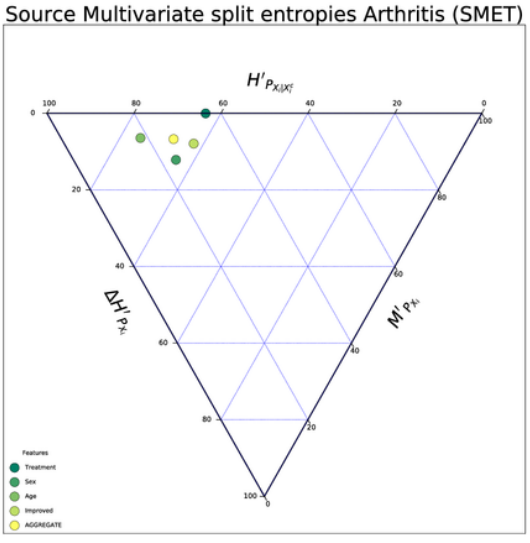


Figure 20. Arthritis Source Multivariate Entropy Triangle

- Then using the Iris dataset in Figure 21, we can see a balanced dataset ($\Delta H_{P_{Ki}} \leq 20\%$) with almost no redundant information, meaning that it is a good database from a classification task [2], as the mutual information shared by the classes is high ($M'_{P_K} \geq 80\%$)

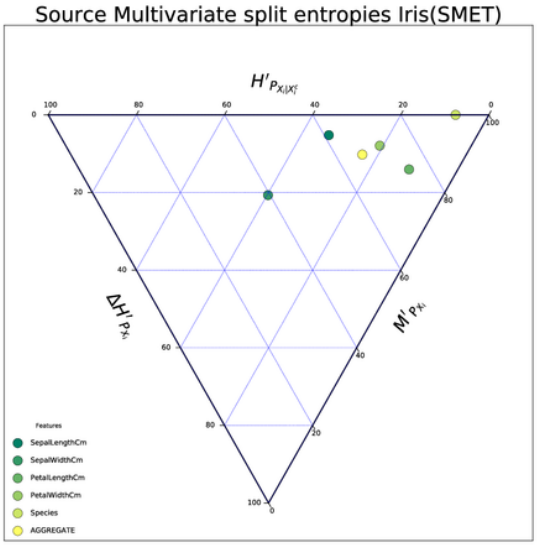


Figure 21. Iris Source Multivariate Entropy Triangle (SMET)

- Finally, in the Breast Cancer dataset represented in Figure 22 we can see a dataset with no redundant information. ($H_{P_{K|K^C}} \leq 10\%$). As mentioned in [2], these types of databases can be solved within a supervised classification task, but not perfectly:

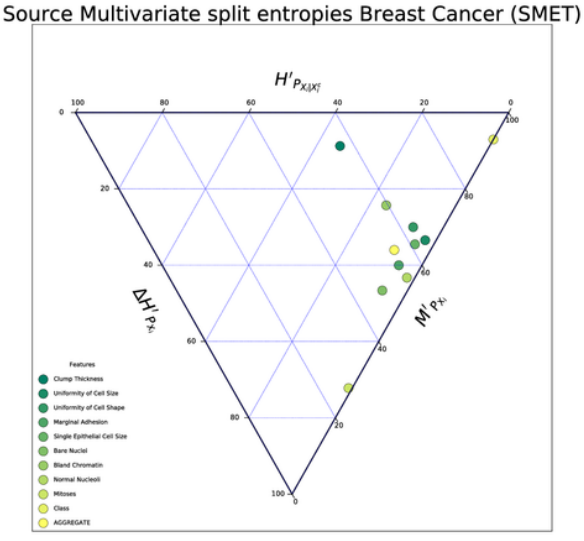


Figure 22. Breast Cancer Source Multivariate Entropy Triangle SMET

2.3.3 Analysis of data transformation (CMET)

The aggregate Channel Multivariate Entropy Triangle CMET [3] is an exploratory tool to assess the efficiency of multivariate channels. We also present a practical contribution in the application of these balance equations and diagrams to the assessment of information transfer efficiency for PCA as a feature transformation and selection procedure in machine learning applications.

Figure 23. Representation of a data transformation scenario for the evaluation of the CMET (from [24])

Data Transformation is used in machine learning the transform available data for instance, observations in a dataset $\overline{X}$, into another data with better characteristics, which will be represented as the transformed feature vector $\overline{Y}$ [3]. Then, the objective in a classification scenario as presented in Figure 23 is to explore different transformations of the observations for a better posterior classification. In our case, as an instance of the mentioned transformations, we will just evaluate the Principal Component Analysis (*PCA*)[5] which are usually used both for feature transformation and dimensionality reductions. In fact, we are going to evaluate for the performance assessment employing both methods as we will firstly transform the random vector $\overline{X}$ into a $\overline{Y}$ ranked features, and then, we will apply features selection selecting subsets $\overline{Y}_j$ spanning form the first-ranked to the *j-th* feature.

Hence, using the Iris database (cfr. Table 1) used in the previous sections and based on the study carried out in [3, 24] we are going to apply the PCA transformation. In Figure 24 we can highlight the crosses, and the triangles that will represent both the decomposition of the input's features $\overline{X}$ (2. 57), and the information decomposition of the transformed features $\overline{Y}_j$ using (2. 58)(2. 57), as well as the circles representing the aggregate decomposition of (2. 55). The transformed features $\overline{Y}_j$ depict several possible features sets that can be taken from $\overline{Y}$, by the selection of the first j features.

Figure 24 explores how the information in the whole database $\overline{X}$ is transported to different nested candidate feature sets $\overline{Y}$. Then we can notice that the $\overline{X}$ points falls on a left side parallel line meaning that the redundancy $\Delta H_{P_{\overline{X}}}$ is a constant value independently of the $\overline{Y}_j$ selected, which does not happen when we increase the number of features selected j as there is a monotonic increase of the average information transmitted $I'_{P_{\overline{XY}_j}}$. So, as we can see, the more output features selected, the higher absolute mutual information, but the lower relative mutual information, between input and output. [3, 24]

[5] ftp://statgen.ncsu.edu/pub/thorne/molevoclass/AtchleyOct19.pdf

Pursuing the analysis, we focus on the $\overline{Y_J}$ points which reflects an increase of the redundancy as soon as j increases selecting more features accompanied by a decrease of the mutual information $I'_{P_{\overline{XY_J}}}$.

Finally, while evaluating the points that represent $\overline{XY_J}$we can appreciate that a lopsided, inverted U pattern is shown, reaching its peak of transmitted information before j reaches its maximum as mentioned in [3, 24]. Meaning that if we look for a faithful representation, then transmission of information must be optimized, so in the case we take a subset of the first 3 features from the output $\overline{Y_J}$, the information transmitted maximum for this case.

To sum up, for a classification process, while applying feature transformation we should maximize the average mutual information accumulated by the transformed features. It is depicted in the Figure 24 which means that we can successfully asses the performance of a data transformation by using the information-theoretic tools presented in the entropy triangle.

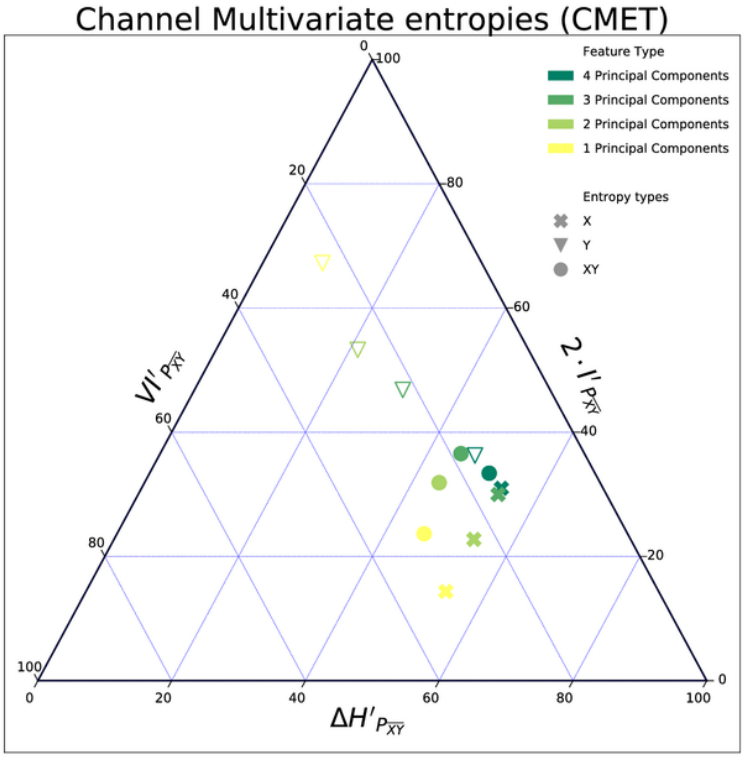


Figure 24. Representation of the CMET for the evaluation of PCA features transformation using the iris data set

2.4 Related work

2.4.1 Prior Implementations

There is a set of developed packages implementing the concepts seen from [1, 2, 3] but for other programming languages. First, there is a Matlab package[6] created for the implementation of the *bivariate channel Entropy Triangle*, and other entropic measures that out of the scope of this Project such as the entropy-modulated accuracy (EMA) or the normalized information transfer function (NIT) presented in [1]. This software was just a proof-of-concept, but we can extract interesting insights for the code development of the *bivariate channel entropy triangle* even though Python and Matlab are two different languages.

In addition, we can find also a Weka plug-in[7] created by [1] intended to expand the information theoretic tools developed in the previous Matlab package, to a wider community for its use. Nevertheless, the functionalities represented fall short as it is just an extension of the previous package to the Weka community.

Furthermore, the Matlab package was a starting point for [1] in the definition of the R package[8] for the extension of the *bivariate Entropy triangle*, for representing all possible cases shown in the theory [3, 2] extending the functionalities of the previous packages to a new level in which we can represent as well, the Source Multivariate Entropy Triangle (SMET) and the Channel Multivariate Entropy Triangle (CMET) thanks to the development of the balance equations for each case.

However, the philosophy of R caters the researchers but not the occasional user, for this reason we will implement the Python package for the aim of making the technique known to a wider audience and to lower the technological barrier for using it.

2.5 Objectives and Quality Requirements

Now that we have analyzed all the theoretical concepts necessary to understand what is behind the balance equations and therefore the entropic triangle, we will divide the main objective of the project into small goals that together will help us to execute the project correctly. Therefore, since a software has to be implemented for the entropy triangle representation, we will encapsulate the current objectives according to the functionalities to be developed for the project compliance.

The software must fulfill the following functional requirements:

1. The package must supply Python, with the information-theoretic evaluation metrics seen and described in [1, 2, 3] through:

---

[6] https://es.mathworks.com/matlabcentral/fileexchange/30914-entropy-triangle
[7] http://apastor.github.io/entropy-triangle-weka-package/
[8] https://github.com/FJValverde/entropies

a) Correct calculation of the balance equations calculation for the *bivariate case*.

b) Correct calculation of the balance equations calculation for the *multivariate source case.*

c) Correct calculation of the balance equations calculation for the *multivariate joint case*

2. The package must provide Python with the *Entropy Triangles* described in [1, 2, 3]

   a) Visualization of the *Channel Bivariate Entropy triangle*

   b) Visualization of the *Source Multivariate Entropy triangle*

   c) Visualization of the *Source Multivariate Entropy triangle*

3. Furthermore, the package must be distributable for its use within the community

   a) Appropriate software *packaging for distribution*

   b) Uploaded to a server for *open source software*.

Moreover, the library should accomplish some Quality requirements:

1. From a user´s perspective, the package must be *accessible and installable* and *usable*. Hence, we should provide an installation guide and a user's manual.
2. From the software perspective, the package has to be as light as possible, and it should exploit the libraries already created for a *functionality reuse*.

3. SOFTWARE IMPLEMENTATION

In this chapter, we give a brief description of the features used for the definition of the entropy triangle package for Python as well as the package management system used for the development of the code, and the requirements needed to make the package distributable to the community.

3.1 System Architecture

Python is an open-source, portable, object-oriented software which is interpreted, meaning that there is no separation between compilation and execution. All this combined with the fact that it is a very simple language to learn, facilitates the constant increase of the user's community and the available resources numbers. These resources can be programmed in one of the two principal Python versions for code developing, the 2.x and the 3.x. In this project we will use the Python 3 generation as it is the one which will stand in the future *(version 3.6.5)*

The resources available for Python are called *libraries* which are a group of structured modules (Python files) contained within a package. Each program module defines a series of functionalities that can depend on other modules. So, for the project definition, we will first define properly the packaging system and the libraries used for the implementation as well as the requisites needed for the creation and distribution.

3.1.1 Python Packaging System

Python packaging system intends to follow the Linux-based systems, treating software as a collection of small self-contained units, called modules. A *module* is a single python file which can have a set of functions, classes or variables defined and implemented (or exported from other packages), whereas a *package* is a directory which can contain a collection of modules or more packages [25].

Python packages must store a `__init__.py` file required to make python treat the directory as containing a package. This file can be empty but it´s often used to perform a setup such as importing classes, functions into the package level, etc. [26]. This will be the unique mandatory file for the creation of a basic package. For package deployment, we will need to add a set of specific files mentioned in the following sections. Figure 25 shows an example of a basic package containing modules, the `__init__.py` required file used for the initialization and sub packages

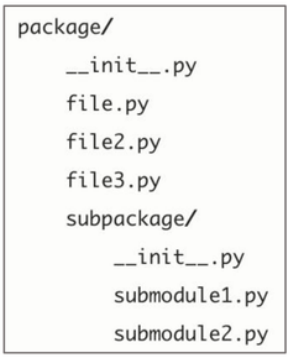


Figure 25. Sample python package (from [26])

Note that modules within a package can import functionalities from other modules or sub packages. In summary, the package creation simplicity has promoted the development of a huge number of different functionalities of Python packages, which, in addition to the ease of accessing to them, has created a big and active community of users. This variety of packages will be helpful for code development as it will allow us to take advantage of previously developed functionalities for the creation of the project's package.

3.1.2 Packages Used

The variety of packages available allows us to approach problems differently in each situation. Therefore, it is necessary to select the most useful packages for each case. From the theoretical background presented in section 2, we will select and import some predefined libraries, that encapsulates the functionalities required for the production of the entropy triangle. First of all, we are going to work with different databases, so we are going to use Pandas package to handle the data:

- *Pandas* [27]: the library used for data manipulation and analysis. It offers some data structures and operations for handling numeric tables and numerical series.

Then, we need to calculate some entropic measurements, which entails mathematical calculations for the balance equation definitions, for that reason we will use some scientific packages as the following:

- *NumPy* [28]: provides multidimensional array objects, matrices and arrays objects and operations, and an assortment of routines for fast operations on arrays, including mathematical, logical, basic linear algebra, basic statistical operations, etc.
- *SciPy* [29]: is an ecosystem from mathematics, science and engineering. It contains modules for fields such as linear algebra, special functions and signal processing.
- *Scikit-learn* [30]: is a simple and efficient package for data mining and data analysis, featuring the main machine learning algorithms, such as classification, regression and clustering. It is built on the libraries mentioned NumPy, SciPy.

Finally, once we have calculated the entropic measurements we will need to plot the entropy triangle. There is a package pre-defined by Marc Harper for plotting ternary plots, it is called *ternary* and it´s used for ternary representations, but we will go deeper on this in following sections. The ternary package is built over the matplotlib library:

- *Matplotlib*: is a Python 2D plotting library which produces publication of quality figures as plots, scatterplots, histograms, etc. [31]

These are the main packages that will be imported during the code implementation. In next section we will explain deeply the functions and features used from each library.

3.1.3 Python Package Manager

The software development then will need the set of packages specified and also a Python interpreter, to compile and link the code in the version 3.6.5. For this purpose, we can create a personalized virtual environment to store all the libraries needed just for this project. So, the first option will be downloading all the packages and the interpreter manually into our system in binary form, as they are available in source code for every platform, but the main problem for this scenario is that we will have to handle the packages ourselves, which is not be the best.

However, there are tools available that will allow us to automatically generate several virtual environments with different versions of the packages in parallel and the systems can handle themselves these packages. These tools are known in Python as *Package managers*.

A *Package manager* is a utility intended to simplify the tasks of locating, installing, upgrading and removing Python packages from local or remote hosts [32]. We can find several managers to use during the Project creation that will simplify the development, for example:

- *Pip*: is the reference Package manager system used to install and manage software packages. The tool includes a command-line interface, which makes installing packages as easy as issuing one command, where you can also specify the version [33],

  ```
  pip install [package_name]
  ```

  Furthermore, you can install all files needed for a Project just by using a `requirements.txt` file,

  ```
  pip install -r requirements.txt
  ```

  The requirements.txt file includes all package names and the versions you want to download. Python versions higher than 3.4 include *pip* by default. So, in our case we may benefit from their use.

- *Conda:* is an open source cross-platform package management system and environment, providing installation, execution and updating of different packages along with their dependencies, and allowing users to easily install different packages software versions and any required libraries. Furthermore, *Conda* easily creates, saves, load and switches between environment in your local computer. In the case that we have an environment defined by *conda,* we can still use the pip install for the package's installation within the *conda* virtual environment, so the *conda* manager is compatible with pip [34, 35].

Therefore, considering the benefits of working with *conda*, we are going to use it to develop the code. During the project we will use the *conda* system included in a distribution called Anaconda that will provide us in addition to a package management system, more functionalities such as the ability to create virtual environments.

3.1.3.1 Anaconda Distribution

It is a free open source multiplatform distribution for package management and deployment, specially used for data science and machine learning applications. It contains a large number of packages at the same time as the mentioned *conda* package management system and the virtual environment management called Anaconda Navigator [36].

Therefore, to implement the code, a specific virtual environment will be created with the packages needed and the Python version selected. This environment will be created through *Anaconda navigator* [37] which is a desktop *graphical user interface* (GUI) that allows to launch applications and easily and manages efficiently *conda* packages, environments and channels without the use of the command-line.

This interface contains a series of applications that will serve as development environment. In our case we will work with *JupyterLab*, a server-client application that allows editing and running Python code via web browser. It can be executed on a local desktop without requiring nay internet access or can be installed on a remote server and accessed through the internet. This application is a *Jupyter Notebook*[9] extension that will be used to create vignettes with tool use cases.

The Figure 26 shows an outline of the composition of the anaconda distribution, showing in layers the functionalities disposed at the system.

[9] http://jupyter.org/

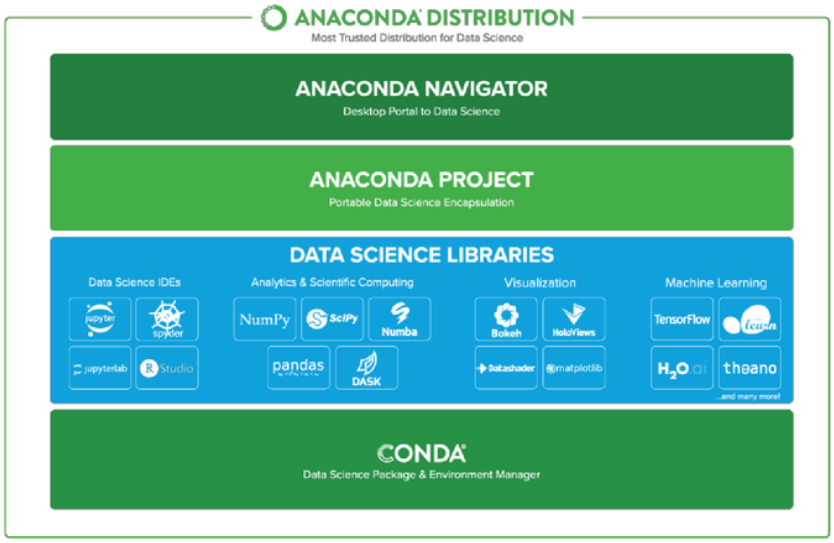


Figure 26. Anaconda Distribution composition (reprinted from [38])

In addition, while developing the code, we will use in parallel the Anaconda distribution for the software programming, and the *GitHub*[10] platform as a software online repository, for storage, but also for control version using *git*[11].

So, once we defined which is the package manager and the tools used for code development, we will go on implementing the entropy triangle package. Before going deeper with the implementation there will still be some considerations taken during package creation, as we have the goal of creating a distributable package. Bearing this in mind, we must know the indispensable package files requirements to be distributable. Therefore, once everything is set, we will upload the *Pypi* package, which is the main repository used by the Python community.

3.1.4 Python Package Index (PyPI)

One of our goals is to distribute the library among the community, therefore, we will need to upload the package created to a public repository. This will allow users to access it and seize the functionalities implemented. Then, according to the available systems, we have different platform options where we can store our package, as the *Anaconda cloud*, or the *conda-forge community* but the most powerful and used one is *PyPI*.

Python Package Index[12] *(PyPI)* is a third-party software repository for Python where you can find any package uploaded by the community for its use. Thus, our objective will be

[10] https://github.com/Jaimedlrm/entropytriangle
[11] version control software
[12] https://pypi.org/

to first define the whole package, and then adapt it in order to be distributed to the community, and finally upgrade it to a repository [39].

In next section we will see some mandatory requirements needed for configuring, packaging and distributing software.

3.1.4.1 PyPI File Requirements

Uploading libraries to PyPI requires a set of files for the configuration of the package. We will define the needed modules as mentioned in [40], that are going to be implemented for the packages *PyPI* upgrade. The modules will be used for the creation of built distributions versions that will be uploaded to the platform. The needed files will be the following ones:

- `setup.py`: Root directory file where aspects of the project are configured. This file contains the global `setup()` function stored in the `setuptools` module , whose arguments will be the specific details of the project. There are a bunch of arguments as the name, version, python version, author, license, etc. One of the main arguments of the function are `install_requires`, which is used for specifying the dependencies that the project minimally needs to run, and `python_requires` to specify the python version your project work with. In [40] the arguments that can be entered in the function are shown.

- `setup.cfg`: File that contains default options for setup.py commands. It is used to configure the behaviour of the various setup commands for your project. The supported commands can be listed using the following request,

```
setup.py --help-commands
```

- `Readme.md`: File covering the goal of the project, which can be defined as the `long_description` parameter in the `setup.py`

- `Manifest.in`: It is required when you need additional files that are not included in a source distribution

- `LICENSE`: Document detailing the terms of distribution of the project that every package should include. Although there are different licenses, we will choose the *MIT License* for this project, as it allows users to modify the code that we are going to create, for future developments.

- `Requirements.txt`: Document specifying packages dependencies. The setup.py module will read this file to pass as arguments `install_requires` of function `setup()` the specified dependencies

3.1.4.2 PyPI Package Upgrading

Once all the files specified in the previous section have been created, we are going to package and upload the project so that it can be installed from *PyPI*. We are going to represent, as a tutorial the packaging of the `entropytriangle` project, which is represented in the Figure 27. Therefore, we will detail the necessary steps to package and uploading the project.

Figure 27. `entropytriangle` package before the distribution

3.1.4.2.1 Packaging the project

38

In this section we are going to introduce the steps needed for the upload of the package to the PyPI repository explained in [40]. First of all, we will have to create a *source distribution*, which is a distribution format containing files and metadata and essential source files needed for the installation of the package [41].

Consequently, in the directory where this package is located we are going to execute the command:

```
python setup.py sdist
```

In the Figure 28 we can appreciate that the command creates some distribution files in the top of the image.

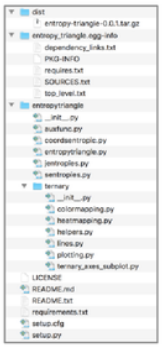


Figure 28. `entropytriangle` package after creating a *source distribution*

Afterwards, a *built distribution* [41] is going to be assembled using *Wheels*[13] which is a built package format for Python, that can be installed without needing to go through the "build" process.

There are two types of wheels depending whether the project is supported in Python 2 and Python 3, or just in one of the two versions. If the project is pure Python [40] and supports natively Python 2 and Python 3, then we are creating a Universal Wheel. However, as the entropy package is just supported by Python 3, thus we will create a *Pure Python wheel*.

Figure 29. Packaging for Python libraries structure (reprinted from [42])

If the *wheel l*ibrary is not installed, firstly we will download it using either the pip or the *conda* package manager.

```
pip install wheel
```

```
conda install wheel
```

Then for the creation of the pure python wheel that are *not universal* we are going to use this command:

```
python setup.py bdist_wheel
```


13 is a ZIP-format archive with specially formatted filename and the *.whl* extension

It will detect that the code is pure python, and build a wheel that is named such that it is usable on any python installation with the same major version as the version you used to build the wheel with [40].

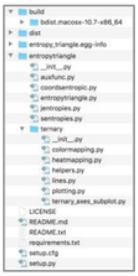


Figure 30. entropytriangle package with the wheel created (upper folder "build")

Now we have this set of files, we are ready to upload the project to the *PyPI* platform.

3.1.4.2.2 Uploading the project to PyPI

When we run the command to create the distribution, a new directory `dist/` is going to be created under the projects root directory. There we will find the distribution files to upload.

Firstly, for uploading the distribution library, we will need to create a *PyPI* user account in the website. Once we have an account, we will upload the distribution by using the package twine, which is a utility for interacting with *PyPI*, that offers a secure replacement for `setup.py` upload [43]:

```
pip install twine
```

Then the command for uploading the package to the repository of the created *PyPI* account will be:

```
twine upload dist/*
```

With this command our code will be uploaded to the repository and will be available to be downloaded for any user after entering your credentials from the *PyPI* account. However there are times that this command doesn´t work and we will need to create a file `~/.pypirc` which is a file containing the credentials and the *URL* of the repository [40]

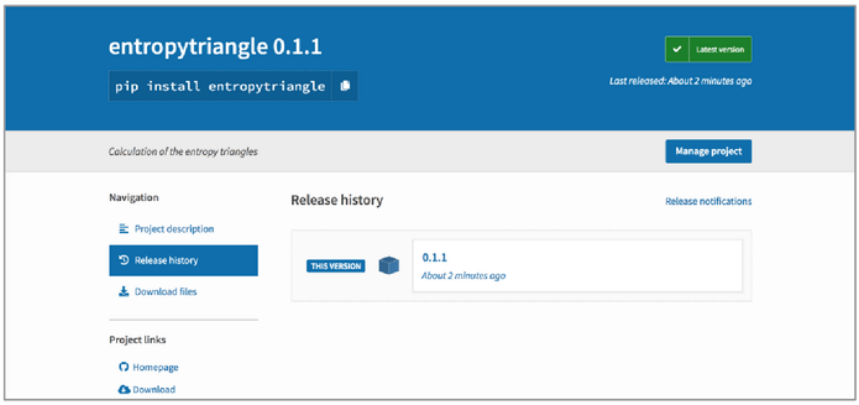


Figure 31. PyPI entropytriangle package repository[14]

In section 3.2 we will explain how to download the repository to be used. For later versions, if we upgrade the Project, the procedure will be just the same one, but updating as well the `setup.py` version argument. At the *PyPI* repository, then the upgrade package will appear as the latest version.

As we can see in Figure 31, the package is already available in the *PyPI* repository and is accessible and downloadable just by using the *pip* commands for the installation. In the case of working with the *conda* package manager, the pip command will also work as they are compatible, and it will appear on the display when typing `conda list`. However, the package will not work, because the *conda* packaging manager uses another way of packaging called recipes. So, in next section, we will try to provide with the packages to conda users by uploading the appropriate package to the Anaconda cloud.

3.1.4.3 PyPI required files as a conda package installation solution using Anaconda Cloud

Bearing in mind the fact that the *conda* users will not be able to install PyPI packages, we tried to define a solution for the installation of the `entropytriangle` using *condas*. We reach to the conclusion that there is a way for packaging and uploading the contents to the *Anaconda cloud*, which is a central repository working as PyPI.

The normal flow of a *conda* packaging process will be to create the *conda* recipe with the package metadata and configuration settings, and then upload it to the system. However, we are going to seize the required files we created in for the PyPI library uploading, to upload it to the *Anaconda Cloud* for distribution.

From [44] we can get insights into building a package without using a recipe. First, we will need to register in *Anaconda Cloud* for the repository creation. Once we are

[14] https://pypi.org/project/entropytriangle

registered, then in the command line we will need to login to anacondas using the following command

```
anaconda login
```

and typing the profile credentials. Then we will need to type in the command line at the `entropytriangle` package directory this the command:

```
python setup.py bdist_conda
```

It is a quick way to build packages without using *recipe*, but it has some limitations. This command will generate the distribution package and provide us with an instruction that we will execute to upload the library to *Anaconda Cloud*.

```
# If you want to upload this package to anaconda.org later, type:
#
# $ anaconda upload /Users/jaime.de.los.rios/anaconda3/conda-bld/osx-64/entropytriangle-0.0.1-py36_0.tar.bz2
```

Figure 32. Command line message for the upload to the Anaconda Cloud

If we execute the instruction, the package will be uploaded to the repository of the associated account that we have registered in the command line to this domain.

```
Using Anaconda API: https://api.anaconda.org
Using "Jaimedlrm" as upload username
Detecting file type...
File type is "conda"
Extracting package attributes for upload
Creating package "entropytriangle"
Creating release "0.0.1"
Uploading file "Jaimedlrm/entropytriangle/0.0.1/osx-64/entropytriangle-0.0.1-py36_0.tar.bz2"
 uploaded 47 of 47Kb: 100.00% ETA: 0.0 minutes
Upload complete
package located at:
https://anaconda.org/jaimedlrm/entropytriangle
```

Figure 33. Installation of the Python package in the Anaconda cloud

The package will be ready in the repository to be installed and used from conda. The command for the installation will also be included in the repository where it is stored.

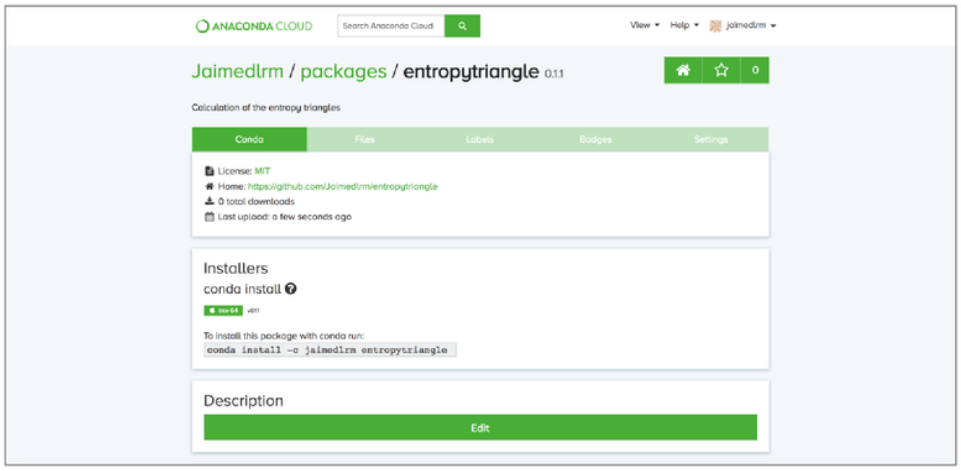


Figure 34. Anaconda Cloud `entropytriangle` repository[15]

So, at the end, both *PyPI* users working with the *pip* manager and *conda* users will be able to download and install the `entropytriangle` package. As mentioned in section 3.2 we will provide an installation guide which will help users in the process of downloading properly the package.

3.2 System Design

3.2.1 Package Structure

The package design will rely on the requirements mentioned in section 3.1, whereupon the possibilities have been evaluated to generate the components of the package that fit best with the proposed project goals. For that reason, the package will follow a modular design, importing and reusing the functionalities developed in a specific module.

From the beginning, we conclude that the best way to carry out the implementation was to divide it into two parts corresponding with the two needed functionalities. The first part is the development of the evaluation metrics needed to develop the balance equations for the different cases seen in section 2, and the second is the creation of the plots necessary to build the entropy triangle. There will be implemented in both auxiliary modules, which will contain general functions used in other scenarios. This approach allowed us to first fix and evaluate the result in each specific point and finally unify it to form the final package.

[15] https://anaconda.org/jaimedlrm/entropytriangle

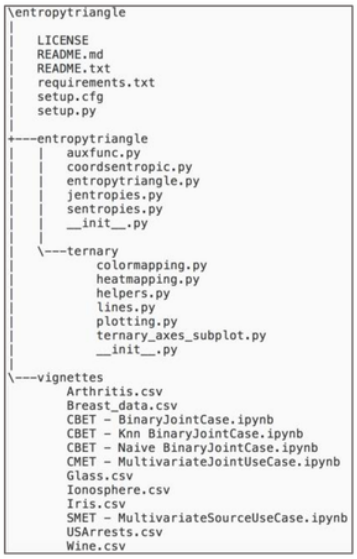


Figure 35. `entropytriangle` package structure

Besides, the package is composed of the necessary files so that it can be distributed, according to the standards fixed by *PyPI* seen in section 3.1.4.1, with the purpose of uploading it to the *PyPI* platform and be accessible through package managers. Finally, a folder of vignettes was created to store real cases uses available in the GitHub repository and attached at the end of this document. Notice that the vignettes only will be available in the GitHub repository, not in the download package as the weight of the vignettes folder increases the overall package weighting. The Figure 35 shows the final structure of the package, in which you can identify modules exposed in next section.

22

3.2.2 Evaluation Metrics Modules

These modules encapsulate all the necessary functionalities to calculate both balance equations in the case of the channel and the source. We can clearly differentiate the scenarios in which we can use the equations (*CBET, CMET* and *SMET*), therefore, it will be necessary to define the data structures that we are going to use in each case, since, depending on these, the procedure will vary.

Three different modules will be implemented. The first one, `jentropies.py` will be used to evaluate the joint entropies metrics in the cases where it is necessary to represent channel schemes. The second one, `sentropies.py` will have as purpose calculating the evaluating the entropies values in the source schemes. And to compute the necessary values in both modules, we will implement a third one, `auxfunc.py` module which will provide the necessary functions for the calculation of these entropies. Therefore, we will be able to analyze all possible situation depending on the presented scheme. However, even though different scenarios are represented, we will try to unify the outputs of the functions, for simplifying the plotting depiction.

The entropy calculation functions from `sentropies.py` and `jentropies.py` used to calculate the proportions of the joint distributions have as output a Pandas data frame, whose name columns will be referencing the entropy metrics of the balancing equation they represent. This will serve to know at all times the variables that are represented in the triangle.

auxfunc.py

At the auxfunc.py module we implemented all the auxiliary functions necessary to correctly calculate the balancing measures in the *sentropies* and *jentropies* modules. This file will serve as a repository from where the other created modules will reuse the functions they need. In the module it defines the following functions:

- `discretization(df,nbins)`: Function created to discretize a database, according to a specific number of bins. The reason for its creation is the impossibility of the `pandas.cut()` function to discretize strings. Therefore, the designed function will take a Pandas Data Frame, and will segment column by column the values in a series of bins applying a cast to pass from a numeric variable or string to a categorical variable. For the discretization of a string column, the `LabelEncoder` class of the library `scikit-learn.preprocessing` will used.

  This function will be assimilated to the `discretize`[16] function created for *R* in the `infotheo` library [45]. The bins in which each column will be segmented according to the R function will be equal to:

  $$nbins = nrows(DataFrame)^{1/3} \tag{3.1}$$

  So, the output of the function will be a data frame with the columns discretized in *n* equal width *bins*.

- `sjoin()`: Function created for merging all the entries of a data frame into one pandas series. Used within the entropy calculation and the *conditional entropy calculation*. The output will be a data frame with all the columns merged.

- `ent()`: Function created to calculate the entropy of a Pandas series structure and the joint entropy of a data frame. The operation is done calculating the distribution of the inputs and applying the formula seen in the equation () using the function `scipy.entropy()` function that only works with probability distributions. The output will be the entropy of the element passed as argument.

- `condentropy()`: Function created to calculate the conditional entropy of a data frame input $\bar{X}$ with respect to an input $\bar{Y}$, applying the chain rule saw in (2. 4) but with random vectors

[16] https://rdrr.io/cran/infotheo/man/discretize.html

$$H_{P_{\overline{X}|\overline{Y}}} = H_{P_{\overline{X}\overline{Y}}} - H_{P_{\overline{X}}} \quad (3.2)$$

jentropies.py

Module defined for the calculation of the entropic variables for binary and multivariable channels cases. As there are different scenarios, we will proceed in a different way. For the case in which a multi-class classification task is evaluated, the data structure used will be arrays for the entropic values of the *Channel Bivariate balance equations* calculation (section 2.2.1.1). On the other hand, for the case in which a feature transformation is evaluated in a machine learning process (section 2.2.3.1), the data structure used will be the Data Frame that offers pandas to calculate the values of the *Channel Multivariate balance equations*.

- `jentropies_binary(c_matrix,base)`: Evaluates the information metrics of a multi-class classification scenario. The function takes as input a contingency matrix in 2-d array format and the base used in the calculation of the function `scipy.entropy()`, for the calculation the entropic values of the variables k and $\hat{k}$ of a multi-label classification task. The function:
  - Calculates Marginal Entropies H_{P_K}, $H_{P_{\hat{K}}}$and Uniform Entropies H_{U_K}, $H_{U_{\hat{K}}}$
  - Determine, the *joint entropy* of the distributions $H_{P_{K\hat{K}}}$ and the *conditional entropies*, $H_{P_{K|\hat{K}}}$ and, $H_{P_{\hat{K}|K}}$
  - Computes the *split entropy factors* $\Delta H_{P_K}, \Delta H_{P_{\hat{K}}}, MI_{P_{K\hat{K}}}, VI_{\hat{k}}$ with the data already calculated using equations (2. 11)(2. 13)(2. 14) and then the joint ones just applying addition.
  - Creates the output data frame `edf` where the first entries represent the split entropy fractions values and the last one the joint entropy values.

Table 2. Entropy Data Frame of a K-nn multiclass classificatory (using Breast Cancer)

Type	H_U2	H_P2	DeltaH_P2	M_P2	VI_P2
X	1.0	0.942683	0.057317	0.786867	0.155816
Y	1.0	0.942683	0.057317	0.786867	0.155816
XY	2.0	1.885366	0.114634	1.573734	0.311632

- `jentropies(df1,df2,base)`: Evaluates the information metrics of a feature transformation scenario. Calculates through two data frames representing the random vectors $\overline{X}$ and $\overline{Y}$, a dataframe with the first entries representing the split entropy values and the last one the joint entropy proportions for the evaluation. This function:

- Discretize both data frames (not discretized before) using `discretize()` function from `auxfunc.py`
- Calculate the metrics $H_{P_{\bar{X}}}$, $H_{P_{\bar{Y}}}$ and $H_{U_{\bar{X}}}$, $H_{U_{\bar{Y}}}$
- Determines the variation of information $VI_{P_{\overline{XY}}}$ from the `condentropy()` function from `auxfunc.py` (2. 23)
- Computes the split entropy factors $\Delta H_{P_{\bar{X}}}$, $\Delta H_{P_{\bar{Y}}}$, $MI_{P_{\overline{XY}}}$ with the data already calculated using equations (2. 24)(2. 25)(2. 32) and then the aggregate entropy just applying addition.
- Creates the output Data Frame `edf` where the first entries represent the split entropy fractions values and the last one the joint entropy values.

Table 3. Entropy Data Frame from a *PCA Feature Transformation* (using Iris)

Type	H_U	H_P	DeltaH_P	M_P	VI_P
X	10.872675	5.021737	5.850938	4.182282	0.839454
Y	9.287712	6.057566	3.230146	4.182282	1.875284
XY	20.160387	11.079303	9.081084	8.364564	2.714739

sentropies.py

Module define for calculating entropic variables in the case of multivariable source, with the aim of identifying the performance of a database in a machine learning process according to the information of its variables. Then, we will depict the multivariate source balance equations through a function whose arguments will be the data frame representing the source wanted to evaluate.

- `sentropies(df,type,base)`: Calculates from a data frame representing some data source of a machine learning process, the entropy values of both joint and individual variable distribution. The function allows us to calculate the entropy variables whether the user wants to measure the mutual entropy as the total correlation or just the dual correlation by introducing the argument `type` to the function. The output will be the `edf` data frame used in the channel case with the columns representing the entropy variables.
  - Discretization of the data frame (not discretized before) using `discretize()` function from `auxfunc.py`
  - Calculate the metrics $H_{P_{\bar{X}}}$ and $H_{U_{\bar{X}}}$
  - Determines the variation of information $VI_{P_{\bar{X}}}$ from the `condentropy()` function from `auxfunc.py`

- Computes the split entropy factors $\Delta H_{P_{X_i}}$, $M'_{P_{\bar{X}}}$, $H_{P_{X_i|X_i^c}}$ with the data already calculated using equations (2. 44)(2. 45)(2. 46) and then the aggregated values just applying addition.
- Creates the output Data Frame `edf` where the first entries represent the split entropy fractions values and the last one the joint entropy values.

Table 4. Entropy Data Frame of the Iris Data set evaluation

Name	H_Uxi	H_Pxi	DeltaH_Pxi	M_Pxi	VI_Pxi
SepalLengthCm	2.321928	2.200620	0.121308	1.417675	0.782945
SepalWidthCm	2.321928	1.841723	0.480205	0.917768	0.923955
PetalLengthCm	2.321928	1.995571	0.326357	1.738118	0.257453
PetalWidthCm	2.321928	2.137460	0.184468	1.654826	0.482635
Species	1.584963	1.584963	0.000000	1.465241	0.119721
AGGREGATE	10.872675	9.760337	1.112338	7.193628	2.566709

The values at the final entropy data frame `edf` are not normalized yet, this will be done at the plotting modules while calculating the ternary coordinates.

3.2.3 Ternary Package

Focusing on the visualization part, we observed that the matplotlib library has not implemented the possibility of representing ternary diagrams, so we proposed 2 solutions to the problem, the first was to create a ternary geometry, which would take a lot of effort and time. The second solution proposed was the use of a package already implemented in the community to save us time. Finally, we decided to use the *python-ternary* package created by Marc Harper [46] available on the GitHub[17] . The library will be used to plot ternary diagrams in 2-dimensional simplex projected onto a two-dimensional plane on matplotlib. This library is under the MIT license[18], which means that its use is free, and in addition we make modifications on it for our purpose, for the features that suit us.

In fact, during the Source triangle implementation, we encountered the problem of the diagram inversion to represent the inverted triangle. Thus, taking advantage of the fact that the package has a MIT license, some adjustments were made in the classes defined to solve the problem. Some of the modules will not be used, and therefore we will not mention them during the description of the package.

First of all, we tried to modify the code, but since we didn´t find a solution to plot from the same class the inverted triangle and the normal triangle with some modifications, we decided to duplicate the library and have a package for the normal triangle and another for the inverted triangle, a bad practice, clearly. Finally, after some extensive analysis the

[17] https://github.com/marcharper/python-ternary
[18] https://opensource.org/licenses/MIT

problem was solved by introducing functions and parameters that allow us to plot successfully from the same class the two situations.

To sum up, the final package consists of a series of modules that implement a series of plotting and coordinate definition functionalities, which wrapped into `ternary_axes_subplot.py` for the characterization of the `TernaryAxesSubplot` class that manages the matplotlib axes, the scale and the boundary scale.

helpers.py

Group of auxiliary functions to project the points to the simplex from the 2-simplex. So for the representation of the inverted diagram, we are going to add the function `rotate_point()` which will receive as parameters a point and a rotating angle. Then, we will perform matrix multiplication between the point and the rotation matrix $^{()}$ to obtain the rotated point. The main module functions [46] are:

- `project_points()`: Maps (x,y,z) coordinates to planar simplex
- `project_sequence()`: Projects a point or sequence of points using `project_point()` to lists `xs, ys` for plotting with *Matplotlib*.
- `convert_coordinates()`: Convert a 3-tuple in data coordinates into to simplex data coordinates for plotting.
- `rotate_point()`: rotate a point p by the matrix multiplication of the coordinates with the rotation matrix for the desired angle.

lines.py

This module provides the necessary functions to plot the lines and contours of the triangle as well as some gridlines to define the measures of regions [46]. The main functions are:

- `line()`: Draws a line on *matplotlib* axes from `p1` to `p2`. The function can perform a rotation of the points, passing as an argument the rotation angle needed.
- `horizontal_line(),left_parallel_line(),right_parallel_line()`: plot parallel lines to the right, left and bottom axis.
- `boundary()`: Plots the boundary of the simplex. Creates and returns matplotlib axis if none given.
- `gridlines()`: Plots grid lines excluding boundary.
- `ticks()` : Set ticks marks and labels for the

plotting.py

Module containing the plotting functions. From this file we will just use some functionalities as the `resize_drawing_canvas()` used for checking whether the drawing surface of the representation is enough, `clear_matplotlib_ticks()`, for clearing the default matplotlib axes, or the one specified by the axis argument. Finally the `scatter()` function will be the one used for plotting the entropy triangle points where each point satisfies $x + y + z = scale$ [19].

ternary_axes_subplot.py

Wrapper module where the ternary class is created named as `TernaryAxesSubplot`. It will import all the necessaries functions from the functionality's packages for the implementation of the representation. The main function, `figure ()`, will create an instance of the `TernaryAxesSubplot` class, with defined arguments `ax`, used matplotlib axes, `scale` that will fix the scale of the diagram, `permutation` and the `angle` of rotation that during the inverted triangle plotting will be *-60* degrees, solving the initial problem. The default `angle` value will be *0* degrees (that is, no rotation).

TernaryAxesSubplot class

The class depicting the *ternary diagram*. The parameters for member functions simply pass through to ternary's functions with the same names. This class manages the *matplotlib* `axes`, the `scale`, and the `boundary scale` to ease the use of ternary plotting functions. In the Figure 36 we can see the depiction of both diagrams created from an instance of the TernaryAxesSubplot class.

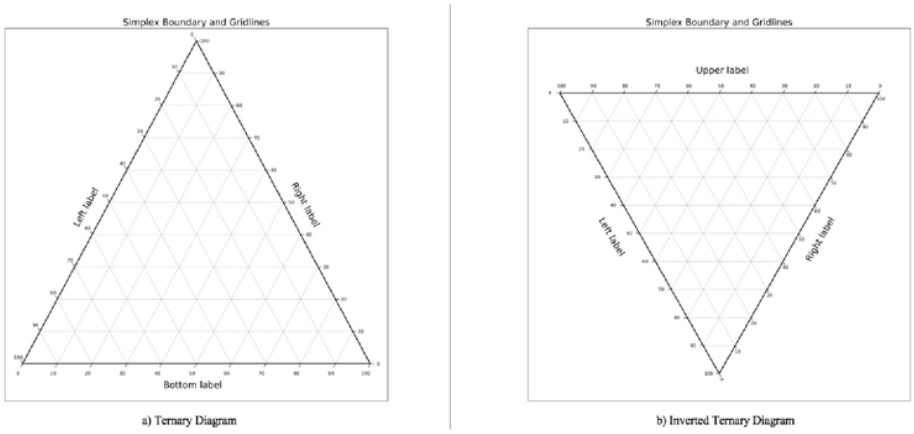


Figure 36. Ternary Diagrams for the representation of a) the Channel case and b) the Source case, rotated.

[19] https://en.wikipedia.org/wiki/Ternary_plot

3.2.4 Plotting Modules

Once solved the rotated triangle scenario with the modifications made on the ternary package, we are going to develop the necessary modules to transform the data frames calculated from section 4.2.1 functions, to the entropy coordinates necessary for the triangle points definition, as well as the function in which we will define the instance of the `TernaryAxesSubplot` class to plot both the triangle and the points. For this purpose, we previously taught about the output format of the entropies data frames (`edf`), as we can see that for each situation, the data frame contains different columns names used for the scenario recognition (*CBET, SMET or CMET*). In the following modules we will comment the approach used for the calculation of the coordinates.

coordsentropic.py

The `coordsentropic.py` module encapsulates the necessary functions to calculate the coordinates of the entropy triangle points for each case, and to identify which scenario is being analyzed according to the name of the columns of the `edf` data frame. In order to do this, a series of global variables will be declared, each of them composing lists with the possible names that the columns can take in the different situations.

- `derivedSplitSmetCoords = ["DeltaH_Pxi", "M_Pxi", "VI_Pxi"]`
- `aggregateSmetCoords = ["H_Ux", "DeltaH_Px", "M_Px", "VI_Px"]`
- `dualAggregateSmetCoords = ["H_Ux", "DeltaH_Px", "D_Px", "VI_Px"]`
- `cbetEntropicCoords = ["H_U2", "H_P2", "DeltaH_P2", "M_P2", "VI_P2"]`
- `cmetEntropicCoords = ["H_U", "H_P", "DeltaH_P", "M_P", "VI_P"]`

These variables will indicate us from an `edf` with the calculated entropic variables, which type of balance equation is being represented and therefore which type of entropic triangle have to be plotted. Consequently, comparison functions have been defined for the recognition of the `edf` coordinates type.

- `hasSplitSmetCoords(df)`: Evaluation of the coincidence of the data frame columns names with variable `derivedSplitSmetCoords`
- `hasAggregateSmetCoords(df)`: Evaluation of the coincidence of the data frame columns names with variable `aggregateSmetCoords`
- `hasDualAggregateSmetCoords(df)`: Evaluation of the coincidence of the data frame columns names with variable `dualAggregateSmetCoords`
- `hasCbetEntropicCoords(df)`: Evaluation of the coincidence of the data frame columns names with variable `cbetEntropicCoords`

- `hasCmetEntropicCoords(df)`: Evaluation of the coincidence of the data frame columns names with variable `cmetEntropicCoords`

Finally, `entcoords()` will be defined. It will take a data frame with entropic measurements, normalize it and calculate each entry coordinates according to the scale triangle´s scale.

- `entcoords (df, scale)`: The function receives as arguments a data frame that will be calculated by the `sentropies` or `jentropies` modules, and the scale that will be the size that will have the entropy triangle. The function will:
    - Takes the last 3 columns of each `edf` data frame entry and divide them between the *uniform distribution entropy*
    - Scaling the normalized balance equation of each entry by multiplying by the `scale` factor.
    - Append the tuples representing the normalized coordinates of each entry into a list (prepared for the scatter function from the `TernaryAxesSubplot`).

Entropytriagle.py

The `entropytraingle` module presents the main plotting function for the depiction of the entropy triangle as well as the points representing the entropic coordinates obtained within the evaluation metrics module. Importing the ternary package to be able to use the functionalities and represent the diagram. In addition, it presents a series of functions that will help us to plot different shapes and colors to clearly represent the points.

- `get_cmap(number)`: used for setting range of colors to represent points using the scatter function. This function has as arguments, a number of integers to return the same number of colors of a random scale using the function of `matplotlib.pyplot plt.cm.get_cmap([name of the palette] , number)`
- `markers(n)`: sets the marker that is going to be used for the representation of a point, returning a string from a list of markers.
- `entriangle()`: Function implemented for the plotting of the entropy data frame. There is a set of arguments specified for design configurations. When we use this function, it will read the columns names of the data frame we are passing as argument. With this information we will display the correct version of the triangle (normal or inverted) and the correct labels for the diagram as they vary depending from the scenario. The function will store also the image in a pdf format for a high-quality representation diagram.
- `entriangle_list()`: The functionality will be the same as the `entriangle()` but just for the channel bivariate entropy triangle and the channel multivariate entropy

triangle. `entriangle_list()` function will represent a list of data frames in the same diagram. The function will store also the image in a pdf format for a high-quality representation diagram. As an example of the `entriangle_list()` function, the Figure 37 will depict the *CMET* using the Wine dataset[20]

These are the modules implemented for the graphic representation of the tool. Working together with those created for the calculation of the balance equations will provide us the necessary functionalities to fulfill the functional objectives described in the section 2.5. However, although the functionality is developed, a number of indispensable files are still missing for the project packaging and uploading to a repository.

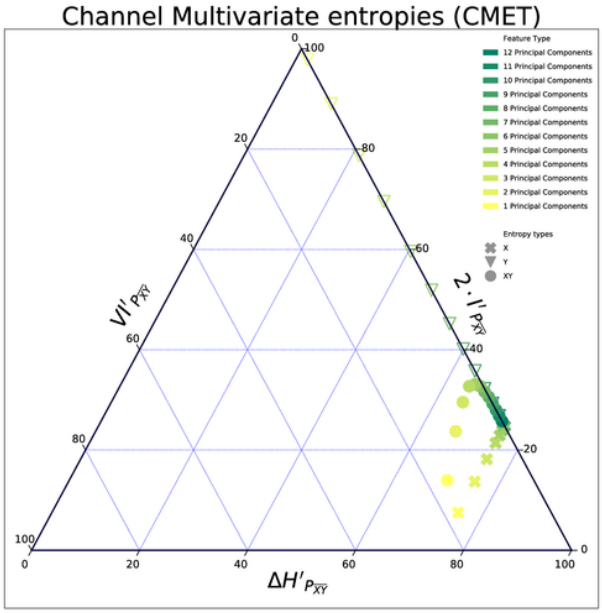


Figure 37. PCA Transformation CMET of the Wine dataset

3.2.5 Init Module

After developing all the mentioned functionalities to successfully represent the different types of entopic triangles, we still have to define the `__init__.py` file which is the necessary python module to treat the directory as a package. `__init__.py` will contain a series of import statements for the user´s access to the functionalities developed only by executing the import `entropytriangle` command after installing it. We will also import the auxiliary functions as well as the ternary package with the modifications made, for the user's convenience.

With this file in our directory we will have all the necessary components to have our `entropytriangle` package correctly implemented. However, as we have seen in the

[20] https://archive.ics.uci.edu/ml/datasets/Wine

section 3.1.4.1 since one of our objectives is to distribute the library using Python Index, we will need some additional files to upload it to the *PyPI* repository.

3.2.6 Distribution Modules

Following the requirements mentioned in section 3.1.4.1 we will create the necessary for packaging and distributing the project.

setup.py

In this file, a series of variable fields will be defined for setting up the package information. We will Indicate the project´s authors, the versions of both the package and Python that has been used during development. It will also include the project license, the download URLs and the web address of the *GitHub* repository where it is stored along with the requirements that will serve as use cases.

The module will use a script to read the project´s libraries requirements that the system must have for the installation, in the same `requirements.txt` folder.

setup.cfg

We will use this file to specify that the wheel created for the distribution is not universal, meaning it will not support both versions of Python.

requirements.txt

At the requirements file we will introduce the dependencies and the versions needed for the installation of the library. We will use the packages mentioned in section 3.1.3

readme.txt and readme.md

These two files are used as a description of the package contents for the users.

3.3 Manuals

3.3.1 User Manual

One of the main objectives set in the project was to provide a user manual with real cases, so that users who wish to use the tool are able to understand and see a real example of the use of the tool. That is why we have included in the GitHub repository[21] a series of *Jupyter Notebooks* that will allow users to have an advanced knowledge on how to use the

[21] https://github.com/Jaimedlrm/entropytriangle

package. These Jupyter notebooks are intended to supply the same functionality as R's *vignettes* and we will often refer to them by this name hence.

Figure 38 displays, as an example, one CMET use cases presented as a vignette. However, the actual output of the use cases will be added at the endo of this report as a proper user manual.

Multivariate Joint Use Case (Single DataFrameCase)

In this vignette a use case of the Multivariate Channel Entropy Triangle is presented. We are going to evaluate some feature transformation performed with the PCA algorithm in terms of information-theoretic metrics.

Importing Libraries

Once we have our package installed, we are going to import the functionalities by:

In [53]:

```
#By using the lower command we will import all the necessary packages as numpy,
 pandas, ...)
from entropytriangle import *

# If you already have installed the required packages, just import the entropy t
riangle by
#import entropytriangle
```

Download the databases

Now, you need to create a data frame. In this particular case, in the local directory where the vignette is stored we have the csv files of the databases analyzed so we just have to read them with pd.read_csv

In [39]:

```
#df = pd.read_csv('Arthritis.csv',delimiter=',',index_col='Unnamed: 0')
df = pd.read_csv('breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop(['
 Sample code number'],axis = 1).replace('?',np.nan) # In this DB the missing val
ues are indicated by ?
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
#df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop('Wine',axis = 1)
```

Figure 38. CMET Manual use case Breast Cancer

3.3.2 Installation Manual

We provide a variety of solutions for the installation to accommodate for each user's preferences. Starting from this point, the cross-platform nature of Python will allow users to install the package independently of the operating system. Thus, we will provide some options for the installation of the package such as the use of the pip or conda package managers through the command line as explained in section 3.1.3. The requirements for the installation are showed in the `requirements.txt` and the `setup.py` contained within the directory.

3.3.2.1 Pip

The package can be installed by using *pip* (section 3.1.3). Pip can install either Source distributions or wheels, but as we upload the repository to PyPI it will install the wheel format that provides faster installation. Pip will install the package in the `\Local\Programs\Python\Python36\Lib\site-packages` directory by default. So, for the installation users have to type in the command line: (Notice that you may need the `sudo` command)

```
pip install entropytriangle
```

This command will download the package directly from the PyPI repository where it is stored. If the requirements needed for the package usage are not able, you can install the libraries and the versions needed by typing the command prompt.

```
pip install -r requirements.txt
```

Being the requirements.txt a text file with the names of the needed libraries. In addition, if you download the source distribution entropytriangle.zip you can also install it by using:

```
pip install entropytriangle.zip
```

3.3.2.2 With setup.py

Alternatively, the user can clone the GitHub repository from the and run the setup.py in the usual manner as follows:

1. Clone the repository using the URL:

   ```
   git clone https://github.com/Jaimedlrm/entropytriangle.git
   ```

2. Move to the directory where the package is cloned:

   ```
   cd entropytriangle
   ```

3. Finally install the package using the setup.py file:

   ```
   sudo python setup.py install
   ```

3.3.2.3 Conda

For *conda* users, we defined the procedure of the package creation in section 3.1.4.3, therefore just by using the command

```
conda install -c jaimedlrm entropytriangle
```

The package will be installed in the root environment. Note that this *conda* command will take the package from the Anaconda cloud, while the pip version will install it from PyPI.

3.3.2.4 Using a Local copy of the repository

The last method we propose and the least practical is the "installation" of the package is storing it in the repository in which you are working and accessing it through the command import which will import the modules and functionalities needed

4. DISCUSION

In this section, we will comment the conclusions reached during the first stage of the project and whether or not the goals have been achieved, as well as the proposed future lines mentioned for a future second phase extension.

4.1 Conclusions

4.1.1 Software Developed

The main project goal was the creation of a software that would allow us to use in Python a series of information-theoretic tools defined in [1, 2, 3] due to their high usability for machine learning related tasks.

We can assert that the package presented meets the requirements that have been proposed, by correctly implementing all the necessary functionalities and allowing users to install successfully the tool from the both repositories *GitHub* and *PyPI*. Moreover, we provided a user's manual through the vignettes, also uploaded to *GitHub*, to learn how to use the package.

When programming a software project there are many decisions to be taken that will determine the fulfillment of the proposed objectives. The first success in this case was to choose Anacondas as the necessary environment to work on the code since it provides us with a series of facilities that have been fundamental for the creation of the library, either by the package's management or the accessibility to IDEs for programming and executing the code. Furthermore, the choice of a modular architecture for the library designed allowed us to focus our efforts when developing the code, dividing the package into two types of modules: those needed for the calculation of entropy, and, on the other hand, those that will provide us with visual functionalities.

The information-theoretic measures proposed in [1, 2, 3], were correctly implemented thanks to a correct data type selection and the outsourcing of the measures by the use of the packages for scientific tasks such as *pandas, NumPy or SciPy*.

However, some of the functionalities were not already implemented and therefore a set auxiliary functions were developed such as the total *discretization* of a data frame and the calculation of the *conditional entropy* of two random vectors represented through two data frames. Combining these functions to the already implemented ones, we successfully compute the entropic metrics needed for the *CBET, CMET and SMET* scenario.

Another relevant aspect was the implementation of the necessary functionalities to represent the triangles and the package with the necessary characteristics to be able to paint the rotated triangles. The modifications made under the permission of the MIT license, will allow us to execute in the future the command git pull-request to add the functionality to the package. The reuse of Marc Harper's ternary package[22] under the MIT license permission to represent the entropic triangles, and the subsequent contribution providing with the necessary features to be able to paint the rotated triangles, was decisive in achieving the objectives within the established deadlines. These modifications will be reported to the author of the library for a future inclusion on the next package versions.

To sum up, we have fulfilled the functional requirements that we laid out original, as we are able to both calculate the entropic measures needed and represent them correctly in a ternary diagram for each of the possible cases.

---

[22] https://github.com/marcharper/python-ternary

4.1.2 Package Distribution

In addition to the development of qualitive software, another important aspect is the community distribution. For that reason, we fixed as a secondary goal the correct package upgrading, so it can be accessed easily. We decided to store the package in a collaborative development platform, GitHub, not only because we control the versions through git features that facilitates software development, but also because it is a platform that is widely used by Python users, so it can be followed by users and encourage some collaboration.

Once uploaded to GitHub, we came to realize that the Python version we were using for the library developing (Python 3.6.5) has pip as the default package manager. Therefore, we looked for a solution where the package is just easily installed by using the *pip* command as seen in section 3.3.2. We concluded that we were going to create a built distribution that we would later upload to *PyPI*. In this way, users could access the package in several ways. Furthermore, we developed a package distribution for the conda users, as the pip command did not install correctly the packages in the conda manager. So, at the end the users have a wider range of options for downloading the package from different sources.

Finally, in the *GitHub* repository, a series of vignettes were added to teach users both how to use the package and how to install it. All this has allowed us to meet the requirements set for the distribution of the package as well.

4.2 Next Steps

As mentioned before, we define the roadmap knowing that some of the implementations purposed won´t be achieved. So, for a second stage we are going to point out some interesting next steps for this project, from the starter point that will be the software already implemented.

4.2.1 Improve visualization options

From the functionalities implemented by Marc Harper in [46] there are some modules not used yet that could help us to generate more powerful displays such as the representation of heat maps as presented in Figure 39, or the drawing of lines that define the tendency of the classifiers when we modify the classifier features. In addition, we could also add a better implemented legend as well as improved markers with both color and shapes more remarkable, to provide as much graphical information as possible.

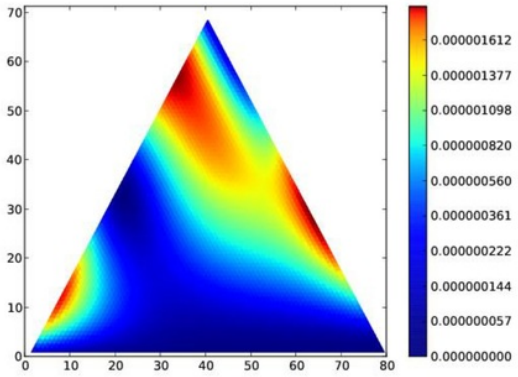


Figure 39. Representation of an Example Heatmap in a Ternary Diagram [23]

4.2.2 Include interactive Charts

Design a graphic user interface that allows us with the graph and to know the exact points and values, so that the user, moving the cursor around the graph finds all the information about the entropic points.

4.2.3 Accuracy evaluation

As an extension from the information theoretic functionalities we can add other complementary evaluation metrics for multi-class classifiers defined by the authors of the papers [1] such as the Entropy Modulated Accuracy (EMA) for the measurement of classification performance and the Normalized information transfer (NIT) which assess the accuracy of classifiers measuring the effectiveness of a learning process. Once the metrics functions are calculated we will just need to plot the values using the functions defined in this report.

4.2.3.1 Region detection

New package users may not understand properly the concepts behind the entropy triangle after reading the documentation with the theoretic explanation. For this reason, an interesting extension will be to implement a module, which will export a document with the summary of the entropy triangle analysis based on the points areas. This functionality for example will explain users the information contained within a data source in an easy manner *(SMET)*, explaining the regions that the features are displayed.

With all the changes applied to the tool, we will have a powerful tool for evaluating machine learning tasks, in a way never seen before.

---

[23] Reprinted from:
https://github.com/marcharper/pythonternary/blob/master/readme_images/23_80_0.png

APPENDIX

A. **Socio-economic framework.**

Nowadays, the importance of analyzing large volumes of data as a way to achieve a sustainable competitive advantage is an unquestionable reality in the business world. For this reason, companies are increasingly investing more and more resources to capture and store as much data as possible in order to take advantage of its subsequent analysis.

Big Data and advanced Business Intelligence techniques are changing the way companies approach many processes as decision making, relationships with customers, marketing activities, industrial processes and so on. Among Big Data techniques, machine learning is a scientific discipline which allows to identify complex patterns from incalculable volumes of data, processing them to predict some behavior. They base their operation on previous experience or knowledge that guides them in their decisions, achieving the application of this technology in multiple areas as mentioned

33

One of the main constraints in machine learning techniques development is the shortage of talent with a good experience in this field. The combination of skills that are required to master this discipline as knowledge of new programming languages (Python and R), data and systems architecture, analysis and visualization and project management, is a handicap in talent development and recruitment.

This lack of talent is creating a huge competition in talent retention and acquisition and it's going to be an area of great professional opportunities for workers in the next years. This competition is increasing considerably the salaries for machine learning professionals in a moment in which their demand is growing heavily.

As companies are increasing data collection from customers, concerns about private data usage is also increasing. New regulation on data privacy in Europe (GDPR – General Data Protection Regulation) has been developed to force companies to guarantee individuals' and companies' privacy rights.

At this moment, companies that put big data capabilities at the center of their digital transformation strategy are going to be ahead in the fight for business success in the very troubled environment that digitization is creating for traditional businesses.

B. Legal Framework

As the package is intended to be open source for future developments, the software implemented is licensed under the MIT License[24]. This license imposes very few restrictions on the reuse of software, as it grants the end user rights such as copying, modifying, merging, distributing, etc.

In Figure 40 we can see 3 different points highlighted[25]

```
MIT License

Copyright (c) 2018 Jaime de los Ríos

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Figure 40. `entropytriangle` MIT License

- *Conditions*: the condition is that the copyright statement and the rights portion is included in all copies or substantial retrievals of the Software. This is the condition that would invalidate the license if not fulfilled.
- *Rights*: are the following ones: unrestricted; including using, copying, modifying, integrating with other Software, publishing, sublicensing, or selling copies of the Software, and allowing persons to whom the Software is delivered to do the same.
- *Responsibility limitation,* lastly you have a disclaimer or note of limitation of the usual responsibility in this type of licenses.

[24] https://opensource.org/licenses/MIT
[25] http://umh2820.edu.umh.es/wp-content/uploads/sites/885/2016/02/Licencia-MIT.pdf

C. Economic Budget

The project can be considered as an academic study, which does not involve costs. However, its accomplishment has involved a time and material cost to the student and the tutor who have supported it. Therefore, the cost associated with the project will be estimated by identifying the work-effort involved and the materials employed in order to estimate it for future similar investigations.

We will assign a specific price per work hour for the student and the tutor aligned to the current market framework (September 2018). Therefore, we will estimate the time consumed by the workforce in the four project months, by designating a number of ours per fractioned stage.

Firstly, the Table 5 represents the different phases of the project. To analyze the working time that the tutor has supervised the work, we estimate that an hour of work of the student, takes 15% of the tutor´s time.

Table 5. Project Stages

Proyect Stages	Duration (Hours)
Technical documentation	90 hours
Package Development	150 hours
Report creation	100 hours

Identifying the estimated cost of an engineer working hours in the market, working for about 2,000 € per month a total of 22 working days and 8 hours daily, we consider a stipend of 11.3 € / hour of the student. The tutor's hour will earn approximately 45 €/hour. It should be mentioned that the calculated price is without taxes that will suppose an additional 21% (VAT). Therefore, we can calculate the costs of the personnel as observed in Table 6.

Table 6. Workforce cost

Workforce	Unit price	Hours	Amount
Student / Engineer	11,2 €/hour	240 hours	2.712 €
Tutor (Supervisor)	45 €/hour	36 hours	1.620 €
	Total	**276 hours**	**4.332,00 €**

Once we calculated the cost of the required manpower, we will evaluate the cost of the material used. As the project is related to software implementation, the cost of systems used as Anaconda, GitHub or PyPI, will be free. At the same time, the workspace used in this case was the university, the reason why we do not calculate office supply expenses. However, assuming that the computer used has a monthly cost of € 30 per month and including the project´s travel costs, the charge of material will be as seen in the Table 7

Table 7. Material Amount

Expense	Hours	Amount
Free software and storage	0 €/hour	0 €
Computer	30 €/month	120 €
Office Supplies	0 €/month	0 €
Other Expenses	40 €/month	160,00 €
	Total	**280,00 €**

Hence, the Table 8 will represent the calculation of the total project budget bearing in mind the taxes applied to the final values.

Table 8. Total Project Budget

Expense	Amount
Workforce	4.332 €
Material	280 €
Taxable	**4.612 €**
VAT (21 %)	992,40 €
TOTAL	**5.534,40 €**

D. Time Schedule

The project lasted 4 months and encompassed a series of phases that can be divided into 3 main groups. First of all, there is the technical documentation stage, where the necessary knowledge was acquired in order to successfully implement the necessary tools.

Next, we carried out a design and implementation phase in which we created the distributable package and then tested it. Finally, once the code was finalized, we proceeded to complete the Project report where we documented all the knowledge and tools developed during the work.

These general intervals have been broken down into a series of stages which are as follows:

- *Stage 0*: Project planning: Initially, we proceeded to design a roadmap with the steps to follow for the correct work execution.
- *Stage 1*: Documentation phase: This stage corresponds to the procedures carried out to find and extract all the necessary theoretical concepts on which the tools we implemented were based. The concepts are those shown in chapter 2.
- *Stage 2*: Development planning: Phase in which the guidelines for the correct execution of the code are defined. It was carried out in parallel with the end of the documentation phase.
- *Stage 3*: Software development: Interval where the package was developed with all the necessary requirements to achieve the objectives set. It covers the period of time taken to program the software explained in section 3.2.
- *Stage 4*: Software testing phase: Period where the correct functioning of the library was checked.
- *Stage 5*: Creation of use cases: Practical cases were designed to serve a user manual for the community
- *Stage 6*: Reporting: Phase in which the report was created where all the characteristics of the project are found.

Figure 41 displays a visual depict of the diagram Gantt showing the mentioned intervals

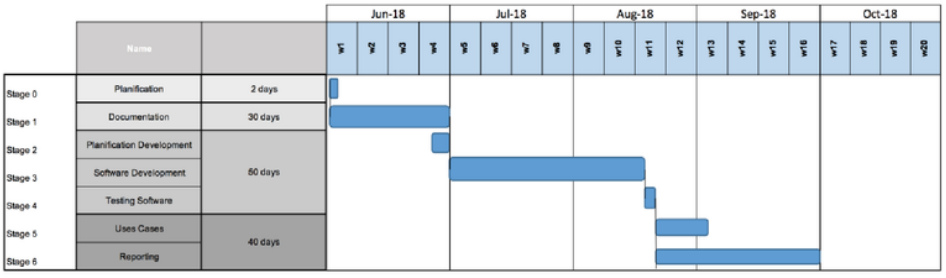


Figure 41. Projects Stages Gantt diagram

Bibliography

[1] F. J. Valverde-Albacete and C. Pelaez Moreno, "100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox.," *PLOS ONE* , vol. doi:10.1371/journal.pone.0084217, 2014.

[2] F. J. Valverde-Albacete and C. Pelaez Moreno, "The Evaluation of Data Sources using Multivariate Entropy Tools," *Expert Syst. Appl,* vol. 78, p. 145–157, 2017.

[3] F. J. Valverde-Albacete and C. Pelaez Moreno, "Assessing Information Transmission in Data Transformations with the Channel Multivariate Entropy Triangle," *Entropy,* vol. 20, no. (7), p. 498, June 2018.

[4] F. J. Valverde-Albacete and C. Pelaez Moreno, " Two information-theoretic tools to assess the performance of multi-class classifiers," *Pattern Recognit. Lett,* vol. 31, p. 1665–1671, 2010.

[5] "Ocw.mit.edu," [Online]. Available: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-441-information-theory-spring-2010/lecture-notes/MIT6_441S10_lec01.pdf. [Accessed 2 September 2018].

[6] "En.wikipedia.org," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Entropy_(information_theory). [Accessed 14 August 2018].

[7] R. Yeung, "A new outlook on Shannon's information measures," *IEEE Trans. Inf. Theory,* vol. 37, p. 466–474, 1991.

[8] F. Reza, "An Introduction to Information Theory," in *McGraw-Hill Electrical and Electronic Engineering Series*, New York, NY, USA; Toronto, ON, Canada; London, UK, McGraw-Hill Book Co, 1961.

[9] C. Shannon, "A mathematical theory of communication.," *Bell Syst. Tech. J,* vol. XXVII, pp. 379–423, 623–656, 1948.

[10] M. Meila, "Comparing clusterings—An information based distance," *J. Multivar. Anal,* vol. 28, p. 875–893, 2007.

[11] D. MacKay, "Information Theory, Inference and Learning Algorithms," Cambridge, UK, Cambridge University Press, 2003.

[12] R. James, C. Ellison and J. Crutchfield, "Anatomy of a bit: Information in a time series observation," *Chaos,* vol. 21, no. 037109, 2011.

[13] S. Watanabe, "Information theoretical analysis of multivariate correlation," *J. Res. Dev,* vol. 4, pp. 66-82, 1960.

[14] G. Tononi, O. Sporns and G. Edelman, " A measure for brain complexity: Relating functional segregation and integration in the nervous system," *Proc. Natl. Acad. Sci USA,* vol. 91, p. 5033–5037, 1994.

[15] M. Studený and J. Vejnarová, "The Multiinformation Function as a Tool for Measuring Stochastic Dependence. In Learning in Graphical Models," *Springer: Dordrecht, The Netherlands,* p. 261–297, 1998.

[16] T. Han, "Nonnegative entropy measures of multivariate symmetric correlations.," *Inf. Control,* vol. 36, p. 133–156, 1978.

[17] S. Abdallah and M. Plumbley, " A measure of statistical complexity based on predictive information with application to finite spin systems," *Phys. Lett. A,* vol. 376, p. 275–281, 2012,.

[18] G. Tononi, " Complexity and coherency: Integrating information in the brain," *Trends Cognit. Sci.,* vol. 2, p. 474–484, 1998.

[19] W. McGill, "Multivariate information transmission.," *Psychometrika,* vol. 19, p. 97–116, 1954.

[20] T. Sun Han, "Multiple mutual informations and multiple interactions in frequency data.," *Inf. Control,* vol. 46, p. 26–45, 1980.

[21] A. Bell, in *The co-information lattice. In Proceedings of the Fifth International Workshop on Independent Component Analysis and Blind Signal Separation*, Nara, Japan, 1-4 April 2003.

[22] F. Valverde Albacete and C. Peláez-Moreno, " The Multivariate Entropy Triangle and Applications.," in *Hybrid Artificial Intelligence Systems (HAIS 2016);*, Seville, Spain, Springer, , 2016;, p. 647–658.

[23] V. Pawlowsky-Glahn, J. Egozcue and R. Tolosana-Delgado, "Modeling and Analysis of Compositional Data," Chichester, UK, John Wiley & Sons, 2015.

[24] F. J. Valverde-Albacete and C. Pelaez Moreno, "The Channel Multivariate Entropy Triangle and Balance Equation," *Journal of latex class files,* vol. 14, p. 8, August 2015.

[25] "Learnpython.org," 2018. [Online]. Available: https://www.learnpython.org/en/Modules_and_Packages. [Accessed 16 September 2018].

[26] Grouchy and Mike, "Be Pythonic: __init__.py - Mike Grouchy," [Online]. Available: http://mikegrouchy.com/blog/2012/05/be-pythonic-__init__py.html. [Accessed 16 September 2018].

[27] Pandas.pydata.org, "Python Data Analysis Library — pandas: Python Data Analysis Library," [Online]. Available: https://pandas.pydata.org/. [Accessed 17 September 2018].

[28] Numpy.org, "What is NumPy? — NumPy v1.16.dev0 Manual," 2018. [Online]. Available: https://www.numpy.org/devdocs/user/whatisnumpy.html. [Accessed 16 September 2018].

[29] "Python Data Analysis Library — pandas: Python Data Analysis Library," 2018. [Online]. Available: https://pandas.pydata.org/. [Accessed 17 September 2018].

[30] K. Singh and Atul, 2018. [Online]. Available: https://www.quora.com/What-Python-libraries-do-you-use-in-machine-learning. [Accessed 17 September 2018].

[31] Matplotlib.org, "Matplotlib: Python plotting — Matplotlib 3.0.0 documentation," [Online]. Available: https://matplotlib.org/ . [Accessed 17 September 2018].

[32] "Python Package Manager," 18. [Online]. Available: https://en.wikipedia.org/wiki/Python_Package_Manager. [Accessed 17 September 2018].

[33] "pip (package manager)," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Pip_(package_manager). [Accessed 17 September 2018].

[34] freeCodeCamp.org, "Why You Need Python Environments and How to Manage Them with Conda," 2018. [Online]. Available: https://medium.freecodecamp.org/why-you-need-python-environments-and-how-to-manage-them-with-conda-85f155f4353c. [Accessed September 2018].

[35] Conda.io, "Managing environments — Conda documentation," 2018. [Online]. Available: https://conda.io/docs/user-guide/tasks/manage-environments.html. [Accessed September 2018].

[36] En.wikipedia.org, "Anaconda (Python distribution)," [Online]. Available: https://en.wikipedia.org/wiki/Anaconda_(Python_distribution). [Accessed September 2018].

[37] Docs.anaconda.com, "Anaconda Navigator — Anaconda 2.0 documentation," [Online]. Available: https://docs.anaconda.com/anaconda/navigator/. [Accessed September 2018].

[38] Anaconda, "Distribution - Anaconda," 2018. [Online]. Available: https://www.anaconda.com/distribution/. [Accessed 14 September 2018].

[39] "Python Package Index," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Python_Package_Index#Notes. [Accessed September 2018].

[40] Packaging.python.org, "Packaging and distributing projects — Python Packaging User Guide," 2018. [Online]. Available: https://packaging.python.org/guides/distributing-packages-using-setuptools/#initial-files. [Accessed 19 September 2018].

[41] Packaging.python.org, "Glossary — Python Packaging User Guide," [Online]. Available: https://packaging.python.org/glossary/#term-built-distribution. [Accessed September 2018].

[42] Packaging.python.org, "An Overview of Packaging for Python — Python Packaging User Guide," [Online]. Available: https://packaging.python.org/overview.

[43] Packaging.python.org, "Project Summaries — Python Packaging User Guide," 2018. [Online]. Available: https://packaging.python.org/key_projects/#twine. [Accessed 15 September 2018].

[44] Conda.io, "Conda build recipes — Conda documentation," [Online]. Available: https://conda.io/docs/user-guide/tasks/build-packages/recipe.htm. [Accessed 21 September 2018].

[45] Meyer and P. E., "Information-Theoretic Measures," http://homepage.meyerp.com/software, 14 08 2009. [Online]. Available: https://cran.r-project.org/web/packages/infotheo/infotheo.pdf.

[46] Harper and Marc, "python-ternary: Ternary Plots in Python," *Zenodo,* vol. 10.5281/zenodo.34938, 2015.

[47] "Web.standford.edu," [Online]. Available: https://web.stanford.edu/~montanar/RESEARCH/BOOK/partA.pdf. [Accessed 20 September 2018].

E. USER MANUAL (VIGNETTES) CBET

Binary Joint Use Case (Single DataFrameCase)

In this vignette a use case of the Binary Channel Entropy Triangle is presented. We are going to evaluate different multiclass-classification scenarios in order to analyze the data. The main functionalities for the classification of the database will be extracted from: https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/ (https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/)

Importing Libraries

We import the package entropytriangle, which will import the modules needed for the evaluation

In [1]:

```
from entropytriangle import * #importing all modules necessary for the plotting
```

Download the databases

In this case, the csv files for the use case, are stored locally

In [2]:

```
#df = pd.read_csv('Arthitris.csv',delimiter=',',index_col='Unnamed: 0').drop(['I
D'],axis = 1)
df = pd.read_csv('Breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop([
'Sample code number'],axis = 1).replace('?',np.nan) # in this DB the missing val
ues are represented as '?'
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
#df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop(['Wine'],axis = 1)
```

In [3]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 699 entries, 1 to 699
Data columns (total 10 columns):
Clump Thickness               699 non-null int64
Uniformity of Cell Size       699 non-null int64
Uniformity of Cell Shape      699 non-null int64
Marginal Adhesion             699 non-null int64
Single Epithelial Cell Size   699 non-null int64
Bare Nuclei                   683 non-null float64
Bland Chromatin               699 non-null int64
Normal Nucleoli               699 non-null int64
Mitoses                       699 non-null int64
Class                         699 non-null object
dtypes: float64(1), int64(8), object(1)
memory usage: 60.1+ KB
```

In [4]:

```
df = df.fillna(0)
df.head(5)
```

Out[4]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitose
1	5	1	1	1	2	1.0	3	1	
2	5	4	4	5	7	10.0	3	2	
3	3	1	1	1	2	2.0	3	1	
4	6	8	8	1	3	4.0	3	7	
5	4	1	1	3	2	1.0	3	1	

Prepare the data for the classification (Features - Classes)

We are going to load the train_test_split that will allow us to separate automatically the data in a train/test sets. Additionally, we are going to import the contingency matrix that will allow us to calculate the joint entropy matrix of the classifier

In [5]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Separating the data farmes features and classes

In [6]:

```
X = df[df.columns[df.columns != 'Class']]
y = df['Class']
```

We are now to define some classificators for evaluating their performance with the BreastCancer database

In [7]:

```
# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

KNN

KNN - Classifier (Don´t run the code if you want to implement other classifier)

Downloading the sklearn Knn classifier and fitting it into our data

In [8]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
```

Out[8]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkows
ki',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

Once we have design our classifier, we are going to evaluate the accuracy

In [9]:

```
print(knn.score(X_test, y_test))
```

```
0.9771428571428571
```

Finally, we will compute the confusion matrix of the classified data

In [10]:

```
knn_predictions = knn.predict(X_test)
cm = confusion_matrix(y_test, knn_predictions)
cm
```

Out[10]:

```
array([[110,   2],
       [  2,  61]])
```

KNN - Channel Bivariate Entropy Triangle Plotting

The last step will be calculating the entropic measures for the contingency matrix and plot the entropy triangle. The coordinates will be calculated multiplying the normalized values needed by the scale used for plotting the triangle, and will appear behind the triangle plot for comparission

In [11]:

```
edf = jentropies_binary(cm)
#edf1 = jentropies(pd.DataFrame(y_test),pd.DataFrame(knn_predictions))
```

In [12]:

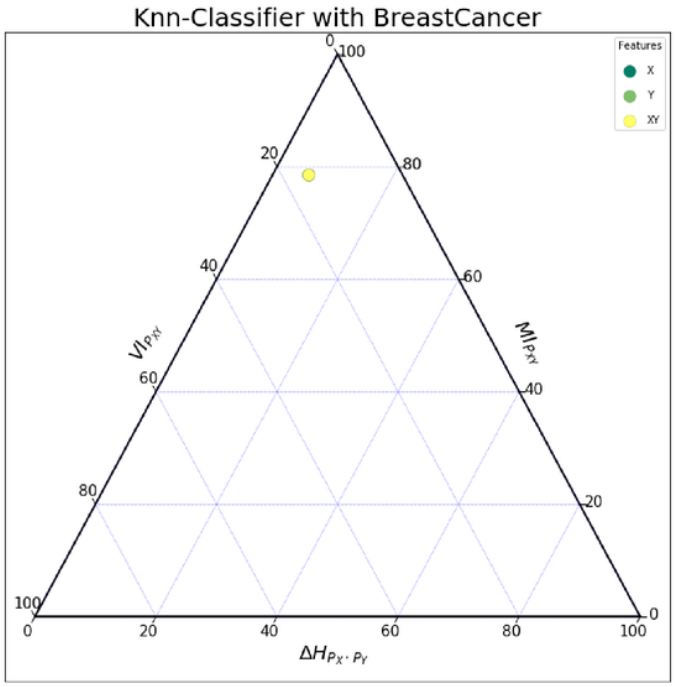
```
edf
#edf1
```

Out[12]:

Type	H_U2	H_P2	DeltaH_P2	M_P2	VI_P2
X	1.0	0.942683	0.057317	0.786867	0.155816
Y	1.0	0.942683	0.057317	0.786867	0.155816
XY	2.0	1.885366	0.114634	1.573734	0.311632

In [13]:

```
entriangle(edf,s_mk=150, gridl = 20, pltscale=12 ,fonts = 20, ticks_size= 15,cha
rt_title="Knn-Classifier with BreastCancer")
```



Naive Bayes

Naive Bayes - Classificator

Now we are going to provide another example of the channel bivariate entropy triangle using the Naive Bayes classificatior. We will need first to download the GaussianNB class from scikit-learn and apply the following command to train the classifier

In [14]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB().fit(X_train, y_train)
```

Once we have model our classifier, we are going to evaluate the accuracy using the test set

In [15]:

```
print(gnb.score(X_test, y_test))
```

```
0.9542857142857143
```

Finally, we will compute the confusion matrix of the classified data

In [16]:

```
gnb_predictions = gnb.predict(X_test)
cm = confusion_matrix(y_test, gnb_predictions)
cm
```

Out[16]:

```
array([[106,   6],
       [  2,  61]])
```

Naive Bayes - Channel Bivariate Entropy Triangle Plotting

The last step will be calculating the entropic measures for the contingency matrix and plot the entropy triangle. The coordinates will be calculated multiplying the normalized values needed by the scale used for plotting the triangle. First we will calculate the entropy data frame for the contingency matrix

In [17]:

```
edf = jentropies_binary(cm)
```

In [18]:

```
edf
```

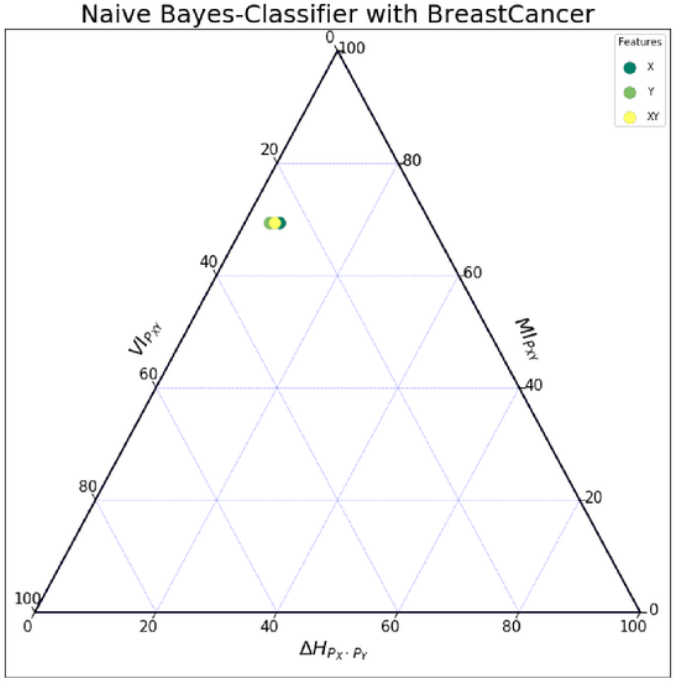
Out[18]:

Type	H_U2	H_P2	DeltaH_P2	M_P2	VI_P2
X	1.0	0.942683	0.057317	0.694046	0.248637
Y	1.0	0.960035	0.039965	0.694046	0.265989
XY	2.0	1.902718	0.097282	1.388092	0.514626

Once we obtained the appropiate entropy data frame, we just need to execute the entriangle function for the plotting

In [19]:

```
entriangle(edf,s_mk=150, gridl = 20, pltscale=12 ,fonts = 20, ticks_size= 15,cha
rt_title="Naive Bayes-Classifier with BreastCancer")
```


Naive Bayes-Classifier with BreastCancer
Features
X
Y
XY
VI_{PXY}
MI_{PXY}
$\Delta H_{P_X \cdot P_Y}$
0
20
40
60
80
100

F. USER MANUAL (VIGNETTES) SMET

Multivariate Source Use Case (Single DataFrameCase)

In this vignette I will represent a use case of the Source Multivariate Entropy Triangle with some individual Databases

Importing Libraries

We import the package entropytriangle, which will import the modules needed for the evaluation

In [1]:

```
from entropytriangle import * #importing all modules necessary for the plotting
```

Dowloading a set of Databases

In this case, the csv files for the use case, are stored locally

In [2]:

```
#df = pd.read_csv('Arthritis.csv',delimiter=',',index_col='Unnamed: 0').drop(['I
D'],axis = 1)
#df = pd.read_csv('Breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop
(['Sample code number'],axis = 1).replace('?',np.nan) # in this DB the missing v
alues are represented as '?'
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop(['Wine'],axis = 1)
```

In [3]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

In [4]:

```
df.head(10)
```

Out[4]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

Discretizing the Data before entropy calculation

We have defined a function for discretizing a hole dataset, the function divides de entries in "NROWS(DF)^(1/3)" equally sized spaces, and turn the data types in "categories"

In [5]:

```
df = discretization(df)
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/auxfunc.py:35: UserWarning: Discretizing data!
  warning("Discretizing data!")
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null category
SepalWidthCm     150 non-null category
PetalLengthCm    150 non-null category
PetalWidthCm     150 non-null category
Species          150 non-null category
dtypes: category(5)
memory usage: 2.8 KB
```

In [7]:

```
df.head(10)
```

Out[7]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	3	0	0	0
2	0	2	0	0	0
3	0	2	0	0	0
4	0	2	0	0	0
5	0	3	0	0	0
6	1	3	0	0	0
7	0	2	0	0	0
8	0	2	0	0	0
9	0	1	0	0	0
10	0	2	0	0	0

Source Entropies Measures calculation

Once we have our data discretized, we will start by calculating the values of the entropies for the posterior plots

In [8]:

```
'''
As the database is previosly discretized we won´t need the values of the bins
'Type variable select the entropy calculation:'
    Total: Total source entropy decomposition (CPx)
    Dual : Dual source entropy decomposition (DPx instead of CPx)
'''

edf = sentropies(df , type = 'total' , base = 2)
```

In [9]:

```
edf
```

Out[9]:

Name	H_Uxi	H_Pxi	DeltaH_Pxi	M_Pxi	VI_Pxi
SepalLengthCm	2.321928	2.200620	0.121308	1.417675	0.782945
SepalWidthCm	2.321928	1.841723	0.480205	0.917768	0.923955
PetalLengthCm	2.321928	1.995571	0.326357	1.738118	0.257453
PetalWidthCm	2.321928	2.137460	0.184468	1.654826	0.482635
Species	1.584963	1.584963	0.000000	1.465241	0.119721
AGGREGATE	10.872675	9.760337	1.112338	7.193628	2.566709

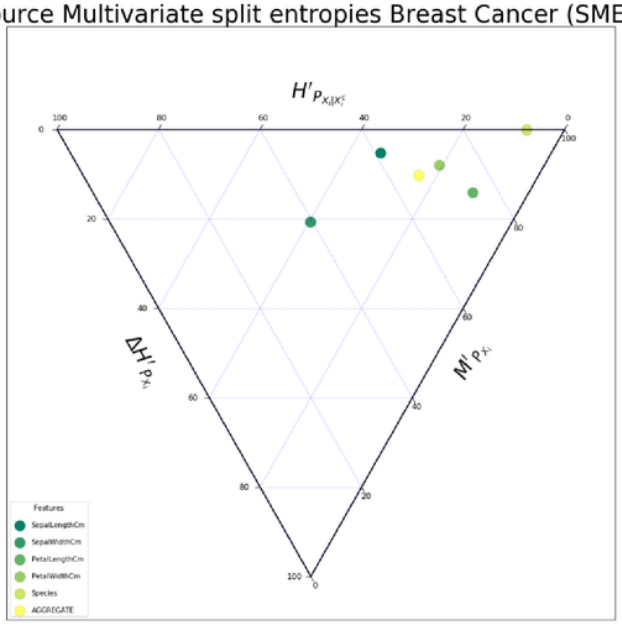
Source Entropies Entropy Triangle Plotting

The last step will be plotting the values calculated previously. The coordinates will be calculated multiplying the normalized values needed by the scale used for plotting the triangle, and will appear behind the triangle plot for comparission

In [10]:

```
entriangle(edf,s_mk=250,scale= 100, pltscale=16 , ticks_size=12, gridl = 20, cha
rt_title = "Source Multivariate split entropies Breast Cancer (SMET)")
```

Source Multivariate split entropies Breast Cancer (SMET)



G. USER MANUAL (VIGNETTES) CMET

Multivariate Joint Use Case (Single DataFrameCase)

In this vignette a use case of the Multivariate Channel Entropy Triangle is presented. We are going to evaluate some feature transformation performed with the PCA algirthm.

Importing Libraries

We import the package entropytriangle, which will import the modules needed for the evaluation

In [1]:

```
from entropytriangle import * #importing all modules necessary for the plotting
```

Download the databases

In this case, the csv files for the use case, are stored locally. Now it´s time to load the database in which we are going to apply the feature transformation

In [2]:

```
#df = pd.read_csv('Arthitris.csv',delimiter=',',index_col='Unnamed: 0').drop(['I
D'],axis = 1)
#df = pd.read_csv('Breast_data.csv',delimiter=',',index_col='Unnamed: 0').drop
(['Sample code number'],axis = 1).replace('?',np.nan) # in this DB the missing v
alues are represented as '?'
#df = pd.read_csv('Glass.csv',delimiter=',')
#df = pd.read_csv('Ionosphere.csv',delimiter=',')
df = pd.read_csv('Iris.csv',delimiter=',',index_col='Id')
#df = pd.read_csv('Wine.csv',delimiter=',').drop(['Wine'],axis = 1)
```

In [3]:

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 150 entries, 1 to 150
Data columns (total 5 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), object(1)
memory usage: 7.0+ KB
```

In [4]:

```
df.head(5)
```

Out[4]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

In [5]:

```
df = discretization(df).fillna(0)
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/auxfunc.py:35: UserWarning: Discretizing data!
  warning("Discretizing data!")
```

Prepare the data for the PCA feature transformation (Features - Classes)

Importing the Sklearn modules for the feature transformation

In [6]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Splitting the Data for the Standarization of the features before the transformation

In [7]:

```
features = df.columns.drop('Species')
x = df[df.columns.drop('Species')].values
# Separating out the target
y = df.loc[:,['Species']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/sklea
rn/utils/validation.py:475: DataConversionWarning: Data with input d
type object was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Transformation of the data. We will store the entropy dataframes in a list, which will store in each possition the features transformations with the corresponding number of number of principal components. The number of principal components will be:

Number of cols of original df - index

Example list[0] = Feature transformation with (iris features cols = 4) - (index = 0) = 4 Principal components

In [8]:

```
li = list()
for i in range(len(df.columns)):
    pca = PCA(n_components = (len(df.columns)-1)-i)
    principalComponents = pca.fit_transform(x)
    columns = list(map(lambda x: "principal component " + str(x), range(len(df.c
olumns)-1-i)))
    principalDf = pd.DataFrame(data = principalComponents, columns= columns)
    li.append(principalDf)
```

Channel Multivariate Entropy Triangle

Calculation of the entropy Data Frame for each of the dataframes of the list

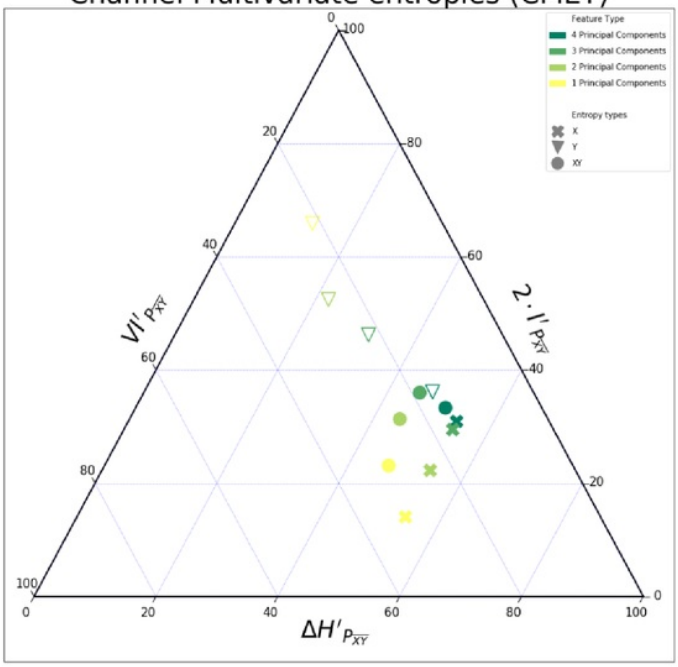
In [9]:

```
edf = list()
for i in range(len(li)-1):
    edf.append(jentropies(df,li[i]))
```

```
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/jentropies.py:50: UserWarning: Discretizing data from X D
ataFrame before entropy calculation!
  warning("Discretizing data from X DataFrame before entropy calcula
tion!") #' Throwing a Warning for communicating a discretization of
data
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/auxfunc.py:35: UserWarning: Discretizing data!
  warning("Discretizing data!")
/Users/jaime.de.los.rios/anaconda3/lib/python3.6/site-packages/entro
pytriangle/jentropies.py:54: UserWarning: Discretizing data from X D
ataFrame before entropy calculation!
  warning("Discretizing data from X DataFrame before entropy calcula
tion!") #' Throwing a Warning for communicating a discretization of
data
```

In [10]:

```
entriangle_list(edf,s_mk=300,pltscale=15)
```



PRUEBA 1

INFORME DE ORIGINALIDAD

12%	7%	10%	1%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	Francisco J. Valverde-Albacete, Carmen Peláez-Moreno. "The evaluation of data sources using multivariate entropy tools", Expert Systems with Applications, 2017 Publicación	4%
2	mdpi.com Fuente de Internet	3%
3	Francisco Valverde-Albacete, Carmen Peláez-Moreno. "Assessing Information Transmission in Data Transformations with the Channel Multivariate Entropy Triangle", Entropy, 2018 Publicación	2%
4	Francisco J. Valverde-Albacete, Carmen Peláez-Moreno. "Two information-theoretic tools to assess the performance of multi-class classifiers", Pattern Recognition Letters, 2010 Publicación	<1%
5	www.info612.ece.mcgill.ca Fuente de Internet	<1%

6	journals.plos.org Fuente de Internet	<1%
7	Submitted to Universidad Carlos III de Madrid Trabajo del estudiante	<1%
8	www.mdpi.com Fuente de Internet	<1%
9	www.coursehero.com Fuente de Internet	<1%
10	Submitted to Wheeler High School Trabajo del estudiante	<1%
11	codingcompiler.com Fuente de Internet	<1%
12	link.springer.com Fuente de Internet	<1%
13	Submitted to Harrisburg University of Science and Technology Trabajo del estudiante	<1%
14	johanlouwers.blogspot.com Fuente de Internet	<1%
15	Vani K, Deepa Gupta. "Text plagiarism classification using syntax based linguistic features", Expert Systems with Applications, 2017 Publicación	<1%

16	repositorio.uam.es Fuente de Internet	<1%
17	umar-yusuf.blogspot.com Fuente de Internet	<1%
18	Thomas Mailund. "Beginning Data Science in R", Springer Nature, 2017 Publicación	<1%
19	packaging.python.org Fuente de Internet	<1%
20	Submitted to Indian Institute of Technology, Madras Trabajo del estudiante	<1%
21	pastel.archives-ouvertes.fr Fuente de Internet	<1%
22	Lecture Notes in Electrical Engineering, 2011. Publicación	<1%
23	jcrps.com Fuente de Internet	<1%
24	Lecture Notes in Computer Science, 2016. Publicación	<1%
25	repository.asu.edu Fuente de Internet	<1%
26	Submitted to Universidad Pontificia de Salamanca	<1%

Trabajo del estudiante

27	Submitted to University of Stellenbosch, South Africa Trabajo del estudiante	<1%
28	docs.scipy.org Fuente de Internet	<1%
29	discovery.ucl.ac.uk Fuente de Internet	<1%
30	Tamas Ferenci, Levente Kovacs. "Using total correlation to discover related clusters of clinical chemistry parameters", 2014 IEEE 12th International Symposium on Intelligent Systems and Informatics (SISY), 2014 Publicación	<1%
31	Submitted to Pathfinder Enterprises Trabajo del estudiante	<1%
32	sachajournals.com Fuente de Internet	<1%
33	Communications in Computer and Information Science, 2016. Publicación	<1%
34	hal.archives-ouvertes.fr Fuente de Internet	<1%
35	openaccess.city.ac.uk Fuente de Internet	<1%

36	H.A. Babri. "Bayesian Learning for Software Architecture Recovery", 2007 International Conference on Electrical Engineering, 04/2007 Publicación	<1%
37	kkpatel7.files.wordpress.com Fuente de Internet	<1%
38	tel.archives-ouvertes.fr Fuente de Internet	<1%
39	RiÅ†Ä·eviÄ s, K., and R. Torkar. "Equality in cumulative voting: A systematic review with an improvement proposal", Information and Software Technology, 2013. Publicación	<1%
40	www.predictiveanalyticstoday.com Fuente de Internet	<1%
41	uhra.herts.ac.uk Fuente de Internet	<1%
42	laral.istc.cnr.it Fuente de Internet	<1%
43	Ryan G. James, Christopher J. Ellison, James P. Crutchfield. "Anatomy of a bit: Information in a time series observation", Chaos: An Interdisciplinary Journal of Nonlinear Science, 2011 Publicación	<1%

44	Valverde-Albacete, F.J.. "Two information-theoretic tools to assess the performance of multi-class classifiers", Pattern Recognition Letters, 20100901 Publicación	<1%
45	Slimane Lemmouchi. "Robustness of emerged community in social network", Proceedings of the International Conference on Management of Emergent Digital EcoSystems - MEDES 09 MEDES 09, 2009 Publicación	<1%
46	Valverde-Albacete, Francisco J., and Carmen Peláez-Moreno. "100% Classification Accuracy Considered Harmful: The Normalized Information Transfer Factor Explains the Accuracy Paradox", PLoS ONE, 2014. Publicación	<1%
47	"Advanced Data Mining and Applications", Springer Nature America, Inc, 2010 Publicación	<1%
48	Francisco J. Valverde-Albacete, Carmen Peláez-Moreno. "100% Classification Accuracy Considered Harmful: The Normalized Information Transfer Factor Explains the Accuracy Paradox", PLoS ONE, 2014 Publicación	<1%

Tho Hoan Pham, Tu Bao Ho, Quynh Diep

49 Nguyen, Dang Hung Tran, Van Hoang Nguyen. "Multivariate Mutual Information Measures for Discovering Biological Networks", 2012 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future, 2012

<1%

Publicación

Excluir citas	Activo	Excluir coincidencias	Apagado
Excluir bibliografía	Activo		