ESTIMADORES DE LA ENTROPÍA E INFORMACIÓN MUTUA

SOFÍA ALMEIDA BRUNO

RESPONSABLES DE TUTORIZACIÓN:

Alberto Guillén Perales
Departamento de Arquitectura y Tecnología de Computadores
ETSIIT

Ana I. Garralda Guillem
Departamento de Matemática Aplicada
Facultad de Ciencias

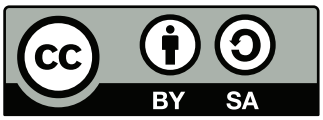
UNIVERSIDAD
DE GRANADA

Doble Grado en Ingeniería Informática y Matemáticas
ETSIIT – Facultad de Ciencias
Universidad de Granada
Curso 2019-2020

Sofía Almeida Bruno: *Estimadores de la entropía e información mutua*, Doble Grado en Ingeniería Informática y Matemáticas, Curso 2019-2020

RESPONSABLES DE TUTORIZACIÓN:
Alberto Guillén Perales
Departamento de Arquitectura y Tecnología de Computadores
ETSIIT

Ana I. Garralda Guillem
Departamento de Matemática Aplicada
Facultad de Ciencias



AGRADECIMIENTOS

Me gustaría agradecer a Alberto y Ana que aceptaran tutorizarme el trabajo fin de grado, los consejos y correcciones durante la realización del mismo y el tiempo dedicado para poder resolver mis dudas.

Agradezco a mis padres por ser siempre el motor de repuesto y por ayudarme a centrarme en el próximo kilómetro.

Quisiera agradecer a los que han leído este trabajo y me han ayudado cuando no encontraba la mejor forma de redactar algo, así como con las "dudas LaTeX" y la edición de imágenes.

Finalmente, me gustaría agradecer a todos los que me han acompañado en estos cinco años: a los que me han enseñado, a los que me han ayudado, a los que me han inspirado, a los que han compartido tantas horas de estudio (colectivo o individual), a los que han estado ahí cuando ha hecho falta y cuando no.

RESUMEN

En este trabajo se estudiarán algunos conceptos primordiales en el ámbito de la teoría de la información: entropía e información mutua, además de sus propiedades y conceptos relacionados. El objetivo principal de este proyecto es el estudio de estimadores, tanto de la entropía como de la información mutua, desde un punto de vista teórico, donde nos centraremos en los estimadores basados en los k vecinos más cercanos y en los del tipo de Kozachenko - Leonenko, y práctico, donde realizaremos un análisis experimental de implementaciones de algunos de los estimadores anteriores desde diferentes perspectivas: análisis del error, estudio de las diferencias medias mediante *T-test* y análisis de rendimiento. Las conclusiones obtenidas en el análisis comparativo de los diferentes estimadores a nivel práctico dependerán de la dimensión y del tamaño de las muestras.

PALABRAS CLAVE: entropía, información mutua, estimadores basados en los k vecinos, estimadores de tipo de Kozachenko - Leonenko, Python.

ABSTRACT

This project is developed in the context of concepts related to information theory, such as entropy and mutual information. Properties and proposals about how to estimate such concepts are studied. In addition, an analysis from a practical perspective for some implementations of these estimators is presented.

Many concepts associated with the information provided by random variables were born in the scope of information theory. Information theory searches for theoretical limits to the message communication capacity, moreover, it develops schemes that allow to achive an optimal capacity for message exchanging. Some authors consider information theory as a branch of probability theory, although it can also be viewed as a branch of theory of dynamical systems.

Concepts related to information have been around for a long time, but it is with Shannon's work, in the 40s decade, when they have been formalized and given a meaning. Shannon defined two measures associated with information, that will be the basic definitions in this project. On the one hand, Shannon defined *entropy*, starting from the definition that Hartley had previously given in the thermodynamics field, as a measure of the "quantity of information" that a random variable provides about itself. Formally, for a discrete random variable X with probability mass function $p(x)$, entropy, $H(X)$, is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

On the other hand, he defined the *mutual information*, a measure of the quantity of information which a random variable provides about another random variable. Analytically, for two discrete random variables X and Y with joint probability mass function $p(x, y)$ and marginals probability mass functions $p(x), p(y)$ respectively, mutual information, $I(X; Y)$, is defined as

$$I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}.$$

These measures can be defined also in the continuous case and, moreover, there are more information measures, for example, relative entropy, proposed by Kullback.

From a statistical point of view, we can see these measures related to information as dependency measures between two random variables. In

fact, if we compare mutual information with Pearson's correlation coefficient, for example, we will see that the first one gives us more information than the second one, because it does not measure a specific type of dependency and it characterizes independence.

In practice, it is not always possible to calculate entropy or mutual information from a random variable, because it is necessary to know its probability mass function or probability density function. In this context, it appears the problem about how to estimate entropy or mutual information from observations of that random variable. The problem of obtaining precise estimations of entropy and mutual information is important because of its several applications. This problem constitutes the core objective of this project.

Obtaining entropy or mutual information is useful not just in the information theory field or in thermodynamics, but also in many other fields such as economy, where it can be used to study economic growth, or biology, where it can be used to predict the secondary structure of RNA. In the machine learning field, estimates of entropy are used to measure how good a model is or to estimate its hiperparameters. Mutual information can be used as a rule to feature selection, clustering analysis or causality detection.

In this project, we consider the objective of analysing the notions of entropy and mutual information, some related concepts, such as conditional entropy, and the properties of these notions, in addition to the relations between the different concepts. Since entropy and mutual information depend on the probability mass function or probability density function of the random variables, which are not always known, we will focus on analyzing how to estimate them. This analysis will be developed in two levels: the theoretical level, in which we will compare different proposals of estimators, and the practical level, in which we will compare some implementations of the estimations studied in the theory. In the theoretical level, we will briefly introduce binned estimators and we will describe estimators of type k nearest neighbours and also estimators of type Kozachenko - Leonenko. These estimators will allow us, for observations of a random variable that we do not know its theoretical distribution, approximate its entropy and mutual information. In the practical level, we will analyse experimentally two implementations of some of the estimators studied theoretically and we will make a comparative analysis between them. This analysis will consider the quality of the solution through the observation of the error and by the difference between mean performance provided by the *T-Test*. The efficiency in terms of the execution time through profiling by developing a performance analysis is also provided.

After the theoretical study and the practical experimentation we have deduced some conclusions. The analysis of the performance of the estimators when data came from a normal distribution allows us to affirm that the errors in the estimators grow with the dimension of the data. When we estimate entropy, the error obtained by the implementation [12] is slightly smaller than the error obtained by the implementation [35]. However, when we estimate mutual information, the errors of both estimations are similar.

The realization of the *T-Test* on the entropy implementations allow us to conclude that the performance of the estimators is similar for small dimensions, but as dimension grows, the differences between the estimators become significant. In the case of entropy, the tests made comparing the estimators with the theoretical estimator for data following a normal distribution allow us to conclude that the estimator [12] presents smaller errors and, therefore, would be preferable. In the case of mutual information, however, we can not conclude anything in this sense, because the errors obtained by both estimators were similar.

Paying attention to the efficiency, the quickest implementation for calculating entropy is [35], except when the number of observations and the dimension are large, in which case it is faster to use the implementation [12]. In the study of mutual information, if we had to choose the quickest version it will depend on the size of the sample and its dimension. For small samples, the implementation [35] is faster, just like for medium-sized samples and large dimension. In the rest of situations, large samples and medium-sized samples with small and medium-sized dimensions are the fastest in the implementation [12]. We notice that for large dimensions and large sample sizes the execution times increase considerably. Times near to some seconds of execution when the dimension is small, became times between 45 minutes and one hour for large dimensions and large sample sizes. It is necessary, therefore, to use parallelism on execution level if we would like to obtain results in acceptable running times.

KEY WORDS: entropy, mutual information, estimators based on the k nearest neighbours, estimators of type Kozachenko - Leonenko, Python.

ÍNDICE GENERAL

1

INTRODUCCIÓN

Este trabajo se desarrolla en el contexto de conceptos asociados a la teoría de la información, como son la entropía y la información mutua. Se estudiarán sus propiedades y algunas propuestas sobre cómo estimarlos. Además, se analizarán desde un punto de vista práctico algunas implementaciones de estos estimadores, profundizando en la calidad de las estimaciones así como en el rendimiento computacional de las mismas.

1.1 MOTIVACIÓN Y ANTECEDENTES

Es en el ámbito de la teoría de la información donde surgen conceptos relativos a la información que nos proporcionan las variables aleatorias. La teoría de la información busca límites teóricos a la capacidad de intercambiar mensajes, así como desarrollar esquemas que alcancen una capacidad óptima para el intercambio de mensajes. Algunos autores consideran la teoría de la información como una rama de la teoría de la probabilidad, aunque también se puede ver como una rama de la teoría de los sistemas dinámicos [16].

Aunque ya existían conceptos relacionados con la información, es con los trabajos de Shannon en la década de los 40 cuando realmente se formalizan y se les aporta significado [32]. Shannon define dos medidas asociadas a la información, que serán las definiciones centrales de este trabajo. Por un lado, define la *entropía*, a partir de la definición que Hartley había dado en el campo de la termodinámica [6], como una medida de la "cantidad de información" que una variable aleatoria aporta sobre sí misma. Formalmente, para una variable aleatoria discreta X con función masa de probabilidad $p(x)$ se define la entropía, $H(X)$, como

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

Por otro lado, define la *información mutua*, una medida de la cantidad de información que una variable aleatoria nos proporciona sobre otra. Analíticamente, para dos variables aleatorias X e Y discretas con función masa de probabilidad conjunta $p(x, y)$ y funciones masa de probabilidad marginales $p(x), p(y)$ respectivamente, se define la información mutua $I(X; Y)$, como

$$I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}.$$

Estas medidas se pueden definir también en el caso continuo y, además, no son las únicas, podemos encontrar también la entropía relativa propuesta por Kullback, por ejemplo [8].

Desde un punto de vista estadístico, podemos ver estas medidas asociadas a la información como medidas de dependencia entre dos variables. De hecho, si comparamos la información mutua con el coeficiente de correlación de Pearson, por ejemplo, veremos que la primera nos proporciona más información que la segunda, ya que no mide un tipo concreto de dependencia y caracteriza la independencia.

En la práctica, no siempre es posible calcular la entropía o información mutua de una variable aleatoria, puesto que es necesario conocer su función masa de probabilidad o función de densidad. Por ello, surge el problema de cómo estimarla a partir de una realización muestral de dicha variable. El problema de obtener estimaciones precisas de la entropía y la información mutua es importante debido a las múltiples aplicaciones que hacen uso de ellas. Este problema constituye uno de los objetivos centrales de este proyecto.

Calcular la entropía o la información mutua resulta de interés no solo en el ámbito de la teoría de la información o en la termodinámica, ya que se les ha encontrado utilidad en numerosos campos como la economía, donde se puede utilizar para estudiar el crecimiento económico [22], o la biología, para predecir la estructura secundaria del ARN [11]. En el campo del aprendizaje automático se utilizan estimaciones de la entropía para medir la bondad de un modelo [15] o para estimar sus hiperparámetros. La información mutua se puede utilizar como criterio para seleccionar variables [10], hacer un análisis clúster [4] o detectar causalidades [17].

1.2 OBJETIVOS

1. *Revisión teórica sobre entropía, información mutua y estimadores de las mismas*.

   En este trabajo nos planteamos el objetivo de analizar las nociones de entropía e información mutua, algunos conceptos relacionados, como la entropía condicionada, y sus propiedades más importantes, además de las relaciones entre los diferentes conceptos. Puesto que estos conceptos dependen de la función masa de probabilidad o función de densidad de las variables aleatorias, que no siempre resulta conocida, se continuará analizando cómo estimarlas. Este análisis se desarrollará en dos niveles: a nivel teórico, estudiando diferentes propuestas de estimadores, y a nivel práctico, comparan-

do varias implementaciones de los mismos. En el nivel teórico, se introducirán brevemente los estimadores basados en particiones y se desarrollarán estimadores basados en los k vecinos más cercanos y de tipo de Kozachenko - Leonenko [19]. Estos estimadores nos permitirán, para una observación de una variable de la que no conocemos su distribución teórica, aproximar la entropía y la información mutua.

2. *Analizar y comparar implementaciones de estimadores de la entropía y de la información mutua.*

   El segundo objetivo a alcanzar se corresponde con el análisis de los estimadores desde un punto de vista práctico. En este nivel, se analizarán experimentalmente implementaciones de algunos de los estimadores estudiados teóricamente y se realizará una análisis comparativo entre ellas. Este análisis contemplará la calidad de la solución a través de la observación del error, diferencia entre comportamientos medios mediante *T-Test* y el rendimiento en términos de tiempo de ejecución mediante *profiling*.

Esta memoria se ha estructurado en cuatro capítulos, este primer capítulo introductorio y tres capítulos adicionales.

En el Capítulo 2 se estudian los conceptos básicos necesarios para el desarrollo del proyecto. Entre otras, se presenta la definición de estimador y sus propiedades, se describen algunas distribuciones que se usarán a lo largo de este trabajo de fin de grado y se definen dos medidas de la dependencia como motivación para la definición de la información mutua. Asimismo, se presentan los conceptos de teoría de la información a estudiar: entropía e información mutua, así como una serie de propiedades y relaciones entre ellas. Este estudio se realizará tanto para el caso discreto como para el continuo. Se continúa estudiando estimadores de los conceptos de entropía e información mutua. Se estudiará brevemente un estimador basado en particiones y en más profundidad un estimador de la entropía basado en los k vecinos más cercanos. También se estudiarán estimadores del tipo de Kozachenko - Leonenko, tanto para la entropía como para la información mutua. Para cerrar este capítulo se analizarán dos implementaciones en Python de los estimadores estudiados anteriormente: las implementaciones presentadas en [12] y [35].

En el Capítulo 3 se establece la metodología utilizada para la validación computacional de los conceptos teóricos y el estudio experimental de los estimadores. Se describen brevemente las herramientas a utilizar, el lenguaje de programación y las bibliotecas utilizadas, centrándonos posteriormente en aquellas necesarias para llevar a cabo el análisis de rendimiento. Durante el estudio de las implementaciones adquiere interés conocer cómo funciona el paralelismo en Python y las herramientas

que proporciona la biblioteca `scikit-learn` para calcular los vecinos más cercanos. Además, el cálculo de los vecinos más cercanos resultará fundamental para realizar las estimaciones, por ello, se describen las clases en que la biblioteca `scikit-learn` tienen implementado este algoritmo. Se discuten los fundamentos teóricos del *T-Test* que será posteriormente incluido como parte de la metodología de análisis y validación computacional. La última etapa del análisis de las implementaciones es el análisis de rendimiento, se incluye en este capítulo una explicación de las bibliotecas utilizadas para la realización del mismo.

El Capítulo 4 recoge los resultados derivados del análisis experimental desarrollado. Esta experiencia computacional, en primer lugar, analiza el comportamiento de los estimadores cuando los datos siguen una distribución normal, conocida teóricamente. Continúa realizando un *T-Test* para saber si las diferencias en media de los estimadores son estadísticamente significativas. Asimismo, se desarrolla el análisis de rendimiento de las distintas versiones desarrolladas intentando extraer conclusiones acerca de los requerimientos computacionales de cada método.

La memoria finaliza con un apartado de conclusiones, así como un apartado en el que se especifica el presupuesto necesario para su desarrollo.

Del análisis experimental se concluye que los estimadores estudiados incrementan sus errores al aumentar la dimensión de los datos de muestra. Además, el aumento de la dimensión, así como el aumento del tamaño de muestra, provoca un incremento notable en los tiempos de ejecución. Tras el *T-Test* concluimos que las implementaciones de la entropía en media resultan similares para dimensiones pequeñas, pero cuando esta aumenta las diferencias entre ellas también lo hacen, preferiremos en este caso la implementación [12] de la entropía, pues es la que obtiene menores errores en la comparación con una variable aleatoria con distribución normal. En el caso de la información mutua, las diferencias entre los estimadores resultan significativas en todos los casos, pero las comparaciones con variables aleatorias con distribución normal no nos permiten decantarnos por ninguno. Atendiendo a los tiempos de ejecución, la elección de la implementación dependerá tanto de la dimensión como del tamaño de muestra.

2

FUNDAMENTOS

2.1 CONCEPTOS PREVIOS

En esta sección se incluyen algunas definiciones y resultados sobre probabilidad y estadística estudiados a lo largo del grado que utilizamos en el cuerpo del trabajo.

Definición 2.1.1 (σ-álgebra)**.** Sea Ω un conjunto no vacío. Una σ-álgebra $\mathcal{A}$ sobre Ω es una familia de subconjuntos de Ω que verifica:

- el conjunto vacío, $\emptyset$, pertenece a $\mathcal{A}$,
- si A pertenece a $\mathcal{A}$, entonces su complementario $\Omega \setminus A$ también pertenece a $\mathcal{A}$,
- si tomamos una serie de conjuntos $\{A_i\}_{i\in\mathbb{N}}$ de $\mathcal{A}$, entonces su unión numerable $\cup_{i\in\mathbb{N}} A_i$ también pertenece a $\mathcal{A}$.

Ejemplo 2.1.1. La familia de conjuntos Borel $\mathcal{A} = \mathcal{B}(\mathbb{R}^n)$ es una σ-álgebra sobre $\mathbb{R}^n$.

Definición 2.1.2 (Medida de probabilidad)**.** Sea $\mathcal{A}$ una σ-álgebra sobre un conjunto no vacío Ω. Una medida de probabilidad P es una función $P : \mathcal{A} \to [0, 1]$ tal que

- $P[\Omega] = 1$,
- si $\{A_i\}_{i\in\mathbb{N}}$ son conjuntos de $\mathcal{A}$ disjuntos dos a dos, entonces

$$P\left[\bigcup_{i\in\mathbb{N}} A_i\right] = \sum_{i\in\mathbb{N}} P[A_i].$$

La tupla $(\Omega, \mathcal{A}, P)$ se llama espacio de probabilidad. Los conjuntos de $\mathcal{A}$ se llaman sucesos.

Definición 2.1.3 (Variable aleatoria)**.** Sea $(\Omega, \mathcal{A}, P)$ un espacio de probabilidad. Un variable aleatoria $X = (X_1, \dots, X_n)$ es una función medible

$$X : (\Omega, \mathcal{A}) \to (\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n)).$$

La condición de medibilidad quiere decir que $X^{-1}(B) \in \mathcal{A} \quad \forall B \in \mathcal{B}(\mathbb{R}^n)$.

Cuando $n = 1$ diremos que la variable aleatoria es unidimensional, mientras que cuando $n > 1$ diremos que la variable aleatoria es multidimensional, en ocasiones también denominada, vector aleatorio. En el caso de la variable aleatoria multidimensional podremos querer información sobre ella misma, $X = (X_1, \dots, X_n)$, lo que llamaremos variable aleatoria conjunta, o sobre alguna de las variables aleatoria X_i, que denominaremos variable aleatoria marginal.

Cada variable aleatoria nos da una medida de probabilidad, su probabilidad inducida.

Definición 2.1.4 (Probabilidad inducida)**.** La probabilidad inducida por un variable aleatoria $X = (X_1, \dots, X_n)$, o distribución de X, se define como la función

$$P_X[B] := P[X^{-1}(B)],$$

para todo $B \in \mathcal{B}(\mathbb{R}^n)$.

Nota: en adelante por comodidad, omitiremos el subíndice X cuando no haya lugar a confusión.

Definición 2.1.5 (Función de distribución)**.** La función de distribución de una variable aleatoria unidimensional X es la aplicación $F_X : \mathbb{R} \to [0, 1]$ definida como:

$$F_X(x) = P[X \leq x].$$

Para una variable aleatoria multidimensional, $X = (X_1, \dots, X_n)$, la función de distribución es $F_X : \mathbb{R}^n \to [0, 1]$ y está dada por

$$F_X(x_1, \dots, x_n) = P[X_1 \leq x_1, \dots, X_n \leq x_n].$$

Proposición 2.1.2. La función de distribución F_X de la variable aleatoria X es no decreciente, continua a la derecha y $\lim_{x \to -\infty} F_X(x) = 0$, $\lim_{x \to +\infty} F_X(x) = 1$.

Cuando la imagen de la variable aleatoria X sea numerable, diremos que la variable aleatoria es *discreta*, será descrita por la función masa de probabilidad que asigna una probabilidad a cada valor de la imagen de X. Por otro lado, cuando la imagen de la variable aleatoria X sea infinita no numerable, diremos que la variable aleatoria es *continua*. Si además puede ser descrita por una función de densidad, será *absolutamente continua*.

Si la variable aleatoria X es discreta, la función de distribución tendrá puntos de discontinuidad en los valores de la imagen de X. Mientras que si la variable aleatoria es continua, su función de distribución será continua.

Definición 2.1.6 (Función de densidad y función masa de probabilidad)**.** X es una variable aleatoria con distribución absolutamente continua si existe una función integrable Lebesgue $f_X : \mathbb{R}^n \to \mathbb{R}$ tal que para cualquier conjunto Borel $B \subset \mathbb{R}$

$$P[X \in B] = \int_B f_X(x)dx,$$

f_X se llama función de densidad de X. Si hay una secuencia (finita o numerable) de vectores $x_1, x_2, ...$ tal que para cualquier conjunto Borel $B \subset \mathbb{R}^n$

$$P[X \in B] = \sum_{x_i \in B} P[X = x_i],$$

entonces decimos que X tiene una distribución discreta que toma los valores $x_1, x_2, ...$ y cuya función masa de probabilidad es $P[X = x_i] = p(x_i)$ en x_i.

Definición 2.1.7 (Soporte)**.** Llamamos soporte de una variable aleatoria continua X al conjunto de puntos donde $f(x) > 0$, lo notaremos $sop(X)$.

Definición 2.1.8 (Probabilidad condicionada)**.** Para cualesquiera dos sucesos $A, B \in \mathcal{A}$ tal que $P[B] \neq 0$, la probabilidad condicionada de A por B se define como

$$P[A|B] = \frac{P[A \cap B]}{P[B]}.$$

Definición 2.1.9 (Función de densidad condicionada)**.** Para cualesquiera dos variables aleatorias X, Y con funciones de densidad f_X y f_Y respectivamente y función de densidad conjunta f, la densidad de X condicionada a que $Y = y$ viene dada por

$$f_X(x|Y = y) = \frac{f(x, y)}{f_Y(y)}.$$

Definición 2.1.10 (Sucesos independientes)**.** Los sucesos $A_1, ..., A_n \in \mathcal{A}$ se dicen independientes si

$$P\left[\bigcap_{j=1}^{k} A_{i_j}\right] = \prod_{j=1}^{k} P\left[A_{i_j}\right],$$

para cualesquiera índices $1 \leq i_1 \leq \cdots \leq i_k \leq n$.

Definición 2.1.11 (Variables aleatorias independientes)**.** Dos variables aleatorias X e Y son independientes si para cualesquiera conjuntos Borel $A, B \in \mathcal{B}(\mathbb{R}^n)$ los sucesos $[X \in A]$ e $[Y \in B]$ son independientes.

De forma general, k variables aleatorias $X_1, \dots, X_k$ serán independientes si para cualesquiera conjuntos de Borel $B_1, \dots, B_k \in \mathcal{B}(\mathbb{R}^n)$ los sucesos $[X_1 \in B_1], \dots, [X_k \in B_k]$ son independientes.

Proposición 2.1.3 (Caracterizaciones de independencia)**.** Dada una variable aleatoria $X = (X_1, \dots, X_n)$ sus componentes son independientes si, y solo si, se da alguna de las siguientes condiciones:

- La función de distribución verifica $F_X(x) = F_{X_1}(x_1) \cdot \dots \cdot F_{X_n}(x_n)$ para todo $x \in \mathbb{R}^n$.
- X tiene función de densidad y $f_X(x) = f_{X_1}(x_1) \cdot \dots \cdot f_{X_n}(x_n)$ para todo $x \in \mathbb{R}^n$, salvo, a lo sumo, en un conjunto de medida nula.

Definición 2.1.12 (Esperanza)**.** La esperanza de una variable aleatoria X unidimensional se define como:

$$\mu = \mathbb{E}[X] = \int_\Omega X(\omega) dP(\omega) = \int_{\mathbb{R}} x dF(x).$$

En el caso discreto, si la variable aleatoria X tiene un número finito de posibles salidas $x_1, \dots, x_k$ cada una con probabilidad $p_1, \dots, p_k$ respectivamente, la esperanza es

$$\mathbb{E}[X] = \sum_{i=1}^{k} x_i p_i.$$

En el caso de que X sea una variable aleatoria absolutamente continua y admita función de densidad, $f(x)$, la esperanza se calcula

$$\mathbb{E}[X] = \int_{\mathbb{R}} x f(x) dx.$$

Si $X = (X_1, \dots, X_n)^T$ es una variable aleatoria multidimensional. Se define la esperanza de X como:

$$\mu_X = \mathbb{E}[X] = \begin{bmatrix} \mathbb{E}[X_1] \\ \vdots \\ \mathbb{E}[X_n] \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix},$$

siempre que existan todas las esperanzas unidimensionales.

Proposición 2.1.4 (Propiedades de la esperanza)**.** Sean $X, X_1, \dots, X_n$ variables aleatorias con funciones de densidad $f_X, f_{X_1}, \dots, f_{X_n}$ respectivamente y $g, g_1, \dots, g_n : \mathbb{R} \to \mathbb{R}$ funciones integrables Lebesgue, entonces se cumplen las siguientes propiedades de la esperanza:

- $g(X)$ es también una variable aleatoria y su esperanza será

$$\mathbb{E}[g(X)] = \int g(x) f_X(x) dx.$$

  Esta propiedad se conoce como *ley del estadístico inconsciente*.

- $\mathbb{E}\left[\sum_{j=1}^{k} c_j g_j(X)\right] = \sum_{j=1}^{k} c_j \mathbb{E}\left[g_j(X)\right]$.
- Si $X_1, \dots, X_n$ son independientes, entonces

$$\mathbb{E}\left[\prod_{i=1}^{n} X_i\right] = \prod_{i=1}^{k} \mathbb{E}[X_i].$$

Teorema 2.1.5 (Desigualdad de Jensen)**.** Si g es una función integrable Lebesgue, convexa y X es una variable aleatoria, entonces

$$g\left(\mathbb{E}[X]\right) \leq \mathbb{E}\left[g(X)\right]$$

dándose la igualdad si, y solo si, g es una función afín.

Definición 2.1.13 (Esperanza condicionada)**.** Se define la esperanza condicionada de la variable aleatoria Y dado X como la variable aleatoria $\mathbb{E}[Y|X]$ tal que

$$\mathbb{E}[Y|X = x] = \int y f_Y(y|X = x) dy.$$

Definición 2.1.14 (Varianza y matriz de covarianzas)**.** Sea X una variable aleatoria unidimensional de esperanza μ_X, definimos su varianza como

$$\sigma_X^2 = \text{Var}(X) = \mathbb{E}\left[(X - \mu_X)^2\right].$$

Se define la matriz de covarianzas de una variable aleatoria multidimensional $X = (X_1, \dots, X_n)^T$ como

$$\Sigma = Cov(X) = \mathbb{E}\left[(X - \mu_X)(X - \mu_X)^T\right] = \begin{bmatrix} \sigma_{11} & \dots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \dots & \sigma_{nn} \end{bmatrix},$$

donde $\sigma_{ij} = Cov(X_i, X_j) = \mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right] = \sigma_{ji}$ es la covarianza de las variables aleatorias X_i y X_j. La podremos definir cuando existan todas las covarianzas.

2.1.1 *Estimadores*

En esta sección recordaremos algunos conceptos sobre estimación puntual que utilizaremos más adelante.

Definición 2.1.15 (Muestra aleatoria simple)**.** Una muestra aleatoria simple de tamaño n de una variable aleatoria X con distribución teórica F, son n variables aleatorias $(X_1, \dots, X_n)$, independientes e idénticamente distribuidas, todas con la misma distribución F.

En un amplio número de estudios estadísticos la distribución teórica F de la variable aleatoria a estudiar es desconocida. Notamos $\mathcal{F}$ a la familia de distribuciones candidatas a ser realmente la distribución (se obtiene a partir de información previa). En estadística paramétrica se conoce la forma funcional de F pero se desconocen uno o algunos parámetros de la misma. Para tratar de obtener información sobre este parámetro desconocido a partir de una muestra utilizaremos los *estimadores*. En estadística no paramétrica se desconoce la forma funcional de F, y se trata de obtener aproximaciones de la distribución o de algunos de sus momentos a partir de una muestra.

Definición 2.1.16 (Estadístico)**.** Se denomina estadístico a cualquier función medible de una muestra aleatoria simple $T(X_1, \dots, X_n)$, donde $T : \mathbb{R}^n \to \mathbb{R}$ es una determinada función medible.

Ejemplo 2.1.6. Algunos estadísticos habituales son:

- $T(X_1, \dots, X_n) = \frac{1}{n} \sum_{i=1}^{n} X_i$, media aritmética o media muestral, denotada por $\overline{X}$.
- $T(X_1, \dots, X_n) = \frac{1}{n} \sum_{i=1}^{n} \left(X_i - \overline{X}\right)^2$, varianza muestral, notada s^2.
- $T(X_1, \dots, X_n) = \text{mín}(X_1, \dots, X_n)$, el menor valor muestral.
- $T(X_1, \dots, X_n) = \text{máx}(X_1, \dots, X_n)$, el mayor valor muestral.

Definición 2.1.17 (Estimador)**.** Un estimador es un estadístico cuyos valores se utilizan para obtener información (estimación puntual) de un parámetro desconocido θ, lo notaremos $\widehat{\theta}(X_1, \dots, X_n)$.

Como comentamos anteriormente, se usan los valores del estimador para obtener información de algún parámetro de la población. Las siguientes propiedades nos informan de la "calidad" de la estimación y permiten comparar diferentes estimadores de un mismo parámetro.

Definición 2.1.18 (Estimador insesgado)**.** Si $\hat{\theta}$ es un estimador del parámetro θ, la diferencia

$$\text{bias}\left(\hat{\theta}\right) = \mathbb{E}\left[\hat{\theta}\right] - \theta$$

se denomina sesgo del estimador $\hat{\theta}$ como estimador de θ. Cuando el sesgo es nulo para cualquier valor del parámetro, es decir, si

$$\mathbb{E}\left[\hat{\theta}\right] = \theta \quad \forall \theta \in \Theta,$$

el estimador $\hat{\theta}$ se dice insesgado para θ.

Ejemplo 2.1.7. Sea $X_1, \dots, X_n$ una muestra aleatoria de una variable aleatoria X y $\mu = \mathbb{E}[X]$. Si el parámetro de interés es la media, μ, podemos utilizar la media muestral, $\overline{X}$, para estimarlo. Calculamos su sesgo,

$$\text{bias}\left(\overline{X}\right) = \mathbb{E}\left[\overline{X}\right] - \mu = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] - \mu = \mu - \mu = 0,$$

hemos comprobado que la media muestral es un estimador insesgado para μ.

Definición 2.1.19 (Eficiencia)**.** Si $\hat{\theta}_1, \hat{\theta}_2$ son dos estimadores del parámetro θ, diremos que $\hat{\theta}_1$ es más eficiente que $\hat{\theta}_2$ si

$$\text{Var}\left(\hat{\theta}_1\right) < \text{Var}\left(\hat{\theta}_2\right).$$

El tamaño de la muestra será un factor que influya en la estimación. Por ello, otra propiedad deseable en un estimador es la consistencia. Para definir esta propiedad es necesario conocer la definición de convergencia en probabilidad, una de las diferentes nociones de convergencia de variables aleatorias.

Definición 2.1.20 (Convergencia en probabilidad)**.** Una sucesión de variables aleatorias $\{X_n\}$ converge en probabilidad a la variable X si para todo $\varepsilon > 0$,

$$\lim_{n\to\infty} P\left[|X_n - X| > \varepsilon\right] = 0.$$

Lo notaremos $X_n \xrightarrow{P} X$.

Definición 2.1.21 (Estimador consistente)**.** Sea $\hat{\theta}_n$ un estimador del parámetro θ para una muestra de tamaño n. Diremos que el estimador $\hat{\theta}_n$ es consistente si, cuando $n \to \infty$, se verifica

$$\hat{\theta}_n \xrightarrow{P} \theta.$$

Lema 2.1.8. Sea $\hat{\theta}_n$ un estimador del parámetro θ para una muestra de tamaño n. Si bias $\left(\hat{\theta}_n\right) \xrightarrow{n\to\infty} 0$ y Var $\left(\hat{\theta}_n\right) \xrightarrow{n\to\infty} 0$, entonces $\hat{\theta}_n \xrightarrow{P} \theta$, esto es, el estimador $\hat{\theta}_n$ es consistente.

Cuando bias $\left(\hat{\theta}_n\right) \xrightarrow{n\to\infty} 0$ diremos que el estimador $\hat{\theta}_n$ es asintóticamente insesgado.

Ejemplo 2.1.9. Sea X una variable aleatoria con media μ y varianza σ^2. Hemos visto que la media muestral es un estimador insesgado de μ. Veamos ahora el comportamiento asintótico de la varianza de la media muestral.

$$\text{Var}(\overline{X}) = \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}(X_i) = \frac{\sigma^2}{n} \xrightarrow{n\to\infty} 0.$$

En el Ejemplo 2.1.7 habíamos visto que el sesgo de $\overline{X}$ era nulo para cualquier tamaño de muestra n. Concluimos, aplicando el Lema 2.1.8, que $\overline{X}$ es un estimador consistente de la media de la variable X.

La media muestral no es solo un estimador insesgado y consistente de la media teórica de una variable aleatoria X. La ley (débil) de los grandes números prueba que este estimador converge en probabilidad a la media teórica.

Teorema 2.1.10 (Ley (débil) de los grandes números)**.** Sean $X_1, \dots, X_n$ variables aleatorias independientes e idénticamente distribuidas con media $\mu < \infty$, entonces la media muestral

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

converge en probabilidad a μ, esto es,

$$\overline{X}_n \xrightarrow{P} \mu.$$

2.1.2 *Algunas distribuciones discretas y continuas*

Durante el desarrollo del trabajo se utilizarán algunas distribuciones que se presentan a continuación.

Definición 2.1.22 (Distribución uniforme discreta)**.** Sea X una variable aleatoria discreta con valores $\{x_1, \dots, x_n\}$. X tiene una distribución uniforme si $P[X = x_i] = \frac{1}{n}$ para $i = 1, \dots, n$. Es decir, la función masa de probabilidad $p(x_i) = \frac{1}{n}$ para $i = 1, \dots, n$.

Ejemplo 2.1.11. Si tiramos un dado no cargado hay seis posibles valores: 1, 2, 3, 4, 5 y 6; cada vez que lanzamos el dado hay probabilidad $\frac{1}{6}$ de que salga cualquiera de ellos.

Definición 2.1.23 (Distribución uniforme continua)**.** Una variable aleatoria X que puede obtener cualquier valor en el intervalo $[a, b]$ con la misma probabilidad tendrá una distribución uniforme. Su función de densidad vendrá dada por

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{si } x \in [a, b] \\ 0, & \text{en otro caso} \end{cases}.$$

Definición 2.1.24 (Distribución normal unidimensional)**.** Una variable aleatoria unidimensional seguirá una distribución normal, o distribución gaussiana, si su función de densidad $f : \mathbb{R} \to \mathbb{R}$ viene dada por

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Generalizamos esta definición para el caso multivariante.

Definición 2.1.25 (Distribución normal multivariante)**.** Una variable aleatoria sigue una distribución normal multivariante si cualquier combinación de sus componentes tiene una distribución normal univariante. Su función de densidad $f : \mathbb{R}^n \to \mathbb{R}$ viene dada por

$$f(x) = \frac{1}{\left(\sqrt{2\pi}\right)^n |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)},$$

donde $\mu \in \mathbb{R}^n$ y Σ es una matriz $n \times n$ definida positiva.

La siguiente distribución, la distribución multinomial, modela la probabilidad de un número determinado de éxitos para n experimentos independientes, donde hay k posibles éxitos, cada uno de ellos con probabilidad de éxito fija.

Definición 2.1.26 (Distribución multinomial)**.** Sean n experimentos independientes donde cada uno produce uno de los sucesos $S_1, \dots, S_k$ para $k \geq 2$ (estos sucesos son mutuamente excluyentes) y cada suceso S_i ocurre con probabilidad p_i ($p_1 + \cdots + p_k = 1$). Definimos las variables aleatorias $X_i,\ i = 1, \dots, k$ como el número de experimentos en el que ocurre S_i. Entonces $X = (X_1, \dots, X_k)$ tiene una distribución multinomial con parámetros n y $p = (p_1, \dots, p_k)$. Su función masa de probabilidad viene dada por

$$f(x_1, \dots, x_k; n, p_1, \dots, p_k) = P[X_1 = x_1 \text{ y } \dots \text{ y } X_k = x_k] = \begin{cases} \frac{n!}{x_1! \dots x_k!} p_1^{x_1} \dots p_k^{x_k}, & \text{si } \sum_{i=1}^k x_i = n \\ 0, & \text{en otro caso} \end{cases},$$

donde $x_1, \dots, x_k$ son enteros no negativos.

Ejemplo 2.1.12. Si tenemos un dado de k lados y queremos calcular cuántas veces sale cada uno de ellos en n lanzamientos, tendremos una distribución multinomial.

El valor esperado de observar la salida i en n sucesos es $\mathbb{E}(X_i) = np_i$, luego $\mathbb{E}(X) = np$.

Veamos cómo son los elementos de la matriz de covarianzas. Los elementos de la diagonal son la varianza de una variable aleatoria con distribución binomial, luego $\text{Var}(X_i) = np_i(1 - p_i)$. Las covarianzas de X_i, X_j para $i \neq j$ son $Cov(X_i, X_j) = -np_ip_j$. Por tanto,

$$\text{Var}(X) = n[\text{diag}(p) - pp^T],$$

donde $\text{diag}(p)$ representa la matriz diagonal que tiene el elemento p_i en la posición i, i.

2.1.3 *Estudio de la dependencia*

Dadas dos variables aleatorias nos podemos preguntar si son o no independientes, es decir, si la realización de una de ellas afecta o no a la realización de la otra. En caso de ser dependientes nos gustaría medir qué tipo de relación existe entre ellas. Hay numerosos métodos que nos permiten realizar esta medida, por ejemplo, el *coeficiente de correlación de Pearson* o el *coeficiente de correlación de Spearman*.

Medir la dependencia entre variables puede ser útil porque nos indica relaciones entre variables que se pueden aprovechar para hacer predicciones de una a partir de la otra. Encontramos infinidad de aplicaciones en

el mundo real a conocer una dependencia entre variables. Por ejemplo, para conocer qué efecto tiene un fenómeno sobre otro. También, si conocemos la dependencia entre variables, podemos utilizar aquellas de las que tenemos información más precisa o fiable para explicar aquellas de las que no.

Definición 2.1.27 (Coeficiente de correlación de Pearson)**.** Se define el coeficiente de correlación de Pearson para un par de variables aleatorias (X, Y) como

$$r_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right]}{\sigma_X \sigma_Y}.$$

El mismo coeficiente para una realización muestral $\{(x_1, y_1), \dots, (x_n, y_n)\}$ se define como

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2(y_i-\bar{y})^2}},$$

donde $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ la media muestral, de forma análoga para $\bar{y}$.

Este coeficiente mide la correlación lineal entre las variables X, Y. Su valor se encuentra entre $+1$ y -1, donde $+1$ indica una correlación lineal positiva total, 0 que no hay correlación lineal y -1 correlación lineal total negativa. Si el coeficiente de correlación de Pearson de dos variables aleatorias es 0, sabremos que no existe una dependencia lineal entre ellas, pero no podremos garantizar la independencia, pues podría existir otro tipo de dependencia entre ellas.

Este coeficiente es paramétrico, depende de las distribuciones marginales, el que presentamos a continuación es no paramétrico.

El coeficiente de correlación de Spearman mide la relación entre dos variables más allá de si es lineal o no. Atiende a si ambas variables tienen la misma monotonía, esto es, si a medida que los valores de una de las variables crecen, los de la otra también lo hacen (o al revés). Para ello, tendremos que ordenar los datos de cada variable de menor a mayor.

Definición 2.1.28 (Coeficiente de correlación de Spearman)**.** Sea (X, Y) una variable aleatoria bidimensional con distribución conjunta $F(x, y)$ y distribuciones marginales $F_X(x)$, $F_Y(y)$ respectivamente. Sean ahora $(X_1, Y_1) \sim F$ y $(X_2, Y_2) \sim F_X \cdot F_Y$ independientes entre sí, entonces el coeficiente ρ de Spearman se define como

$$\rho_{XY} = 3\left(P\left[(X_1-X_2)(Y_1-Y_2) > 0\right] - P\left[(X_1-X_2)(Y_1-Y_2) < 0\right]\right).$$

Para una realización $\{(x_1, y_1), \dots, (x_n, y_n)\}$ de dos variables aleatorias X e Y, ordenamos los datos de ambas variables de forma que $R_x =$

$(R_{x_1}, \dots, R_{x_n})$ es el vector de rangos de los x_i y $R_y = (R_{y_1}, \dots, R_{y_n})$ el de los y_i. El coeficiente de correlación de Spearman entre X e Y viene dado por

$$\rho_{XY} = r_{R_x R_y} = \frac{Cov(R_x, R_y)}{\sigma_{R_x} \sigma_{R_y}},$$

donde r denota el coeficiente de correlación de Pearson. Si todos los puntos son distintos, se puede utilizar la fórmula:

$$\rho_{XY} = 1 - \frac{6 \sum_i d_i^2}{n(n^2 - 1)},$$

donde $d_i = R_{x_i} - R_{y_i}$ es la diferencia entre los rangos de x_i e y_i.

Este coeficiente también toma valores entre -1 y $+1$. Si los valores son cercanos a cero la correlación monótona será muy débil, mientras que si son cercanos a $+1$ o -1 será muy fuerte, el signo nos indicará si a medida que una crece la otra decrece o viceversa. Al igual que coeficiente de Pearson, el de Spearman no caracteriza la independencia. Aunque ambos coeficientes, si las variables aleatorias son independientes, valdrán 0.

Así, hemos visto un par de ejemplos de medidas de dependencia que caracterizan adecuadamente tipos concretos de dependencia, pero no caracterizan la independencia.

La información mutua es otra forma de medir la dependencia entre dos variables aleatorias capaz de determinar otros tipos de dependencias, además de las lineales o monótonas. Podríamos decir que es una medida de la dependencia global. Además, esta medida también nos dará información sobre la independencia de las variables aleatorias en estudio (véanse Teorema 2.2.15 y Corolario 2.2.23.1).

2.2 TEORÍA DE LA INFORMACIÓN

El concepto de información es demasiado amplio como para ser recogido en una única definición. Sin embargo, podemos definir el concepto de entropía, que mide la cantidad de información necesaria para describir una variable aleatoria, o el de información mutua, que refleja la información que una variable aleatoria contiene sobre otra.

Intuitivamente, la entropía es una medida de la información o incertidumbre de una variable aleatoria. El término entropía viene del griego, donde significa *transformación* y tiene relevancia también en otras áreas, como en la física donde se define como el logaritmo del cociente entre temperatura final e inicial de un sistema.

La idea de relacionar el número de estados de un sistema con una medida física viene del siglo XIX, en el que Rudolph Clausius sugirió la denominación "entropía" para esta medida. En 1928, Ralph Vinton Lyon Hartley define el término en el contexto de la teoría de la información, básicamente como el logaritmo del tamaño del alfabeto. Aunque no es hasta 1948 cuando Claude Shannon da una definición matemática del concepto de información como lo conocemos hoy en día, sentando las bases de la teoría de la información, que pasó a formar parte de la teoría de la probabilidad. En su concepto de entropía no todos los símbolos de un alfabeto tienen porqué ser equiprobables. Esto nos permite medir la capacidad de comunicación de un canal. Andrei N. Kolmogorov, en la década de los 60, desarrolló su teoría de la complejidad, otra forma de teoría de la información.

A continuación, se explicarán estos conceptos, y algunos relacionados, junto a demostraciones de sus propiedades y algunos ejemplos, tanto para el caso discreto como para el caso continuo. Las principales fuentes consultadas para ello han sido [8] y [16]. Se han estudiado y comprendido los capítulos correspondientes a entropía y entropía diferencial para posteriormente exponer aquí una síntesis de lo aprendido.

2.2.1 *Caso discreto*

Sea una variable aleatoria discreta, X, que toma valores en el alfabeto $\mathcal{X} = \{x_1, \dots, x_N\}$, con función masa de probabilidad $P_X(x_i) = p(x_i)$.

Definición 2.2.1 (Entropía)**.** La entropía de una variable aleatoria discreta X viene dada por

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

En la fórmula anterior tomaremos, por continuidad, $0 \log 0 = 0$. Así, la entropía queda definida por un funcional de la distribución de X, es decir, que no depende de los valores que tome la variable sino de sus probabilidades.

Según cuál sea la base del logaritmo utilizado en la fórmula de la entropía utilizaremos unas unidades u otras. Para $b \in \mathbb{N}$, notaremos $H_b(X) = -\sum_{x \in \mathcal{X}} p(x) \log_b p(x)$. Si el logaritmo tiene base 2, la entropía se medirá en *bits*, mientras que si la base es e, las unidades serán *nats* (unidad natural de información).

Dada la variable aleatoria discreta X con distribución $p(x)$, podemos considerar la variable aleatoria $g(X)$, cuya esperanza, $\mathbb{E}$, viene dada por:

$$\mathbb{E}_p\left[g(X)\right] = \sum_{x\in\mathcal{X}} g(x)p(x).$$

Tomando $g(X) = \log \frac{1}{p(X)}$ obtenemos otra definición de entropía, la usada en termodinámica:

$$H(X) = \mathbb{E}_p\left[\log \frac{1}{p(X)}\right].$$

La fórmula anterior coincide, efectivamente, con la definición de entropía dada:

$$H(X) = \mathbb{E}_p\left[\log \frac{1}{p(X)}\right] = \sum_{x\in\mathcal{X}} \left(\log \frac{1}{p(x)}\right) p(x) = -\sum_{x\in\mathcal{X}} p(x)\log p(x).$$

Lema 2.2.1. La entropía de una variable aleatoria discreta es no negativa, $H(X) \geq 0$.

Demostración.

$$H(X) = -\sum_{x\in\mathcal{X}} \underbrace{p(x)}_{\in[0,1]} \underbrace{\log p(x)}_{\leq 0} \geq 0.$$

□

El siguiente lema, mediante propiedades del logaritmo, nos permitirá cambiar la base con la que calculemos la entropía.

Lema 2.2.2.

$$H_b(X) = (\log_b a)H_a(X).$$

Demostración. Partiremos de la definición alternativa de H_b y usaremos que $\log_b p = \log_b a \ \log_a p$.

$$H_b(X) = \mathbb{E}_p\left[\log_b \frac{1}{p(X)}\right] = \mathbb{E}_p\left[\left(\log_b a\right)\log_a \frac{1}{p(X)}\right] = \left(\log_b a\right) H_a(X).$$

□

En el siguiente ejemplo calculamos la entropía de una variable aleatoria con probabilidades fijadas.

Ejemplo 2.2.3. Sea

$$X = \begin{cases} a, & \text{con probabilidad } \frac{1}{2} \\ b, & \text{con probabilidad } \frac{1}{4} \\ c, & \text{con probabilidad } \frac{1}{8} \\ d, & \text{con probabilidad } \frac{1}{8} \end{cases}.$$

Utilizamos la fórmula de la entropía y obtenemos

$$H(X) = -\left(\frac{1}{2}\log\frac{1}{2} + \frac{1}{4}\log\frac{1}{4} + \frac{1}{4}\log\frac{1}{8}\right) = -\left(-\frac{1}{2} - \frac{1}{2} - \frac{3}{4}\right)$$
$$= 1 + \frac{3}{4} = \frac{7}{4} \text{ bits.}$$

Ejemplo 2.2.4. Sea

$$X = \begin{cases} 1, & \text{con probabilidad } p \\ 0, & \text{con probabilidad } 1-p \end{cases}.$$

Vamos a analizar la entropía de la variable aleatoria X como función de p, es decir, $H : [0,1] \to \mathbb{R}$ estará dada por $H(p)$. Se calcula como sigue:

$$H(p) := H(X) = -\left(p\log p + (1-p)\log(1-p)\right).$$

Cuando $p = 0$ o $p = 1$, $H(p) = 0$, luego no hay incertidumbre, se conoce el valor que tomará la variable X. Calcularemos la primera y segunda derivada de $H(p)$ para conocer su comportamiento en el intervalo $(0,1)$.

$$H'(p) = -\left(\log p + \frac{p}{p} - \log(1-p) - \frac{1-p}{1-p}\right) = -\left(\log\left(\frac{p}{1-p}\right)\right),$$
$$H''(p) = -\left(\frac{1}{p} + \frac{1}{1-p}\right) \leq 0 \quad \forall p \in (0,1).$$

Hemos comprobado que la función $H(p)$ es cóncava. La incertidumbre máxima se alcanzará en $p = \frac{1}{2}$:

$$H\left(\frac{1}{2}\right) = -\left(\frac{1}{2}\log\frac{1}{2} + \frac{1}{2}\log\frac{1}{2}\right) = -\log\frac{1}{2} = 1.$$

La definición de entropía de una variable aleatoria se puede aplicar a un par de variables aleatorias dando lugar a la entropía conjunta.

Definición 2.2.2 (Entropía conjunta)**.** La entropía conjunta, $H(X,Y)$, de un par de variables aleatorias discretas (X,Y) con distribución conjunta $p(x,y)$ se define como

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y),$$

que podemos expresar también como

$$H(X,Y) = -\mathbb{E}\left[\log p(X,Y)\right].$$

Presentamos el siguiente lema que será útil para la demostración de propiedades en lo sucesivo.

Lema 2.2.5 (Desigualdad de Gibbs)**.** Dadas dos funciones masa de probabilidad $\{p_i\}$ y $\{q_i\}$, entonces

$$\sum_i p_i \log \frac{p_i}{q_i} \geq 0,$$

dándose la igualdad si, y solo si, $q_i = p_i$ para todo i.

Demostración. Sea $I = \{i : p_i > 0\}$, entonces

$$-\sum_{i\in I} p_i \log \frac{p_i}{q_i} = \sum_{i\in I} p_i \log \frac{q_i}{p_i},$$

usando $\log x \leq x - 1$, con igualdad si, y solo si, x = 1, obtenemos

$$\sum_{i\in I} p_i \log \frac{q_i}{p_i} \leq \sum_{i\in I} p_i \left(\frac{q_i}{p_i} - 1\right) = \sum_{i\in I} q_i - \underbrace{\sum_{i\in I} p_i}_{=1} \leq 0.$$

Tendremos la igualdad cuando $\frac{q_i}{p_i} = 1$ para todo i.

□

En el siguiente lema recogemos algunas propiedades de la entropía conjunta.

Lema 2.2.6. Sean X, Y dos variables aleatorias discretas con alfabeto $\mathcal{X}, \mathcal{Y}$ respectivamente. Entonces se verifican:

1. $H(X) \leq H(X, Y)$.
2. $H(X, Y) \leq H(X) + H(Y)$. La igualdad se dará si, y solo si, X e Y son independientes.

Demostración.

1. Para cualesquiera $x \in \mathcal{X}, y \in \mathcal{Y}$ se tiene que $\sum_{y\in\mathcal{Y}} p(x, y) = p(x)$ y $p(x, y) \leq p(x)$, luego:

$$\begin{aligned} H(X,Y) &= -\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y) \log p(x,y) \geq -\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y) \log p(x) \\ &= -\sum_{x\in\mathcal{X}} p(x) \log p(x) = H(X). \end{aligned}$$

2. Usando la definición, $\sum_{y\in\mathcal{Y}} p(x,y) = p(x)$ y propiedades del logaritmo:

$$\begin{aligned}
&H(X,Y) - (H(X) + H(Y)) \\
&= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) + \sum_{x\in\mathcal{X}} p(x)\log p(x) + \sum_{y\in\mathcal{Y}} p(y)\log p(y) \\
&= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) + \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) \\
&\quad + \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y) \\
&= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x)p(y)}{p(x,y)} \leq 0,
\end{aligned}$$

donde la última desigualdad viene de aplicar el Lema 2.2.5 sobre las funciones masa de probabilidad $p(x,y)$ y $p(x)p(y)$. Esta última sería la función masa de probabilidad conjunta de dos variables aleatorias X, Y con distribuciones $p(x), p(y)$ independientes. La igualdad ocurre si, y solo si, $p(x)p(y) = p(x,y)$, esto es si X e Y son independientes.

□

Al considerar dos distribuciones distintas, además de su entropía conjunta podemos definir la entropía condicional de una respecto de la otra.

Definición 2.2.3 (Entropía condicional)**.** Si (X,Y) sigue una distribución $p(x,y)$, la entropía condicional de Y con respecto a X, $H(Y|X)$, se define como:

$$\begin{aligned}
H(Y|X) &= \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \\
&= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x) \\
&= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \\
&= -\mathbb{E}\left[\log p(Y|X)\right].
\end{aligned}$$

Teorema 2.2.7 (Regla de la cadena)**.** Si (X,Y) sigue una distribución conjunta $p(x,y)$, entonces se verifica:

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

Demostración. Para la prueba utilizaremos que $p(x,y) = p(x)p(y|x)$,

$$\begin{aligned} H(X,Y) &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\left(p(x)p(y|x)\right) \\ &= -\left(\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) + \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x)\right), \end{aligned}$$

usando que $\sum_{y\in\mathcal{Y}} p(x,y) = p(x)$, obtenemos

$$\begin{aligned} H(X,Y) &= -\sum_{x\in\mathcal{X}} p(x)\log p(x) + H(Y|X) \\ &= H(X) + H(Y|X). \end{aligned}$$

□

Corolario 2.2.7.1. En las condiciones del Teorema 2.2.7,

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

Lema 2.2.8. Sean X, Y dos variables aleatorias discretas con alfabeto $\mathcal{X}, \mathcal{Y}$ respectivamente. Entonces la entropía condicional cumple:

$$0 \leq H(Y|X) \leq H(Y).$$

Donde la desigualdad de la derecha es una igualdad si, y solo si, X e Y son independientes.

Demostración. La desigualdad izquierda se obtiene de la definición de entropía condicional,

$$H(Y|X) = -\mathbb{E}[\underbrace{\log \overbrace{p(Y|X)}^{\in[0,1]}}_{\leq 0}] \geq 0.$$

Aplicando el Teorema 2.2.7 y el Lema 2.2.6 se obtiene la desigualdad de la derecha:

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

dándose la igualdad solo en el caso en que las variables X e Y sean independientes.

□

Ejemplo 2.2.9. Sea (X, Y) una variable aleatoria con distribución conjunta:

Y \ X	1	2	3	4	
1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{4}$
4	$\frac{1}{4}$	0	0	0	$\frac{1}{4}$
	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	

Calculamos las entropías de las variables marginales:

$$H(X) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - 2\frac{1}{8}\log\frac{1}{8} = -\frac{1}{2} - \frac{1}{2} - \frac{6}{8} = \frac{7}{4} \text{ bits.}$$

$$H(Y) = -4\frac{1}{4}\log\frac{1}{4} = 2 \text{ bits.}$$

Calculamos la entropía conjunta:

$$\begin{aligned} H(X,Y) &= -\frac{1}{4}\log\frac{1}{4} - \frac{2}{8}\log\frac{1}{8} - \frac{6}{16}\log\frac{6}{16} - \frac{4}{32}\log\frac{1}{32} \\ &= \frac{1}{2} + \frac{3}{4} + \frac{6}{4} + \frac{5}{8} = \frac{27}{8} \text{ bits.} \end{aligned}$$

Vemos que, efectivamente, la entropía conjunta es mayor que cada una de las individuales, pero menor que su suma $\left(\frac{30}{8}\right)$. Calculamos las entropías condicionales:

$$\begin{aligned} H(Y|X) =& -\frac{1}{8}\log\frac{1}{4} - \frac{2}{16}\log\frac{1}{8} - \frac{1}{4}\log\frac{1}{2} - \frac{2}{16}\log\frac{1}{4} - \frac{1}{8}\log\frac{1}{2} \\ & - \frac{2}{32}\log\frac{1}{4} - \frac{1}{16}\log\frac{1}{2} - \frac{2}{32}\log\frac{1}{4} - \frac{1}{16}\log\frac{1}{2} \\ =& \frac{1}{2} + \frac{6}{8} + \frac{3}{8} = \frac{13}{8} \text{bits.} \end{aligned}$$

$$\begin{aligned} H(X|Y) =& -\frac{1}{8}\log\frac{1}{2} - \frac{2}{16}\log\frac{1}{4} - \frac{2}{16}\log\frac{1}{4} - \frac{1}{8}\log\frac{1}{2} \\ & - \frac{2}{32}\log\frac{1}{8} - \frac{1}{16}\log\frac{1}{4} - \frac{2}{32}\log\frac{1}{8} - \frac{1}{16}\log\frac{1}{4} \\ =& \frac{1}{4} + \frac{6}{8} + \frac{3}{8} = \frac{11}{8} \text{ bits.} \end{aligned}$$

Observamos que las entropías condicionales son positivas y menores que las entropías individuales, además $H(Y|X) \neq H(X|Y)$. Comprobamos que se verifica el Teorema 2.2.7:

$$H(X,Y) = H(X) + H(Y|X) = \frac{7}{4} + \frac{13}{8} = \frac{27}{8} \text{ bits.}$$

$$H(X,Y) = H(Y) + H(X|Y) = 2 + \frac{11}{8} = \frac{27}{8} \text{ bits.}$$

Pasamos a definir otra medida de información, la entropía relativa, esta será una medida de la distancia entre dos distribuciones. La podemos ver como una medida de ineficiencia al asumir que la distribución es q cuando en realidad es p. Será una medida de ineficiencia en el sentido de cuánta información perderíamos si utilizáramos la distribución q para aproximar a la distribución p.

Definición 2.2.4 (Entropía relativa)**.** La entropía relativa (o "distancia" Kullback Leibler) entre dos funciones masa de probabilidad $p(x)$ y $q(x)$ se define como

$$D(p||q) = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} = \mathbb{E}_p \left[\log \frac{p(X)}{q(X)} \right].$$

Las posibles indeterminaciones en la fórmula anterior se definen como: $0 \log \frac{0}{0} = 0$ y $p \log \frac{p}{0} = \infty$.

Lema 2.2.10. Para dos funciones masa de probabilidad $p(x)$ y $q(x)$ se tiene

$$D(p||q) \geq 0,$$

con igualdad si, y solo si, $p(x) = q(x) \quad \forall x \in \mathcal{X}$.

Demostración. Usando la definición de entropía relativa, se obtiene el resultado aplicando el Lema 2.2.5.

□

La entropía relativa no es realmente una distancia, ya que no es simétrica ni verifica la desigualdad triangular. Sin embargo, a veces es útil pensar en ella como una medida de distancia entre distribuciones.

Ejemplo 2.2.11. Sea $\mathcal{X} = \{0,1\}$ y consideramos dos distribuciones p y q en $\mathcal{X}$. Tomamos $p(0) = 1-r$, $p(1) = r$ y $q(0) = 1-s$, $q(1) = s$, con $r, s \in [0,1]$.

$$D(p\|q) = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} = (1-r) \log \frac{1-r}{1-s} + r \log \frac{r}{s}.$$

$$D(q\|p) = \sum_{x\in\mathcal{X}} q(x) \log \frac{q(x)}{p(x)} = (1-s) \log \frac{1-s}{1-r} + s \log \frac{s}{r}.$$

Si tuviéramos $r = s$, entonces $D(p\|q) = 0 = D(q\|p)$. Tomando $r = \frac{1}{2}$ y $s = \frac{1}{4}$ comprobaremos que no se da la simetría de la entropía relativa.

$$\begin{aligned} D(p\|q) &= \frac{1}{2}\log\frac{1/2}{3/4} + \frac{1}{2}\log\frac{1/2}{1/4} = 1 - \frac{1}{2}\log 6 + \frac{1}{2} = 1 - \frac{1}{2}\log 3 \\ &= 0{,}2075 \text{ bits.} \\ D(q\|p) &= \frac{3}{4}\log\frac{3/4}{1/2} + \frac{1}{4}\log\frac{1/4}{1/2} = \frac{3}{4}\log 6 - \frac{3}{2} - \frac{1}{4} = -1 + \frac{3}{4}\log 3 \\ &= 0{,}1887 \text{ bits.} \end{aligned}$$

Luego, $D(p\|q) \neq D(q\|p)$.

Lema 2.2.12. Si X es una variable aleatoria con alfabeto $\mathcal{X}$, entonces

$$H(X) \leq \log \|\mathcal{X}\|,$$

donde $\|\mathcal{X}\|$ representa al cardinal de $\mathcal{X}$.

Demostración. Sea $u(x) = \frac{1}{\|\mathcal{X}\|}$ la función masa de probabilidad uniforme sobre $\mathcal{X}$, y sea $p(x)$ la función masa de probabilidad de X. Entonces,

$$\begin{aligned} D\left(p\|u\right) &= \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{u(x)} = \sum_{x\in\mathcal{X}} p(x)\log p(x) - \sum_{x\in\mathcal{X}} p(x)\log\frac{1}{\|\mathcal{X}\|} \\ &= -H(X) + \log\|\mathcal{X}\|\,. \end{aligned}$$

Usando el Lema 2.2.10, obtenemos el resultado:

$$0 \leq D\left(p\|u\right) = -H(X) + \log\|\mathcal{X}\|\,.$$

□

La información mutua es una medida de la dependencia entre dos variables aleatorias. Mide la cantidad de información que tenemos sobre una de ellas si hemos observado la otra. Se define como sigue.

Definición 2.2.5 (Información mutua)**.** Dadas dos variables aleatorias X e Y discretas con función masa de probabilidad conjunta $p(x, y)$ y funciones masa de probabilidad marginales $p(x)$ y $p(y)$. La información mutua, representada por $I(X;Y)$, es la entropía relativa entre la distribución conjunta y la distribución producto $p(x)p(y)$.

$$\begin{aligned} I(X;Y) &= D\left(p(x,y)\|p(x)p(y)\right) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \\ &= \mathbb{E}_{p(x,y)}\left[\log\frac{p(X,Y)}{p(X)p(Y)}\right]. \end{aligned}$$

Notamos que la información mutua, al estar definida como una entropía relativa, será no negativa (Lema 2.2.10).

Teorema 2.2.13 (Relación entre entropía e información mutua)**.** La siguiente lista recoge algunas relaciones entre la entropía y la información mutua.

a) $I(X;Y) = H(X) - H(X|Y)$.

b) $I(X;Y) = H(Y) - H(Y|X)$.

c) $I(X;Y) = H(X) + H(Y) - H(X,Y)$.

d) $I(X;Y) = I(Y;X)$.

e) $I(X;X) = H(X)$, la entropía es un caso particular de la información mutua.

Demostración.

a)
$$\begin{aligned} I(X;Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(y)p(x|y)\log\frac{p(y)p(x|y)}{p(x)p(y)} \\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x|y) - \sum_{x\in\mathcal{X}} \underbrace{\left(\sum_{y\in\mathcal{Y}} p(y)p(x|y)\right)}_{p(x)} \log p(x) \\ &= -H(X|Y) + H(X). \end{aligned}$$

b) Se obtiene de forma análoga.

c) Se prueba aplicando la regla de la cadena sobre $H(X|Y)$,

$$I(X;Y) = H(X) - H(X|Y) = H(X) + H(Y) - H(X,Y).$$

d) Se prueba usando el apartado c).

e) $I(X;X) = H(X) - H(X|X) = H(X)$.

□

Ejemplo 2.2.14. Para las distribuciones consideradas en el Ejemplo 2.2.9, calculamos la información mutua:

$$\begin{aligned} I(X;Y) = &\frac{1}{8}\log\frac{1/8}{1/2\cdot 1/4} + \frac{1}{16}\log\frac{1/16}{1/4\cdot 1/4} + \frac{2}{32}\log\frac{1/32}{1/8\cdot 1/4} \\ &+\frac{1}{16}\log\frac{1/16}{1/2\cdot 1/4} + \frac{1}{8}\log\frac{1/8}{1/4\cdot 1/4} + \frac{2}{32}\log\frac{1/32}{1/8\cdot 1/4} \\ &+\frac{1}{16}\log\frac{1/16}{1/2\cdot 1/4} + \frac{1}{16}\log\frac{1/16}{1/4\cdot 1/4} + \frac{2}{16}\log\frac{1/16}{1/8\cdot 1/4} \\ &+\frac{1}{4}\log\frac{1/4}{1/2\cdot 1/4} = \frac{2}{16}\log\frac{1}{2} + \frac{1}{2}\log 2 = \frac{3}{8}\text{ bits.} \end{aligned}$$

Vemos que se cumplen las propiedades del Teorema 2.2.13:

$$\begin{aligned} H(X) - H(X|Y) &= \frac{7}{4} - \frac{11}{8} = \frac{3}{8}\text{ bits} = I(X;Y). \\ H(Y) - H(Y|X) &= 2 - \frac{13}{8} = \frac{3}{8}\text{ bits} = I(X;Y). \\ H(X) + H(Y) - H(X,Y) &= \frac{7}{4} + 2 - \frac{27}{8} = \frac{3}{8}\text{ bits} = I(X;Y). \end{aligned}$$

Teorema 2.2.15. Sean X, Y variables aleatorias discretas con alfabeto $\mathcal{X}, \mathcal{Y}$ respectivamente. Entonces se verifica que $I(X;Y) \geq 0$, dándose la igualdad si, y solo si, X, Y son independientes.

Demostración. El apartado c) del Teorema 2.2.13 nos dice que $I(X;Y) = H(X)+H(Y)-H(X,Y)$, usando el Lema 2.2.6.2 tendremos que $I(X;Y) \geq 0$, dándose la igualdad si, y solo si, X, Y son independientes. □

2.2.2 *Caso continuo*

Hasta ahora las definiciones dadas eran para variables aleatorias discretas, que solo tomaban un conjunto finito de valores. Podemos extender las definiciones estudiadas al caso continuo, para aplicarlas en variables aleatorias continuas. Trabajaremos en este apartado con una variable aleatoria X absolutamente continua con función de distribución F, función de densidad f y conjunto soporte $sop(X)$.

Las definiciones a continuación incluyen integrales, por lo que entenderemos que todas ellas existen para no alargar la redacción.

La entropía en el caso continuo pasa a llamarse entropía diferencial y podemos definirla como sigue.

Definición 2.2.6 (Entropía diferencial)**.** La entropía diferencial $h(X)$ de una variable aleatoria continua X viene dada por

$$h(X) = -\int_{sop(X)} \log\left(f(x)\right) f(x) dx.$$

Como en el caso discreto, la entropía diferencial solo depende de la probabilidad de la variable aleatoria, no de los valores que toma. Notamos que en la definición anterior la variable aleatoria X podría ser, o bien, unidimensional, o bien, multidimensional.

Ejemplo 2.2.16. Consideramos una variable aleatoria X absolutamente continua con una distribución uniforme sobre $[0, a]$. En este intervalo su densidad es $f(x) = \frac{1}{a}$ para $x \in [0, a]$ y para el resto de puntos es nula. La entropía es

$$h(X) = -\int_0^a \log\left(\frac{1}{a}\right) \frac{1}{a} dx = \log a.$$

Notamos que si $a < 1, \log a < 0$, esto es, la entropía en el caso continuo puede ser negativa, no como en el caso discreto.

Ejemplo 2.2.17. Consideramos una variable aleatoria X con distribución normal univariante de media μ y varianza σ^2, su función de densidad es

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

queremos calcular su entropía.

$$\begin{aligned}
h(X) &= -\int_{\mathbb{R}} \log\left(f(x)\right) f(x) dx \\
&= -\int_{\mathbb{R}} \log\left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \\
&= -\frac{1}{\sigma\sqrt{2\pi}} \int_{\mathbb{R}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left[\log\left(e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) - \log\left(\sigma\sqrt{2\pi}\right)\right] dx \\
&= -\frac{1}{\sigma\sqrt{2\pi}} \left[\int_{\mathbb{R}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx - \int_{\mathbb{R}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \log\left(\sigma\sqrt{2\pi}\right) dx\right].
\end{aligned}$$

Ahora, utilizando la ley del estadístico inconsciente (que recordamos en la Proposición 2.1.4), tenemos,

$$h(X) = \frac{\mathbb{E}\left[(X-\mu)^2\right]}{2\sigma^2} + \log\left(\sigma\sqrt{2\pi}\right) \underbrace{\int_{\mathbb{R}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx}_{1}$$

$$= \frac{\sigma^2}{2\sigma^2} + \log\sqrt{\sigma^2 2\pi} = \frac{1}{2} + \frac{1}{2}\log\left(\sigma^2 2\pi\right)$$
$$= \frac{1}{2}\log e + \frac{1}{2}\log\left(\sigma^2 2\pi\right)$$
$$= \frac{1}{2}\log\left(2\pi e\sigma^2\right) \text{ nats.}$$

Ejemplo 2.2.18. Tomamos una variable aleatoria multidimensional $X = (X_1, \dots, X_n)$ con distribución normal multivariante de vector de medias μ y matriz de covarianzas Σ. Su función de densidad es

$$f(x) = \frac{1}{\left(\sqrt{2\pi}\right)^n |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}.$$

Comenzamos los cálculos de la entropía,

$$\begin{aligned} h(X) &= -\int_{\mathbb{R}^n} \log\left(f(x)\right) f(x)dx \\ &= -\int_{\mathbb{R}^n} \log\left(\frac{1}{\left(\sqrt{2\pi}\right)^n |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}\right) f(x)dx \\ &= -\int_{\mathbb{R}^n} \left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) - \log\left[\left(\sqrt{2\pi}\right)^n |\Sigma|^{\frac{1}{2}}\right]\right) f(x)dx \\ &= \frac{1}{2}\mathbb{E}\left[(X-\mu)^T\Sigma^{-1}(X-\mu)\right] + \frac{1}{2}\log\left[(2\pi)^n|\Sigma|\right]. \end{aligned}$$

Continuamos desarrollando la expresión $\mathbb{E}\left[(X-\mu)^T\Sigma^{-1}(X-\mu)\right]$,

$$\begin{aligned} \mathbb{E}\left[(X-\mu)^T\Sigma^{-1}(X-\mu)\right] &= \mathbb{E}\left[\sum_{i,j}(X_i-\mu)(\Sigma^{-1})_{ij}(X_j-\mu)\right] \\ &= \mathbb{E}\left[\sum_{i,j}(X_j-\mu)(X_i-\mu)(\Sigma^{-1})_{ij}\right] \\ &= \sum_{i,j}\underbrace{\mathbb{E}\left[\sum_{i,j}(X_j-\mu)(X_i-\mu)\right]}_{Cov(X_j,X_i)}(\Sigma^{-1})_{ij} \\ &= \sum_j\sum_i(\Sigma)_{ji}(\Sigma^{-1})_{ij} = \sum_j(\Sigma\Sigma^{-1})_{jj} \\ &= \sum_j(I_n)_{jj} = n. \end{aligned}$$

Sustituyendo este valor en la expresión de la entropía, obtenemos,

$$\begin{aligned} h(X) &= \frac{n}{2} + \frac{1}{2}\log\left[(2\pi)^n|\Sigma|\right] = \frac{1}{2}\left(\log e^n + \log\left[(2\pi)^n|\Sigma|\right]\right) \\ &= \frac{1}{2}\log\left[(2\pi e)^n|\Sigma|\right] \text{ nats.} \end{aligned}$$

Teorema 2.2.19. Sean $X_1, \dots, X_n$ una serie de variables aleatorias independientes e idénticamente distribuidas con función de densidad $f_X(x)$. Si notamos por $f(x_1, \dots, x_n)$ a la función de densidad conjunta, entonces

$$-\frac{1}{n}\log f(X_1, \dots, X_n) \xrightarrow{P} \mathbb{E}\left[-\log f_X(X)\right] = h(X).$$

Demostración. Como las variables son independientes, se verifica

$$-\frac{1}{n}\log f(X_1, \dots, X_n) = -\frac{1}{n}\sum_{i=1}^{n}\log f_X(X_i),$$

por el Teorema 2.1.10 tenemos que la expresión anterior converge en probabilidad a $\mathbb{E}\left[-\log f_X(X)\right] = h(X)$.

□

Definición 2.2.7 (Entropía diferencial condicionada)**.** Si X, Y son variables aleatorias con función de densidad conjunta $f(x, y)$, definimos la entropía diferencial condicionada $h(X|Y)$ como

$$h(X|Y) = -\int\int \log\left(f(x|y)\right) f(x, y)dxdy.$$

Como $f(x|y) = \frac{f(x,y)}{f(y)}$, también podemos definirla como

$$h(X|Y) = h(X, Y) - h(Y),$$

siempre que las entropías diferenciales sean finitas.

Definición 2.2.8 (Entropía relativa)**.** La entropía relativa (o "distancia" de Kullback-Leibler) $D(f||g)$ entre dos densidades f y g se define como

$$D(f||g) = \int_{\mathbb{R}^n} \log\left(\frac{f(x)}{g(x)}\right) f(x)dx.$$

Notamos que $D(f||g)$ es finita si, y solo si, el conjunto soporte de f está contenido en el conjunto soporte de g. Al igual que en el caso discreto y motivados por la continuidad consideraremos $0\log\frac{0}{0} = 0$.

Definición 2.2.9 (Información mutua)**.** La información mutua, $I(X;Y)$, entre dos variables aleatorias con función de densidad conjunta $f(x,y)$ y marginales $f(x)$ y $f(y)$ respectivamente, viene dada por

$$I(X;Y) = \int\int \log\left(\frac{f(x,y)}{f(x)f(y)}\right) f(x,y)dxdy = D(f(x,y)||f(x)f(y)).$$

Lema 2.2.20. Siguiendo la notación anterior, la información mutua de dos variables aleatorias X, Y verifica:

$$I(X;Y) = h(X) + h(Y) - h(X,Y).$$

Demostración. Supongamos que las funciones de densidad marginales de X e Y son $f(x)$, $f(y)$, respectivamente, y la conjunta es $f(x,y)$.

$$\begin{aligned}
I(X;Y) =& \int\int \log\left(\frac{f(x,y)}{f(x)f(y)}\right) f(x,y)dxdy \\
=& \int\int \log\left(f(x,y)\right) f(x,y)dxdy \\
&- \int\int \log\left(f(x)f(y)\right) f(x,y)dxdy \\
=& - h(X,Y) - \int\int \log\left(f(x)\right) f(x,y)dydx \\
&- \int\int \log\left(f(y)\right) f(x,y)dxdy \\
=& - \int \log f(x)f(x)dx - \int \log f(y)f(y)dy - h(X,Y) \\
=& h(X) + h(Y) - h(X,Y).
\end{aligned}$$

En la cadena de igualdades anterior hemos utilizado las propiedades de los logaritmos y el Teorema de Fubini.

□

Lema 2.2.21. La información mutua de dos variables aleatorias X, Y se relaciona con la entropía condicional de la siguiente forma:

$$I(X;Y) = h(X) - h(X|Y) = h(Y) - h(Y|X).$$

Demostración. Sean $f(x)$, $f(y)$ las funciones de densidad de X e Y respectivamente y $f(x,y)$ la función de densidad conjunta. Como en el lema anterior, partiremos de la definición de información mutua y utilizaremos propiedades del logaritmo, el Teorema de Fubini y la definición de función de densidad condicionada.

$$\begin{aligned} I(X;Y) &= \int\int \log\left(\frac{f(x,y)}{f(x)f(y)}\right) f(x,y)dxdy \\ &= -\int\int \log\left(f(x)\right) f(x,y)dxdy \\ &\quad + \int\int \log\left(\frac{f(x,y)}{f(y)}\right) f(x,y)dxdy \\ &= -\int \log\left(f(x)\right) f(x)dx + \int\int \log\left(f(x|y)\right) f(x,y)dxdy \\ &= h(X) - h(X|Y). \end{aligned}$$

De manera análoga se obtiene la segunda igualdad.

□

Definición 2.2.10 (Variable aleatoria cuantificada)**.** Sea X una variable aleatoria con función de densidad f. Si dividimos la imagen de X en intervalos de longitud ε y asumimos que la densidad es continua en cada intervalo, el Teorema del valor medio nos dice que, para cada intervalo $[i\varepsilon, (i+1)\varepsilon]$ existe un valor $x_i \in [i\varepsilon, (i+1)\varepsilon]$ tal que

$$f(x_i)\varepsilon = \int_{i\varepsilon}^{(i+1)\varepsilon} f(x)dx.$$

La variable aleatoria cuantificada, X^ε, se define de la siguiente forma

$$X^\varepsilon = x_i, \quad \text{si } i\varepsilon \leq X < (i+1)\varepsilon.$$

Su función masa de probabilidad viene dada por

$$P[X^\varepsilon = x_i] = p_i = \int_{i\varepsilon}^{(i+1)\varepsilon} f(x)dx = f(x_i)\varepsilon.$$

Teorema 2.2.22. Si la densidad f de una variable aleatoria X es integrable Riemann, entonces

$$H(X^\varepsilon) + \log\varepsilon \xrightarrow{\varepsilon\to 0} h(X).$$

Demostración. Partimos de la entropía de X^ε,

$$\begin{aligned} H(X^\varepsilon) &= -\sum_{-\infty}^{+\infty} p_i \log p_i = -\sum_{-\infty}^{+\infty} f(x_i)\varepsilon \log\left(f(x_i)\varepsilon\right) \\ &= -\sum_{-\infty}^{+\infty} f(x_i)\varepsilon \log\left(f(x_i)\right) - \underbrace{\sum_{-\infty}^{+\infty} f(x_i)\varepsilon}_{\int_{-\infty}^{+\infty} f(x)dx = 1} \log\left(\varepsilon\right), \end{aligned}$$

luego, tenemos que $H(X^\varepsilon) + \log \varepsilon = \sum_{-\infty}^{+\infty} f(x_i)\varepsilon \log\left(f(x_i)\right)$.

Como f es integrable Riemann, tomando $\varepsilon \to 0$ obtenemos

$$\lim_{\varepsilon \to 0} \sum_{-\infty}^{+\infty} f(x_i)\varepsilon \log\left(f(x_i)\right) = -\int_{-\infty}^{+\infty} \log\left(f(x)\right) f(x)dx = h(X).$$

Por tanto,

$$H(X^\varepsilon) + \log \varepsilon \xrightarrow{\varepsilon \to 0} h(X).$$

□

Teniendo en cuenta el resultado anterior podemos aproximar la información mutua de dos variables aleatorias por la de sus variables cuantificadas.

$$\begin{aligned} I(X^\varepsilon; Y^\varepsilon) &= H(X^\varepsilon) - H(X^\varepsilon|Y^\varepsilon) \\ &\approx h(X) - \log \varepsilon - (h(X|Y) - \log \varepsilon) = I(X;Y). \end{aligned} \tag{1}$$

De forma más general, podemos definir la información mutua en términos de particiones finitas de la imagen de la variable aleatoria. Sea $\mathcal{X}$ la imagen de la variable aleatoria X. Una partición $\mathcal{P}$ de $\mathcal{X}$ es una colección finita de conjuntos disjuntos P_i de forma que $\cup_i P_i = \mathcal{X}$. La cuantificación de X dada la partición $\mathcal{P}$, que notaremos $[X]_\mathcal{P}$, es la variable aleatoria discreta con función masa de probabilidad

$$P\left[[X]_\mathcal{P} = i\right] = P[X \in P_i] = \int_{P_i} dF(x).$$

Utilizando esto para dos variables aleatorias con sendas particiones, podremos calcular la información mutua de las variables cuantificadas utilizando (1). De esta forma, obtenemos una nueva definición de la información mutua.

Definición 2.2.11 (Información mutua)**.** La información mutua de dos variables aleatorias X e Y viene dada por

$$I(X;Y) = \sup_{\mathcal{P},\mathcal{Q}} I\left([X]_\mathcal{P}, [Y]_\mathcal{Q}\right),$$

donde el supremo es sobre todas las particiones finitas $\mathcal{P}$ y $\mathcal{Q}$.

Notamos que esta definición también será válida cuando las variables aleatorias no tengan definida su función de densidad. Esta definición es equivalente a la Definición 2.2.5 para variables aleatorias discretas y a la Definición 2.2.9 para variables aleatorias continuas.

Teorema 2.2.23. Si f y g son dos funciones de densidad, entonces

$$D\left(f||g\right) \geq 0$$

con igualdad si, y solo si, $f = g$ en casi todo punto.

Demostración. Sea $sop(f)$ el soporte de f, entonces

$$\begin{aligned} -D\left(f||g\right) &= \int_{sop(f)} \log\left(\frac{g(x)}{f(x)}\right) f(x)dx \leq \log\left(\int_{sop(f)} f(x) \cdot \frac{g(x)}{f(x)}dx\right) \\ &= \log\left(\int_{sop(f)} g(x)dx\right) \leq 1 = 0. \end{aligned}$$

La primera desigualdad es la desigualdad de Jensen, Teorema 2.1.5, aplicada a log, una función cóncava que no es afín. Por ello, será igualdad si, y solo si, $\log \frac{g}{f} = 0$ para casi todo punto, esto es, $f = g$ en casi todo punto, la misma condición para que la segunda desigualdad sea igualdad.

□

Corolario 2.2.23.1. $I(X;Y) \geq 0$ con igualdad si, y solo si, X e Y son independientes.

Corolario 2.2.23.2. $h(X|Y) \leq 0$ con igualdad si, y solo si, X e Y son independientes.

Teorema 2.2.24 (Regla de la cadena para la entropía diferencial)**.** Sean $X_1, \ldots, X_n$ variables aleatorias, entonces

$$h(X_1, \ldots, X_n) = \sum_{i=1}^{n} h(X_i|X_1, \ldots, X_{i-1}).$$

Demostración. Realizaremos la prueba por inducción sobre el número de variables.

- Si $n = 2$, entonces $h(X_1, X_2) = h(X_1) + h(X_2|X_1)$ por definición de la entropía diferencial condicionada.
- Suponiendo cierto el caso $n - 1$, veamos que también se cumple para n, utilizando otra vez la definición de entropía diferencial condicionada:

$$\begin{aligned} h(X_1, \ldots, X_n) &= h(X_n|X_1, \ldots, X_{n-1}) + h(X_1, \ldots, X_{n-1}) \\ &= h(X_n|X_1, \ldots, X_{n-1}) + \sum_{i=1}^{n-1} h(X_i|X_1, \ldots, X_{i-1}). \end{aligned}$$

□

Corolario 2.2.24.1. Se tiene que

$$h(X_1, \dots, X_n) \leq \sum_i h(X_i),$$

dándose la igualdad si, y solo si, $X_1, \dots, X_n$ son independientes.

Teorema 2.2.25. Sea X una variable aleatoria absolutamente continua y c un vector de escalares, entonces

$$h(X + c) = h(X).$$

La entropía diferencial no cambia por traslaciones.

Demostración. Sea f_X la función de densidad de X. Llamemos $Y = X + c$, esta será una variable aleatoria cuya función de densidad se relaciona con la de X como sigue: $f_Y(y) = f_X(y-c)$. Calculamos su entropía diferencial usando su definición y aplicando el cambio de variable $y = x + c$.

$$\begin{aligned} h(X + c) = h(Y) &= -\int \log\left(f_Y(y)\right) f_Y(y) dy \\ &= -\int \log\left(f_X(y - c)\right) f_X(y - c) dy \\ &= -\int \log\left(f_X(x)\right) f_X(x) dx = h(X). \end{aligned}$$

□

Teorema 2.2.26. Sea X una variable aleatoria unidimensional y a un escalar no nulo, entonces

$$h(aX) = h(X) + \log|a|.$$

Demostración. Llamemos $Y = aX$, entonces $f_Y(y) = \frac{1}{|a|} f_X\left(\frac{y}{a}\right)$. Usaremos la definición de entropía, la función de densidad recién calculada y un cambio de variables ($x = \frac{y}{a}$):

$$\begin{aligned} h(aX) = h(Y) &= -\int \log\left(f_Y(y)\right) f_Y(y) dy \\ &= -\int \log\left(\frac{1}{|a|} f_X\left(\frac{y}{a}\right)\right) \frac{1}{|a|} f_X\left(\frac{y}{a}\right) dy \\ &= -\int \log\left(\frac{1}{|a|} f_X\left(x\right)\right) \frac{|a|}{|a|} f_X\left(x\right) dx \\ &= -\int \log\left(f_X\left(x\right)\right) f_X\left(x\right) dx + \int \log\left(|a|\right) f_X(x) dx \\ &= h(X) + \log|a|. \end{aligned}$$

□

Corolario 2.2.26.1. Si X es una variable aleatoria multidimensional y A una matriz entonces,

$$h(AX) = h(X) + \log|det(A)|.$$

2.3 ESTIMACIÓN DE LA ENTROPÍA Y DE LA INFORMACIÓN MUTUA

Generalmente no conocemos las funciones de densidad de las variables aleatorias cuya entropía o información mutua queremos calcular, por ello, tratamos de aproximar este valor a partir de información que sí conocemos, como realizaciones muestrales de las variables. El objetivo es, para dos variables aleatorias X, Y (d-dimensionales) y la variable aleatoria conjunta $Z = (X, Y)$, estimar $h(X)$ o $I(X;Y)$ a partir de un conjunto de realizaciones muestrales, $\{z_i = (x_i, y_i) : i = 1, \dots, N\}$, sin conocer las funciones de densidad f_z, f_x y f_y. Asumiremos en el desarrollo a continuación que todas las funciones de densidad son propias, es decir, que no son idénticamente $-\infty$.

Presentaremos brevemente un estimador basado en particiones antes de estudiar estimadores basados en los k vecinos más cercanos y del tipo de Kozachenko - Leonenko.

2.3.1 *Estimador basado en particiones*

En este enfoque, dividiremos el soporte de X e Y en particiones de tamaño finito (no necesariamente el mismo) y aproximamos la información mutua por

$$I(X,Y) \approx I_{binned}(X,Y) = \sum_{i,j} p(i,j) \log \frac{p(i,j)}{p_x(i)p_y(j)},$$

donde $p_x(i) = \int_i f_x(x)dx$, $p_y(j) = \int_j f_y(y)dy$, $\int_i$ representa la integral sobre la i-ésima partición, y $p(i,j) = \int_i \int_j f(x,y)dxdy$.

Llamamos $n_x(i)$ al número de puntos en la i-ésima partición de X, análogamente $n_y(j)$. $n(i,j)$ es el número de puntos en la intersección de $n_x(i)$ y $n_y(j)$.

Aproximamos

$$\begin{aligned} p_x(i) &\approx \frac{n_x(i)}{N}, \\ p_y(j) &\approx \frac{n_y(j)}{N}, \\ p(i,j) &\approx \frac{n(i,j)}{N}. \end{aligned}$$

Cuando N tienda a infinito y el tamaño de cada partición converja a cero, la estimación $I_{binned}(X,Y) \to I(X;Y)$, véase el Teorema 1 de [26].

2.3.2 *Estimadores de la entropía basados en los k vecinos más cercanos*

Introducimos a continuación estimadores de la entropía, $h(X)$, basados en los k vecinos más cercanos. Para ello, consideramos X una variable aleatoria d-dimensional, con función de densidad $f(x)$. Queremos estimar la entropía

$$h(X) = -\int \log\left(f(x)\right) f(x) dx$$

a partir de una muestra aleatoria simple $X_1, \dots, X_N$ de X.

Teniendo en cuenta el Teorema 2.2.19, si tenemos un estimador consistente de $\log f(X_i)$, que notaremos por $\widehat{\log f}(X_i)$, podemos estimar $h(X)$ mediante

$$-\frac{1}{N}\sum_{i=1}^{N} \widehat{\log f}(X_i).$$

Para obtener el estimador $\widehat{\log f}(X)$, partimos de los estimadores de la función de densidad $\hat{f}(X)$ basados en los k vecinos más cercanos.

Sea $X_1, \dots, X_N$ una muestra aleatoria simple y $x_1, \dots, x_N$ una observación de dicha muestra. Consideramos en $\mathbb{R}^d$, la métrica inducida por una norma $\|\cdot\|$ (euclídea, del máximo,...). Fijado $x_i \in \mathbb{R}^d$, ordenamos el resto de observaciones de la muestra en orden creciente de la distancia a este punto y llamamos $\varepsilon_k(x_i)$ a la distancia de x_i a su k vecino más próximo, que notamos por x_{i_k}.

Consideramos la bola de centro x_i y radio $\varepsilon_k(x_i)$,

$$B(x_i, \varepsilon_k(x_i)) = \{y \in \mathbb{R}^d : \|x_i - y\| \leq \varepsilon_k(x_i)\}.$$

Comenzaremos aproximando, para cualquier $j = 1, \dots, N$,

$$P[X_j \in B(x_i, \varepsilon_k(x_i))] \approx \int_{B(x_i, \varepsilon_k(x_i))} f(y) dy.$$

Podemos ver

$$\frac{1}{N}\sum_{j=1}^{N} I\left(X_j \in B(x_i, \varepsilon_k(x_i))\right),$$

donde I es la función indicadora, como un estimador de la distribución $P[X_j \in B(x_i, \varepsilon_k(x_i))]$. Por la definición de $\varepsilon_k(x_i)$ tenemos que

$$\frac{1}{N}\sum_{j=1}^{N} I\left(X_j \in B\left(x_i, \varepsilon_k(x_i)\right)\right) = \frac{k}{N}. \tag{2}$$

Cuando N es grande y k es relativamente pequeño en comparación, $\varepsilon_k(x_i)$ será pequeño porque $\frac{k}{N}$ lo es. Luego, la densidad $f(y)$ en la bola $B(x_i, \varepsilon_k(x_i))$ no cambiará demasiado. Para cualquier punto de la bola consideraremos que su densidad coincide con la del centro de la misma, $f(y) \approx f(x_i)$. Tenemos entonces

$$P[X_j \in B(x_i, \varepsilon_k(x_i))] \approx \int_{B(x_i, \varepsilon_k(x_i))} f(y)dy \approx f(x_i) \int_{B(x_i, \varepsilon_k(x_i))} dy$$
$$= f(x_i) \cdot V_d \cdot \varepsilon_k^d(x_i),$$

donde $V_d \cdot \varepsilon_k^d(x_i)$ es el volumen de la bola d dimensional de radio $\varepsilon_k(x_i)$ (este volumen depende la norma utilizada).

Utilizando (2) para estimar $P[X_j \in B(x_i, \varepsilon_k(x_i))]$, tendremos,

$$f(x_i) \cdot V_d \cdot \varepsilon_k(x_i) \approx P[X_j \in B(x_i, \varepsilon_k(x_i))] \approx \frac{1}{N} \sum_{j=1}^{N} I\left(X_j \in B\left(x_i, \varepsilon_k(x_i)\right)\right)$$
$$= \frac{k}{N}.$$

Tomando

$$f(x_i) \cdot V_d \cdot \varepsilon_k^d(x_i) \approx \frac{k}{N}$$

se obtiene el estimador de la función de densidad,

$$\hat{f}(x) = \frac{k}{N} \cdot \frac{1}{V_d \cdot \varepsilon_k^d(x)}, \quad \forall x \in \mathbb{R}^d.$$

El comportamiento asintótico de este estimador no depende solo del tamaño de la muestra sino también de la dimensión d y del valor k elegido. De hecho, la varianza depende exclusivamente del valor de k, la variabilidad del estimador viene dada por el número de vecinos más cercanos a considerar. Este estimador es sesgado, aunque es asintóticamente insesgado y $\mathrm{Var}\left(\hat{f}\right) \to 0$ solo si $k \to \infty, N \to \infty$ y $\frac{k}{N} \to 0$, véase el corolario 2 de [28] para la demostración general o [7] para el desarrollo del caso $d = 1$. Aumentar el valor de k incrementa el tiempo de cómputo notablemente, dificultando el cálculo efectivo.

Como habíamos expuesto, utilizaremos este estimador de la función de densidad de X para obtener un estimador de $\log f(X)$, este es,

$$\widehat{\log f}(x_i) = \log\left(\frac{k}{N} \cdot \frac{1}{V_d \cdot \varepsilon_k^d(x_i)}\right) = \log\frac{k}{N} - \log V_d - d \log \varepsilon_k(x_i).$$

Obteniendo el estimador de la entropía

$$\begin{aligned}\hat{h}(x) &= -\frac{1}{N}\sum_{i=1}^{N}\widehat{\log f}(x_i)\\ &= -\log\frac{k}{N} + \log V_d + \frac{d}{N}\sum_{i=1}^{N}\log\varepsilon_k(x_i).\end{aligned}$$

Como proviene del estimador $\hat{f}$, el estimador $\hat{h}$ mantiene sus propiedades asintóticas. Para corregir en parte el sesgo del estimador y para solucionar los problemas de cómputo de este estimador, se propone la modificación siguiente, basada en el estimador de Kozachenko - Leonenko.

2.3.3 *Estimadores de la entropía del tipo de Kozachenko - Leonenko*

Consideramos $X_1, \dots, X_n$ una muestra aleatoria simple de la variable aleatoria X con función de densidad f. Partiendo del mismo razonamiento, trataremos de estimar la entropía, $h(X)$, a partir de

$$\hat{h}(X) = -\frac{1}{N}\sum_{i=1}^{N}\widehat{\log f}(X_i).$$

La diferencia vendrá marcada por la forma de estimar $\widehat{\log f}(X_i)$, para corregir el sesgo del estimador anterior.

Siguiendo con la notación anterior, llamamos $\mathcal{E}_{i_k}$ a la variable aleatoria $\|X_i - X_{i_k}\|$ y $P_k(\varepsilon)$ a su función de densidad.

Para obtener el estimador $\widehat{\log f}(X_i)$, para un k fijo, consideramos la función de densidad, $P_k(\varepsilon)$, de la distancia entre X_i y su k-ésimo vecino más cercano, donde ε será una realización de $\mathcal{E}_{i_k}$.

Partimos de una observación $x_1, \dots, x_n$, de la que seleccionamos un punto x_i cualquiera. Vamos a calcular $P\left[\varepsilon \leq \mathcal{E}_{i_k} \leq \varepsilon + h\right]$. Esta probabilidad es igual a la probabilidad de que haya un punto a distancia $r \in [\varepsilon, \varepsilon + h]$ de x_i, de que haya otras $k-1$ observaciones a distancia menor que ε; y de que el resto de puntos se encuentren a mayor distancia de x_i.

Llamamos

$$p_i(\varepsilon) = \int_{B(x_i,\varepsilon)} f(y)dy.$$

Para cada $j = 1, \dots, n$ aproximaremos

$$P\left[X_j \in B(x_i, \varepsilon)\right] \approx p_i(\varepsilon),$$

donde $B(x_i, \varepsilon)$ es la bola centrada en x_i y de radio ε.

Usaremos la fórmula de la distribución multinomial para expresar $P_k(\varepsilon)$,

$$\begin{aligned} P_k(\varepsilon) &= \lim_{h\to 0} \frac{P\left[\varepsilon \leq \mathcal{E}_{i_k} \leq \varepsilon + h\right]}{h} \\ &= \lim_{h\to 0} \left[\frac{(N-1)!}{1!(k-1)!(N-k-1)!} \cdot p_i(\varepsilon)^{k-1} \cdot (1-p_i(\varepsilon))^{N-k-1} \right. \\ &\qquad \left. \left(\frac{p_i(\varepsilon+h) - p_i(\varepsilon)}{h} \right) \right] \\ &= \quad k \binom{N-1}{k} \cdot \frac{dp_i(\varepsilon)}{d\varepsilon} \cdot p_i(\varepsilon)^{k-1} \cdot (1-p_i(\varepsilon))^{N-k-1}. \end{aligned}$$

Observamos que

$$\int_0^\infty P_k(\varepsilon) d\varepsilon = k\binom{N-1}{k} \int_0^1 p^{k-1}(1-p)^{N-k-1} dp = 1,$$

que debe cumplir para ser función de densidad.

Podemos calcular la esperanza de $\log p_i(\varepsilon)$, a partir de la posición de los $N-1$ puntos restantes, con x_i fijo:

$$\begin{aligned} \mathbb{E}\left[\log p_i(\varepsilon)\right] &= \int_0^\infty P_k(\varepsilon) \log p_i(\varepsilon) d\varepsilon \\ &= k\binom{N-1}{k} \int_0^1 p^{k-1}(1-p)^{N-k-1} \log p \, dp \\ &= \psi(k) - \psi(N), \end{aligned}$$

donde ψ es la función digamma, esta expresión se ha obtenido de [19].

Si asumimos que $f(x)$ es constante en la bola de radio ε, tendremos

$$\begin{aligned} p_i(\varepsilon) &\approx \int_{B(x_i,\varepsilon)} f(y) dy \approx f(x_i) \int_{B(x_i,\varepsilon)} dy \\ &= f(x_i) \cdot V_d \cdot \varepsilon^d. \end{aligned}$$

Tomando logaritmos sobre la aproximación de $p_i(\varepsilon)$, obtenemos:

$$\log p_i(\varepsilon) \approx \log f(x_i) + \log V_d + d \log \varepsilon,$$

despejando $\log f(x_i)$,

$$\log f(x_i) \approx \log p_i(\varepsilon) - \log V_d - d \log \varepsilon.$$

Continuamos tomando esperanzas respecto de ε y sustituyendo la esperanza del logaritmo ya calculada, notamos que $\log f(X_i)$ no depende de ε:

$$\begin{aligned}\log f(x_i) &\approx \mathbb{E}\left[\log p_i(\varepsilon)\right] - \log V_d - d\mathbb{E}\left[\log \varepsilon\right] \\ &= \psi(k) - \psi(N) - \log V_d - d\mathbb{E}\left[\log \varepsilon\right].\end{aligned}$$

Hemos calculado un estimador de $\widehat{\log f}(X_i)$ que nos permite obtener un estimador de la entropía:

$$\begin{aligned}\hat{h}(X) &= -\frac{1}{N}\sum_{i=1}^{N}\widehat{\log f}(X_i) = -\frac{1}{N}\sum_{i=1}^{N}\left(\psi(k) - \psi(N) - \log V_d - d\log\varepsilon\right) \\ &= -\psi(k) + \psi(N) + \log V_d + \frac{d}{N}\sum_{i=1}^{N}\log\varepsilon(x_i). \qquad (3)\end{aligned}$$

El estimador anterior es el de Kozachenko - Leonenko en el caso $k = 1$ [20].

La forma de estimar $\widehat{\log f}(X_i)$ elimina una parte del sesgo, aunque se mantiene el sesgo proveniente de aproximar

$$\int_{B(x_i,\varepsilon)} f(y)dy \approx f(x_i)\int_{B(x_i,\varepsilon)} dy.$$

Si la densidad $f(x)$ es constante, el sesgo proveniente de esta aproximación también desaparece y obtenemos un estimador de la entropía, $\hat{h}(X)$, que es insesgado.

2.3.4 *Estimadores de la información mutua del tipo de Kozachenko - Leonenko*

Extendemos las ideas anteriores para estimar la información mutua.

Consideremos ahora la variable aleatoria conjunta $Z = (X, Y)$ con la norma del máximo

$$\|Z - Z'\| = \text{máx}\{\|X - X'\|, \|Y - Y'\|\},$$

donde las normas en X e Y pueden ser cualesquiera, no necesariamente la misma.

Para un k fijo, tomamos uno de los N puntos de la muestra, z_i, y notamos $\varepsilon_k(z_i)$ la distancia de este punto a su k-ésimo vecino más cercano, esta será una realización de $\mathcal{E}_{i_k}$. $\varepsilon_k(x_i)$ y $\varepsilon_k(y_i)$ reflejan la distancia de ese punto proyectada en los subespacios X e Y respectivamente. Por la norma elegida, se verifica

$$\varepsilon_k(z_i) = \text{máx}\{\varepsilon_k(x_i), \varepsilon_k(y_i)\}.$$

Distinguiremos dos algoritmos diferentes para estimar la información mutua según si consideramos los vecinos que se encuentren en el cuadrado de lado $2\varepsilon(z_i)$, o en el rectángulo de lados $2\varepsilon(x_i)$, $2\varepsilon(y_i)$.

Primer algoritmo

Este algoritmo se caracteriza por definir $n_x(i)$ como el número de puntos x_j cuya distancia a x_i es menor que $\varepsilon_k(z_i)$, análogamente se define $n_y(i)$. $\varepsilon_k(z_i)$ es una observación de una variable aleatoria con densidad dada por $P_k(\varepsilon_k(z_i))$, luego $\mathbb{E}\left[\log p_i\left(\varepsilon_k(z_i)\right)\right] = \psi(k) - \psi(N)$ se cumple también en este caso.

Para utilizar el estimador de la entropía calculado anteriormente debemos cambiar: x_i por $z_i = (x_i, y_i)$, d por $d_X + d_Y$ y V_d por $V_{d_X}V_{d_Y}$. Con estas modificaciones obtenemos

$$\hat{h}(X,Y) = -\psi(k) + \psi(N) + \log(V_{d_X}V_{d_Y}) + \frac{d_X + d_Y}{N}\sum_{i=1}^{N}\log\left(\varepsilon_k(z_i)\right).$$

Para obtener una estimación de $I(X;Y)$ utilizaremos el Lema 2.2.20,

$$\hat{I}(X;Y) = \hat{h}(X) + \hat{h}(Y) - \hat{h}(X,Y).$$

Podríamos usar el mismo valor de k para realizar estas estimaciones, pero entonces estaríamos utilizando diferentes escalas en el espacio conjunto y en los marginales. Para un k fijo, la distancia al k-ésimo vecino en el espacio conjunto sería mayor que las distancias a los vecinos en los espacios marginales. El sesgo en el estimador de la entropía proveniente de la no uniformidad de la densidad depende de estas distancias, los sesgos en $\hat{h}(X)$, $\hat{h}(Y)$, $\hat{h}(X,Y)$ no se cancelarían. Para evitar eso, notamos que la estimación se cumple para cualquier valor de k y que no tenemos por qué mantenerlo fijo.

Suponemos que el k-ésimo vecino de x_i se encuentra en uno de los lados verticales del cuadrado de lado $2\varepsilon_k(z_i)$. Entre las rectas verticales $x_i - \varepsilon_k(z_i)$ y $x_i + \varepsilon_k(z_i)$ hay $n_x(i)$ puntos, por definición de $n_x(i)$, luego, $\varepsilon_k(z_i)$ coincide con la distancia entre x_i y su $(n_x(i)+1)$-ésimo vecino más cercano. Tomando $k = n_x(i) + 1$, estimamos la entropía de la variable X mediante

$$\hat{h}(X) = -\frac{1}{N}\sum_{i=1}^{N}\psi(n_x(i)+1) + \psi(N) + \log V_{d_X} + \frac{d_X}{N}\sum_{i=1}^{N}\log\varepsilon_k(z_i).$$

Para la variable Y, $\varepsilon_k(z_i)$ no tendría por qué coincidir con la distancia entre y_i y su $(n_y(i)+1)$-ésimo vecino. Sin embargo, consideramos la ecuación anterior una buena aproximación para $h(Y)$ (cambiando X por Y). Esta aproximación será exacta cuando $n_y(i) \to \infty$ y cuando $N \to \infty$

[33].

Usando estas estimaciones en la ecuación correspondiente y notando $\varepsilon(i) = \varepsilon_k(z_i)$, obtenemos:

$$\begin{aligned} I^{(1)}(X,Y) &\approx \hat{h}(X) + \hat{h}(Y) - \hat{h}(X,Y) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\psi(n_x(i)+1) + \psi(N) + \log V_{d_X} + \frac{d_X}{N}\sum_{i=1}^{N}\log(\varepsilon(i)) \\ &\quad -\frac{1}{N}\sum_{i=1}^{N}\psi(n_y(i)+1) + \psi(N) + \log V_{d_Y} + \frac{d_Y}{N}\sum_{i=1}^{N}\log(\varepsilon(i)) \\ &\quad +\psi(k) - \psi(N) - \log(V_{d_X}V_{d_Y}) - \frac{d_X+d_Y}{N}\sum_{i=1}^{N}\log(\varepsilon(i)) \\ &= \psi(k) - \frac{1}{N}\sum_{i=1}^{N}\big(\psi(n_x(i)+1) + \psi(n_y(i)+1)\big) + \psi(N). \end{aligned}$$

Notamos $\langle \dots \rangle$ a la media sobre todos los $i \in \{1, \dots, N\}$ y sobre todas las realizaciones de las muestras aleatorias,

$$\langle \dots \rangle = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}[\dots(i)].$$

Utilizando esta notación la estimación de la información mutua quedaría:

$$I^{(1)}(X,Y) \approx \psi(k) - \langle \psi(n_x+1) + \psi(n_y+1) \rangle + \psi(N). \tag{4}$$

El principal inconveniente de esta estimación es que solo usamos el estimador de la entropía correctamente en una dirección marginal. Esto parece inevitable si queremos usar (hiper)cubos en el espacio conjunto, como opción alternativa podríamos usar (hiper)rectángulos.

Segundo algoritmo

En este caso, $n_x(i)$ y $n_y(i)$ serán el número de puntos con $\|x_i - x_j\| \leq \varepsilon_k(x_i)$ y $\|y_i - y_j\| \leq \varepsilon_k(y_i)$ respectivamente. Seguimos notando $\varepsilon(i) = \text{máx}\{\varepsilon_k(X_i), \varepsilon_k(Y_i)\}$. Tenemos que distinguir dos situaciones:

a) Los lados $\varepsilon_k(x_i)$ y $\varepsilon_k(y_i)$ quedan determinados por el mismo punto, lo denotamos suceso A.

b) Los lados $\varepsilon_k(x_i)$ y $\varepsilon_k(y_i)$ quedan determinados por puntos diferentes, lo llamamos suceso B.

En ambos casos tenemos que cambiar $P_k(\varepsilon)$ por una probabilidad 2-dimensional:

$$\begin{aligned} P_k(\varepsilon_x, \varepsilon_y) &= P_k^{a)}(\varepsilon_x, \varepsilon_y) + P_k^{b)}(\varepsilon_x, \varepsilon_y) \\ &= P_k\left(\varepsilon_x, \varepsilon_y | A\right) \cdot P(A) + P_k\left(\varepsilon_x, \varepsilon_y | B\right) \cdot P(B) \\ &= P_k\left(\varepsilon_x, \varepsilon_y | A\right) \cdot \frac{1}{k} + P_k\left(\varepsilon_x, \varepsilon_y | B\right) \cdot \frac{k-1}{k}. \end{aligned}$$

En el artículo [19] se especifican los valores de estas probabilidades, necesarias para llegar a la siguiente expresión sobre $q_i \equiv q_i(\varepsilon_k(X_i), \varepsilon_k(Y_i))$, la función de densidad del rectángulo de lados $2\varepsilon_k(X_i) \times 2\varepsilon_k(Y_i)$ centrado en el punto (X_i, Y_i),

$$\mathbb{E}(\log q_i) = \int \int_0^\infty P_k(\varepsilon_x, \varepsilon_y) \log q_i(\varepsilon_x, \varepsilon_y) d\varepsilon_x d\varepsilon_y = \psi(k) - \frac{1}{k} - \psi(N).$$

Aproximamos $\log f(x_i, y_i)$ por

$$\log f(x_i, y_i) \approx \log q_i(\varepsilon(i)) - \log V_d - d \log \varepsilon(i),$$

tomando esperanzas, obtenemos

$$\log f(x_i, y_i) \approx \psi(k) - \frac{1}{k} - \psi(N) - \log V_d - d\mathbb{E}\left[\log \varepsilon(i)\right].$$

Esto nos da el siguiente estimador de la entropía

$$\begin{aligned} \hat{h}(X, Y) &= -\frac{1}{N} \sum_{i=1}^{N} \widehat{\log f}(X_i, Y_i) \\ &= -\frac{1}{N} \sum_{i=1}^{N} \left(\psi(k) - \frac{1}{k} - \psi(N) - \log V_d - d \log \varepsilon(i) \right) \\ &= -\psi(k) + \frac{1}{k} + \psi(N) + \log V_d + \frac{d}{N} \sum_{i=1}^{N} \log \varepsilon(i). \end{aligned}$$

Como ya hicimos en el primer algoritmo, utilizaremos el Lema 2.2.20 para estimar la información mutua, sustituyendo la entropía conjunta por

el estimador recién conseguido y las individuales por el estimador obtenido en el caso anterior, notando que cambia el significado de n_x y n_y.

$$\begin{aligned} I^{(2)}(X,Y) &\approx \hat{h}(X) + \hat{h}(Y) - \hat{h}(X,Y) \\ &= -\frac{1}{N}\sum_{i=1}^{N}\psi(n_x(i)) + \psi(N) + \log V_{d_X} + \frac{d_X}{N}\sum_{i=1}^{N}\log\varepsilon(i) \\ &\quad -\frac{1}{N}\sum_{i=1}^{N}\psi(n_y(i)) + \psi(N) + \log V_{d_Y} + \frac{d_Y}{N}\sum_{i=1}^{N}\log\varepsilon(i) \\ &\quad +\psi(k) - \frac{1}{k} - \psi(N) - \log(V_{d_X}V_{d_Y}) - \frac{d_X+d_Y}{N}\sum_{i=1}^{N}\log(\varepsilon(i)) \\ &= \psi(k) - \frac{1}{k} - \frac{1}{N}\sum_{i=1}^{N}\big(\psi(n_x(i)) + \psi(n_y(i))\big) + \psi(N). \end{aligned}$$

La información mutua se aproximaría por

$$I^{(2)}(X,Y) \approx \psi(k) - \frac{1}{k} - \langle\psi(n_x) + \psi(n_y)\rangle + \psi(N). \tag{5}$$

En general ambas fórmulas dan resultados similares. Para un mismo k, $I^{(1)}$ da menores errores estadísticos (porque $n_x(i)$ y $n_y(i)$ tienden a ser más grandes y tener menores variaciones relativas), pero tiene mayores errores sistemáticos. Esto último es grave solo si estamos interesados en variables con dimensiones muy altas, donde $\varepsilon(i)$ es habitualmente más grande que la marginal $\varepsilon_x(i)$. En ese caso, parece preferible el segundo algoritmo, en caso contrario es indiferente usar uno u otro.

2.4 IMPLEMENTACIONES DE LA ENTROPÍA Y DE LA INFORMACIÓN MUTUA

En esta sección estudiaremos dos implementaciones en Python de conceptos anteriores como la entropía o la estimación mutua. En general, dado un número finito de muestras de una variable aleatoria no podemos calcular de forma exacta la entropía o información mutua, ya que no conocemos su distribución. Para implementarlas es necesario usar algún estimador, como los estudiados en el apartado anterior.

2.4.1 *Implementación 1 - GaelVaroquaux*

Comenzaremos revisando la implementación encontrada en [12]. Esta implementación fue desarrollada por Gaël Varoquaux [3], un investigador de la facultad de Inria que trabaja en el ámbito de la ciencia de

datos, contribuye, entre otros proyectos, en las bibliotecas `scikit-learn` y joblib. En esta implementación hace uso de la biblioteca `scikit-learn` para calcular estimadores de la entropía y la información mutua. Para revisar esta implementación analizaremos el código y nos haremos una idea de qué hace y cómo cada una de las funciones.

- `nearest_distances(X, k)`: para cada punto de un vector `X`, devuelve la distancia a su k-ésimo vecino más cercano. Se implementa haciendo uso de las funciones de la clase `NearestNeighbors` de `scikit-learn` [34].

- `entropy_gaussian(C)`: entropía de una variable gaussiana, se utilizará para hacer pruebas. Si la matriz de covarianzas `C` es escalar, se utiliza la siguiente fórmula para la entropía:

$$h(X) = \frac{1}{2}\left(1 + \log(2\pi)\right) + \frac{1}{2}\log C.$$

  Vimos en el ejemplo 2.2.17 que la entropía de la gaussiana con covarianza σ^2 es:

$$h(X) = \frac{1}{2}\log(2\pi e\sigma^2),$$

  coincidiendo con la implementación.

  Para el caso en que la matriz `C` sea no degenerada, utiliza la expresión:

$$h(X_1, \dots, X_n) = \frac{n}{2}\left(1 + \log(2\pi)\right) + \frac{1}{2}\log|C|,$$

  Coincidiendo con la fórmula vista en el ejemplo 2.2.18.

- `entropy(X, k)`: calcula la entropía de la variable `X` usando el estimador de tipo de Kozachenko - Leonenko visto en (3). Comienza inicializando los elementos que se necesitan para aplicar la fórmula: vecinos más cercanos para todos los elementos del vector, dimensiones de `X`, volumen de la bola unidad (V_d). Notamos que en el cálculo de la constante V_d (`volume_unit_ball`) se utiliza el volumen dado por la distancia euclídea. Donde en el estimador estudiado ponía ε en esta implementación pone `r + eps`, donde `eps` es el menor valor que al ser sumado varía el resultado (de forma que `1+eps` $\neq$ `1`). En ambas implementaciones se añade un ínfimo ruido a los datos, para evitar que varios puntos tengan exactamente las mismas coordenadas. Devuelve el resultado de sustituir estos valores en la expresión (3).

- `mutual_information(variables, k)`: devuelve la información mutua entre cualquier número de variables. Para ello suma las entropías marginales de cada una de ellas y les resta la entropía conjunta (usando el mismo valor k para todas las estimaciones), se aplica el Lema 2.2.20 realizando el cálculo de las entropías con la función `entropy`.

- `mutual_information_2d(x, y, sigma, normalized)`: calcula la información mutua entre dos variables unidimensionales a partir de su histograma conjunto.
- `test_entropy()`: permite validar la entropía comparando la entropía de una variable aleatoria con distribución normal, que conocemos teóricamente y se calcula en `entropy_gaussian`, con la entropía estimada por `entropy`. Si ambos valores difieren demasiado, de disparará un *assert*[1].
- `test_mutual_information()`: permite comprobar la información mutua de una variable aleatoria 2 dimensional estimando los resultados para una variable aleatoria con distribución normal y comparándolos con los resultados teóricos.
- `test_degenerate()`: comprueba que los estimadores no dan valores infinitos.
- `test_mutual_information_2d()`: es una función similar a `test_mutual_information`, pero para probar `mutual_information_2d`.

Si ejecutamos el código original, se producen como salida dos parejas. La primera es la estimación de la información mutua entre 2 variables normales correladas usando la función `mutual_information`. La segunda muestra la misma información para la función `mutual_information_2d`. En ambos casos a la izquierda está el valor obtenido y a la derecha el valor teórico. Se obtienen los siguientes resultados:

```
(0.1092329232318674, 0.11157177565710485)
(2.1173231240729757, 2.2033596236321267)
```

Añadimos a esta tupla la diferencia entre ambos valores:

```
(0.1092329232318674, 0.11157177565710485, 0.0023388524252374)
(2.1173231240729757, 2.2033596236321267, 0.08603649955915094)
```

Observamos una pequeña diferencia entre los valores estimados y los teóricos. Esta pequeña diferencia podría indicar que la estimación es buena. Sin embargo, para afirmar con certeza si la estimación es buena o no habría que fijar un umbral de error y probarla en muchos más casos.

2.4.2 *Implementación 2 - gregversteeg*

Pasamos a revisar la segunda implementación, que encontramos en [35]. Esta implementación, cuyo desarrollador principal es Greg Ver Steeg [2], investigador asociado a la universidad de California del sur, se basa

[1] Un *assert* en Python es una expresión que se debe cumplir, en caso contrario se dispara un error, si este error no es gestionado (como en este caso) provoca la finalización del programa.

en los estimadores estudiados, aquellos propuestos en [19]. Como en el caso anterior, comenzamos analizando el código función por función.

- `entropy(x, k, base)`: utiliza el estimador de tipo Kozachenko - Leonenko visto en (3) basado en los k vecinos más cercanos. `x` debe ser una lista de vectores. En primer lugar, se comprueba mediante un *assert* que el valor de k sea mayor que el número de muestras. Se convierte la lista `x` en un *array*, se almacena el número de elementos y de características, se añade ruido a los elementos de `x` (en la función `add_noise`), se obtiene el árbol de vecinos más cercanos usando árboles `KDTree` o `BallTree` de la biblioteca `scikit-learn` según la dimensión de los datos (en la función `build_tree`). Se obtienen los k vecinos más cercanos de `x` usando este árbol y la función `query_neighbors`, que llama a la función `query` correspondiente de `scikit-learn`, esta función devolverá la distancia de Chebyshev(dada por la norma del máximo) a los vecinos más cercanos y sus índices, pero nos quedamos solo con las distancias. La función devuelve el siguiente valor:

  $$\frac{\psi(N) - \psi(k) + d \cdot \log 2 + \frac{d}{N} \sum \varepsilon(i)}{\log(base)}.$$

  Dividir entre $\log(base)$ se utiliza para que la fórmula pase a estar en base 2. La fórmula coincide con la estudiada, notando que la constante $\log V_d$ es sustituida por $d \cdot \log 2$, debido al uso de la norma del máximo.

- `centropy(x, y, k, base)`: estima la entropía de $X|Y$, se usa la función `entropy` para calcular las entropías de (X, Y) e Y. Estima la entropía condicionada usando la definición $h(X|Y) = h(X, Y) - h(Y)$.

- `tc(xs, k, base)`: calcula la entropía de cada característica, devuelve la suma de las entropías de las características menos la entropía de la variable.

- `ctc(xs, y, k, base)`: similar a la función `tc`, pero calculando las entropías condicionadas.

- `corex(xs, y, k, base)`: análoga a la función `tc` pero en vez de la entropía calcula la información mutua.

- `mi(x, y, z, k, base, alpha)`: calcula la información mutua entre `x` e `y`, utilizando el estimador (4). Comienza comprobando que ambos vectores tienen la misma longitud y que k es menor que la misma. Hace las transformaciones con los datos correspondientes y les añade ruido. Si se ha pasado como parámetro también la `z`, la información mutua a calcular estará condicionada a este valor así

que se añade el valor `z` al conjunto de puntos. De forma similar a como se hizo en la función `entropy`, se construye el árbol de los vecinos más cercanos y se almacena la distancia de cada elemento al k-ésimo vecino más cercano. A continuación, se utiliza la función `avgdigamma` para encontrar el número de puntos a distancia menor que el vecino k-ésimo (`num_points`, en el espacio marginal) haciendo uso de `count_neighbors` y devuelve la media de $\psi(num\_points)$. Si el parámetro `alpha` $>$ `0` se suma a $\psi(N)$ el valor de `lnc_correction`. La función devuelve:

$$\frac{-\langle\psi(n_x)\rangle - \langle\psi(n_y)\rangle + \psi(k) + \psi(N)}{\log(base)}.$$

- `cmi(x, y, z, k, base)`: calcula la información mutua de `x` e `y` condicionado a `z`, hace uso de la anterior función, que en este caso devolvería

$$\frac{-\langle\psi(n_{xz})\rangle - \langle\psi(n_{yz})\rangle + \langle\psi(n_z)\rangle + \psi(k)}{\log(base)}.$$

- `kldiv(x, xp, k, base)`: devuelve la distancia de Kullback Leibler o entropía relativa entre `x` y `xp`. Comienza comprobando que las distribuciones tienen la misma dimensión y que el valor k escogido no supera al número de muestras. La implementación es similar a las ya vistas. En este caso se devuelve

$$\frac{\log len(xp) - \log(len(x) - 1) + d \cdot (\mathbb{E}(nnp) - \mathbb{E}(nn))}{\log(base)},$$

  donde `nn` y `nnp` son las distancias a los vecinos más cercanos de los puntos de `x` y `xp` respectivamente.

- `lnc_correction(tree, points, k, alpha)`: a partir del árbol de vecinos más cercanos, se queda con los k vecinos más cercanos y calcular el valor de la corrección que sumar al valor $\psi(N)$.

- `entropyd(sx, base)`: con esta comienzan una serie de funciones para estimaciones en el caso de variables aleatorias discretas. Se estima la probabilidad contando el número de apariciones de un elemento y dividiendo entre el total de elementos. Para calcular la entropía, se devuelve la suma de las probabilidades multiplicadas por el logaritmo de 1 partido por la probabilidad (por eso no aparece el cambio de signo), todo ello dividido entre $\log(base)$:

$$\frac{\sum p \log \frac{1}{p}}{\log(base)}.$$

  Si las probabilidades son correctas, esta fórmula coincide con la entropía teórica.

- `midd(x, y, base)`: se calcula la información mutua en el caso discreto utilizando la fórmula vista en el Teorema 2.2.13.a), restando a la entropía de `x`, la entropía de `x|y`.
- `cmidd(x, y, z, base)`: calcula la entropía de `(x,y)` condicionada a `z`. La implementación es similar a la del caso continuo, `cmi`, pero llamando a las funciones correspondientes para variables discretas.
- `centropyd(x, y, base)`: entropía de `x|y` para variables discretas. Se calcula como entropía conjunta menos entropía de y.
- `tcd(xs, base)`: a la entropía por características le resta la entropía total.
- `ctcd(xs, y, base)`: a la entropía condicionada por características le resta la entropía condicionada.
- `corexd(xs, ys, base)`: devuelve la suma de la información mutua por columnas menos la información mutua.
- `micd(x, y, k, base, warning)`: con este método empiezan los estimadores mixtos, en los que una variable es discreta y la otra continua. En esta función se calcula la información mutua cuando `x` sea continua e `y` sea discreta. Primero se asegura de que tengamos el mismo número de elementos de ambas variables. Se calcula la entropía de `x` y la probabilidad de cada valor de `y` (como en el caso discreto, contando el número de veces que aparece cada elemento y dividiendo entre el total). A continuación, para cada valor que toma `y` se calcula la entropía de `x` dado `y` (este valor comienza siendo 0 y se le va sumando la entropía calculada). Para hacer este cálculo, en primer lugar, se toma el vector de elementos de `x` dado un `y`, si tiene menos elementos que el k considerado se suma la probabilidad de `y` por la entropía de `x`, añadiendo un *warning* de que en este caso se está asumiendo la entropía máxima. Si el número de elementos es suficiente, se calcula la entropía de este vector y se multiplica por la probabilidad de `y`. La función devolverá el valor absoluto de la diferencia entre la entropía de `x` y la entropía de `x` dado `y`.
- `midc(x, y, k, base, warning)`: este estimador se utilizará cuando `x` sea discreta e `y` continua, consiste en llamar a la función `micd` con los argumentos intercambiados.
- `centropycd(x, y, k, base, warning)`: calcula la entropía de `x` (continua) condicionada a `y` (discreta). Devuelve la entropía de `x` menos la información mutua (utilizando el estimador mixto correspondiente) de `x, y`.
- `centropydc(x, y, k, base, warning)`: utiliza la función anterior con los parámetros `x, y` intercambiados.

- `ctcdc(xs, y, k, base, warning)`: devuelve la suma de la entropía condicionada por columnas menos la entropía condicionada cuando `x` es discreta e `y` continua.
- `ctccd(xs, y, k, base, warning)`: llama a la función `ctcdc` para `x` continua e `y` discreta.
- `corexcd(xs, ys, k, base, warning)`: devuelve el valor de `corexdc` intercambiando los parámetros.
- `corexdc(xs, ys, k, base, warning)`: devuelve la diferencia entre `tcd(xs)` y `ctcdc(xs, ys, k)`.
- Las funciones auxiliares `add_noise`, `query_neighbors`, `count_neighbors`, `avgdigamma` y `build_tree` se han ido comentando a medida que aparecían.
- `shuffle_test`: es una función a la que pasamos la medida a utilizar (entropía, información mutua, condicionadas, …) y los vectores `x` e `y`. Repite `ns` veces la medida barajando el vector `x`, devuelve la media y el intervalo de confianza de los resultados.

Además del archivo con los estimadores, en esta implementación se incluye un segundo archivo `test.py` en el que se realizan varias pruebas sobre los mismos. Si lo ejecutamos devuelve:

```
For a uniform distribution with width alpha, the differential
    entropy is log_2 alpha, setting alpha = 2
and using k=1, 2, 3, 4, 5
result: [0.9675938258710816, 1.0283110450873976,
   0.9611523946690828, 0.9875186838882535,
   1.0917394218821124]

Gaussian random variables

Conditional Mutual Information
covariance matrix
[[4 3 1]
 [3 4 1]
 [1 1 2]]
true CMI(x:y|x) 0.5148736716970265
samples used [10, 25, 50, 100, 200]
estimated CMI [0.2253106080187056, 0.3892879127835147,
   0.4398035226533797, 0.49760670969901555,
   0.512074966143492]
95% conf int. (a, b) means (mean - a, mean + b)is interval
 [(0.297216360850314, 0.4346078735815755),
    (0.3998450496742388, 0.4109331495973501 7),
    (0.32034107567533604, 0.34254631457985824),
```

```
    (0.23963575516458396, 0.2875658611969877),
    (0.17492887513865663, 0.18358453574822897)]
Mutual Information
true MI(x:y) 0.5963225389711981
samples used [10, 25, 50, 100, 200]
estimated MI [0.36751110421959715, 0.508806849561461,
   0.5987137761055255, 0.6182090681075484,
   0.6174309289477204]
95% conf int.
 [(0.39751000110157275, 0.5830073784740326),
    (0.46139075928905027, 0.43094430206983014),
    (0.31222200485617396, 0.3922391298958362),
    (0.250926144320218, 0.35084718632752154),
    (0.18857541697637004, 0.19108560308455347)]

IF you permute the indices of x, e.g., MI(X:Y) = 0
samples used [10, 25, 50, 100, 200]
estimated MI [-0.005738147775725934, -0.02720997970818474,
   0.0010112811193075707, -0.008517516095735011,
   0.0015400222685101536]
95% conf int.
 [(0.24707698319910223, 0.43997790512266205),
    (0.26641749338269727, 0.2214406621687565),
    (0.1889408437170837, 0.21236801559420707),
    (0.18470276983064027, 0.19294766099193206),
    (0.14993655857703755, 0.16121527948941508)]


Test of the discrete entropy estimators

For z = y xor x, w/x, y uniform random binary, we should get
   H(x)=H(y)=H(z) = 1, H(x:y) etc = 0, H(x:y|z) = 1
H(x), H(y), H(z) 1.0 1.0 1.0
H(x:y), etc 0.0 0.0 0.0
H(x:y|z), etc 1.0 1.0 1.0


Kl divergence estimator (not symmetric, not required to have
   same num samples in each sample set
should be 0 for same distribution
result: -0.03587664487774093
should be infinite for totally disjoint distributions (but
   this estimator has an upper bound like log(dist) between
   disjoint prob. masses)
result: 7.818662829432822
```

```
Test discrete.
random: I(X; Y) = 0.0279 ± 0.0048 (maximum possible 2.3194)
deterministic: I(X; Y) = 3.3168 ± 0.0019 (maximum possible
   3.3168)
noisy: I(X; Y) = 2.7575 ± 0.0264 (maximum possible 3.3168)
```

En primer lugar, se calcula la entropía de variables con una distribución uniforme en un intervalo $[0, \alpha]$. En el ejemplo 2.2.16 calculamos la entropía teórica, $h(x) = \log \alpha$. En este caso se tomó $\alpha = 2$, luego la entropía teórica vale 1. Observamos que para los valores k siempre hay una pequeña diferencia entre el valor obtenido y la entropía teórica. A continuación, calcula la información mutua (condicionada y sin condicionar) de una variable aleatoria con distribución normal, de forma teórica y usando el estimador. Por último, prueba los estimadores para el caso discreto.

3

METODOLOGÍA PARA LA COMPARACIÓN DE ESTIMADORES

3.1 TECNOLOGÍA UTILIZADA

El lenguaje de programación utilizado para implementar las diferentes pruebas que nos permiten comparar las implementaciones de los estimadores presentadas anteriormente es Python [38]. No solo es el lenguaje en el que están escritas las implementaciones, es un lenguaje con numerosos módulos y paquetes, tanto en su biblioteca estándar como fuera de la misma, que nos ayudarán a realizar la comparativa desde diferentes puntos de vista.

Python es un lenguaje interpretado, interactivo y orientado a objetos. Incluye módulos, excepciones, tipado dinámico. Soporta varios paradigmas de programación además de la programación orientada a objetos se permite la programación procedural o funcional. Una de las características principales de este lenguaje es su clara sintaxis.

Entre las bibliotecas destacamos el uso de NumPy [24] y pandas [25] para almacenar información, matplotlib [21] para la generación de gráficas, random [29] para generar los conjuntos de datos con los que realizar pruebas, cProfile y pstats [36] para trabajar con las medidas de los tiempos de ejecución.

Para la construcción de los programas se ha utilizado un diseño basado en prototipos. El programa inicial es revisado y mejorado sucesivas veces hasta obtener la versión final. De esta forma, podremos ir viendo cómo se desarrollan las diferentes pruebas para comparar los estimadores si lo están haciendo de forma correcta o no e ir añadiendo funcionalidades o mejoras a medida que nos demos cuenta de que son necesarias. Se evaluó la posibilidad de usar TDD (*Test-Driven Development* o desarrollo guiado por pruebas), pero para este proyecto, esta metodología quedó descartada.

Para tener control sobre las diferentes versiones se ha utilizado Git [13], un sistema de control de versiones libre que nos permitirá añadir las nuevas versiones de forma que podamos observar con facilidad los cambios entre las versiones así como deshacerlos si fuera necesario. El repositorio utilizado en el desarrollo de este trabajo se encuentra en [5].

A continuación, se explican con más detalle las herramientas utilizadas para medir el tiempo de ejecución. Además, se describirá cómo funciona el paralelismo en Python, la existencia del cerrojo global del intérprete (GIL) y cómo realizar el cálculo de los vecinos más cercanos utilizando la biblioteca `scikit-learn`, ya que ambas implementaciones hacen uso de la misma.

3.1.1 *Herramientas para medir el tiempo de ejecución*

Puesto que estamos comparando dos implementaciones diferentes que deben realizar el mismo cálculo, además de la precisión de sus resultados, podemos medir sus tiempos de ejecución para comprobar si hay o no diferencias significativas entre ellos. Para ello, se utilizará la herramienta `cProfile`, que nos permite hacer *profiling* determinístico de programas escritos en Python, y la clase `Stats` que nos permite acceder a los medidas realizadas por la herramienta anterior [36].

La herramienta `cProfile` mide el número de veces que se ejecuta cada función (`ncalls`), el tiempo total empleado en cada función (`tottime`), el tiempo empleado en cada función cada vez que se llamó a la función (`percall`), el tiempo acumulado de cada función y todas a las que estas llamen (`cumtime`) y el tiempo acumulado entre el número de veces que se llame a esta función (`percall`). Esta herramienta será precisa incluso en el caso de funciones recursivas, al distinguir el número de veces que se llama a la función del número de veces que fue invocada de forma recursiva.

El uso de esta herramienta se complementa con la utilización de la clase `Stats`, que permite trabajar con las medidas realizadas por `cProfile`. Usando esta clase podremos ordenar y especificar el número o nombre de las funciones cuyos tiempo queremos observar, así como almacenar en archivos la información para consultarla más adelante.

Por ejemplo, si quisiéramos obtener información sobre los tiempos de ejecución de la función `function` y todas a las que esta llama para saber cuáles emplean más tiempo, podríamos ejecutar el siguiente código:

```
# Activamos el profiler
pr = cProfile.Profile()
pr.enable()

# Ejecutamos la función
function(*args)
```

```
# Desactivamos el profiler
pr.disable()

# Creamos objeto de la clase Stats
stats = pstats.Stats(pr).strip_dirs()

# Imprimimos las 10 subfunciones que consuman más tiempo
stats.sort_stats(pstats.SortKey.TIME).print_stats(10)
```

3.1.2 *Paralelismo en Python*

El uso del paralelismo a nivel de hebra en Python se ve limitado por el cerrojo global del intérprete (GIL). A continuación, veremos qué es GIL y algunas herramientas para desarrollar código en paralelo, a nivel de hebras o de procesos en Python.

GIL, abreviatura de *global interpreter lock*, [14], [18], es el mecanismo usado por el intérprete de Python para asegurar que solo se ejecuta una hebra en cada momento. Su objetivo es hacer más seguro el acceso concurrente a memoria y esto lo consigue al poner un cerrojo sobre el intérprete al completo. Sin este cerrojo, se podrían dar problemas de sincronización en programas multihebra.

Algunos módulos, de la librería estándar o no, liberan este cerrojo para ejecutar código muy costoso computacionalmente. También es habitual liberarlo en operaciones de entrada o salida, que son más lentas, liberando este cerrojo se permite la ejecución de otras hebras en lo que se completan las operaciones de entrada o salida.

El uso de GIL es controvertido, ya que no permite que programas multihebra aprovechen sistemas multiprocesadores. Se ha intentado crear un intérprete que permita la ejecución de múltiples hebras simultáneas sin un cerrojo global, sino con cerrojos locales. Sin embargo, no se ha conseguido, ya que los intentos anteriores bajaban el rendimiento en sistemas monoprocesador. Además, se cree que solucionar este problema haría la implementación mucho más complicada y, por ello, más difícil de mantener.

El módulo `threading` [37] de la biblioteca estándar de Python proporciona una interfaz de alto nivel para trabajar con hebras en este lenguaje. La clase `Thread` permite crear y ejecutar hebras diferentes a la principal. Contiene un constructor para crearlas indicando la función a ejecutar, el método `start()` que permite comenzar su actividad y el método `join()` para esperar a que la hebra termine, entre otros. Sin embargo, debido al cerrojo global (GIL) solo se puede ejecutar una hebra a la vez. El parale-

lismo a nivel de hebra podría ser apropiado cuando queramos ejecutar tareas cuyo cuello de botella sea la entrada salida, en cuyo caso se liberaría este cerrojo.

Una solución para poder ejecutar código en paralelo es usar múltiples procesos en vez de múltiples hebras. El paquete `multiprocessing` [23] de la biblioteca estándar de Python permite la ejecución concurrente de procesos, evitando el cerrojo global del intérprete al usar subprocesos en vez de hebras. La clase `Process` es análoga a la clase `Thread` para el caso de procesos. Como los procesos no comparten memoria es necesario implementar un método de comunicación entre procesos, para esto están la clase `Queue` y la función `Pipe()`. Además, se proporcionan varias alternativas para sincronizar los procesos como cerrojos (`Lock`), semáforos (`Semaphore`) o barreras (`Barrier`). Así, como una clase (`Pool`) para gestionar las tareas que queremos que ejecuten los diferentes procesos.

No solo existen estos módulos para tratar el paralelismo, de hecho, en la documentación de Python se referencian algunas bibliotecas para ello [27]. Entre estas comentaremos `Joblib`, pues es la que utilizan las funciones de `scikit-learn` usadas en las implementaciones estudiadas.

Uno de los objetivos principales de `Joblib` [9] es permitir la ejecución en paralelo de forma sencilla. Esta biblioteca nos permite hacer uso de la concurrencia a nivel de hebra o de proceso según le indiquemos, notamos que a nivel de hebra solo será efectivo si el GIL ha sido liberado. Por defecto, utilizará el módulo `'loky'` para separar procesos y ejecutar tareas simultáneamente en varios procesadores, pero es posible indicarle que utilice el módulo `threading` o `multiprocessing`.

3.1.3 *Cálculo de los vecinos más cercanos en* `scikit-learn`

La biblioteca `scikit-learn` proporciona funcionalidades para el uso métodos basados en los k vecinos más cercanos [1].

Para el cálculo de los k vecinos más cercanos de un conjunto de puntos se puede utilizar la clase `NearestNeighbors` [34] y su método `kneighbors`, que devuelve los índices y distancias de los vecinos de cada punto. De esta forma se realiza el cálculo de los vecinos más cercanos en la implementación 1 de la entropía. Internamente, la clase `NearestNegihbors` utiliza las clases `KDTree` y `BallTree` para encontrar los vecinos más cercanos. La implementación 2 crea directamente uno de estos árboles (según la dimensión) e invoca directamente los métodos que va necesitando.

Si nos encontramos con un conjunto de n puntos, cada uno de dimensión d, podremos utilizar uno de los tres algoritmos proporcionados por la biblioteca `scikit-learn` para el cálculo de los vecinos más cercanos: `brute force`, `K-D Tree` y `Ball Tree`.

`Brute force` es un método de búsqueda por fuerza bruta, realizará todos los cálculos para saber qué puntos están más próximos entre sí. Aunque resulta competitivo para muestras pequeñas, su eficiencia en tiempo es $O(dn^2)$.

Con el objetivo de obtener algoritmos más eficientes que el de fuerza bruta, se crean estructuras de datos basadas en árboles para tratar de minimizar el número de cálculos de distancias. Se puede utilizar un árbol k dimensional (`KDTree`), que es un árbol binario en el que se van repartiendo recursivamente los puntos según su valor respecto de alguno de los ejes. La construcción de este árbol es muy rápida, porque no requiere el cálculo de distancias d dimensionales, es en la búsqueda de un k vecino más cercano cuando hay que calcularlas. El uso de esta estructura será eficiente para dimensiones menores que 20 aproximadamente, con eficiencia en tiempo $O(d \log n)$, sin embargo, a medida que crece la dimensión, se vuelve ineficiente con un coste en tiempo casi de $O(dn)$.

Para solucionar la ineficiencia de los árboles k dimensionales para dimensiones altas, se desarrolló la estructura *ball tree*. En ella, en vez de dividir los puntos según sus valores respecto a algún eje, se hace según una serie de hiperesferas. La construcción de este tipo de árboles es más costosa que la de los árboles k dimensionales, pero mejora la eficiencia de estos para realizar búsquedas cuando las dimensiones sean altas, sus tiempos son del orden $O(d \log n)$.

Para un número de muestras muy pequeño, $n < 30$, los algoritmos de fuerza bruta podrían resultar más eficientes que los basados en árboles. Por ello, las clases basadas en árboles, proporcionan un parámetro `leaf_size` que controla el número de muestras para el que utilizar el algoritmo de fuerza bruta. Este parámetro afecta a los tiempos de ejecución de las aproximaciones basadas en árboles, ya que el árbol se desarrollará hasta que el número de muestras en un nodo hoja alcance el valor `leaf_size`, para calcular la distancia entre puntos de un nodo hoja se utilizará el algoritmo de fuerza bruta. Cuanto mayor sea `leaf_size`, menor será el tiempo de construcción del árbol, ya que se crean menos nodos. Para buscar un k vecino más cercano, si `leaf_size` es muy pequeño, el tiempo de recorrer los nodos puede elevar los tiempos de ejecución, pero si es muy grande, las consultas se realizarán prácticamente por fuerza bruta.

3.2 *T-TEST* PARA COMPARAR LOS ESTIMADORES

Uno de los objetivos de este trabajo es conocer el comportamiento de las implementaciones de la entropía y la información mutua. Queremos saber si, en media, el comportamiento de ambos estimadores es similar. Para ello, tomaremos como hipótesis nula, H_0, que los estimadores en media tienen el mismo comportamiento y como hipótesis alternativa, H_1, que las diferencias en el comportamiento de los estimadores son significativas; y realizaremos un contraste de hipótesis. Para llevar a cabo el contraste de hipótesis nos basaremos en un estadístico al que llamaremos t. Calcularemos el p-valor, la probabilidad de que bajo la hipótesis nula la variable aleatoria T, definida a continuación, pudiera obtener valores más extremos que el valor observado en t. Cuando este sea muy bajo, la hipótesis nula no se corresponderá con los datos observados y podremos rechazarla.

En este caso, los valores de los estimadores dependen de la muestra, por ello, en vez de comparar la media de los estimadores en diferentes datos de muestra, para cada muestra se realiza la estimación con ambas implementaciones y se calcula la diferencia entre ellas, emparejando de esta forma las observaciones. Estas son observaciones $d_1, \dots, d_N$ de la muestra aleatoria $D_1, \dots, D_N$ de la variable aleatoria "diferencia entre ambas estimaciones", se asumirá que sigue una distribución normal, con media μ_D y varianza σ_D^2. La hipótesis nula será,

$$H_0 : \mu_D = 0, \text{ es decir, la diferencia entre las medias es nula.}$$

La variable T se define como

$$T = \frac{\overline{D} - \mu_D}{S_d/\sqrt{N}}.$$

El estadístico t vendrá dado por

$$t = \frac{\bar{d}}{s_d/\sqrt{N}},$$

donde s_d^2 es la varianza muestral. El p-valor, p, a partir de él se calcula como

$$p = 2P\left[T > t | \mu_D = 0\right].$$

Utilizaremos la implementación del *test* encontrada en [31]. Dados dos vectores medirá si la diferencia entre ellos es significativa o no, devuelve el valor del estadístico t y el p-valor. Lo utilizaremos dos veces, en un caso pasándole como parámetro las estimaciones de N valores de la entropía realizadas con ambos estimadores, en otro, las estimaciones de N valores de la información mutua realizadas con ambos estimadores.

4

EXPERIENCIA COMPUTACIONAL Y DISCUSIÓN DE RESULTADOS

4.1 CONDICIONES DE EXPERIMENTACIÓN

Los experimentos mostrados en este capítulo se han realizado en un equipo con las siguientes características:

- Arquitectura x86_64.
- Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz.
- 1 *socket*, 4 núcleos por *socket*, 2 hilos de procesamiento por núcleo.
- 16 GB RAM.
- Sistema operativo Pop!_OS 20.04 LTS.

Se ha utilizado la versión de Python 3.8.2, así como las siguientes versiones de los paquetes: NumPy 1.18.4, pandas 1.0.3, scikit-learn 0.23.1, joblib 0.15.1 y matplotlib 3.2.1.

Además, para generar las muestras se ha creado la clase `data.py`, donde se crea un conjunto de datos de tamaño y dimensión dada y que siga una distribución normal, uniforme o que sea aleatoria.

4.2 ANÁLISIS DE LOS ESTIMADORES

Pasamos a implementar el programa `compare.py` para poder comparar ambas implementaciones y realizar las modificaciones que consideremos.

4.2.1 *Estimadores de la entropía*

Comenzaremos comparando las estimaciones de la entropía. Para ello, creamos la función `ent` a la que pasamos dos variables aleatorias y calculamos su entropía a partir de los estimadores dados por las dos implementaciones.

```
Entropía de X:
Estimador 1: 29.423303907819346
```

```
Estimador 2: 43.499531309829294
Entropía de Y:
Estimador 1: 4.232705400287453
Estimador 2: 6.084047562318489
```

Los estimadores obtienen resultados notablemente diferentes. Esto se debe a que uno de ellos nos está dando la entropía en bits (al pasarlo a base 2) y el otro en nats. Modificaremos el primer estimador, dividiendo su resultado entre $\log(2)$, para que los dos estén en las mismas unidades.

Pasamos a comparar estas entropías con una que conozcamos teóricamente, la distribución uniforme, vista en el Ejemplo 2.2.16, cuya entropía teórica es $\log(a)$. El código utilizado se encuentra en la función `example_ent_unif` y los resultados obtenidos son los siguientes:

```
Entropía x: (a = 2)
Teórica: 1.0
Estimador 1: 1.0261633399126422
Estimador 2: 1.0261633406219435

Entropía x: (a = 100)
Teórica: 6.643856189774725
Estimador 1: 6.683574884777654
Estimador 2: 6.68357488477827
```

Las estimaciones son similares, coinciden con la entropía teórica en el primer decimal y entre hasta el undécimo decimal. Notamos que en este caso la variable aleatoria X es unidimensional.

Para comparar las estimaciones de la entropía en una variable multidimensional, aprovechamos la función `test_entropy` de la primera implementación y la adaptamos para calcular también la entropía con la segunda implementación. Además, añadimos el parámetro `d`, con el que podremos variar la dimensión de la variable considerada, y el parámetro `k` que marcará el número de vecinos a utilizar. La matriz `P`, utilizada para generar la matriz de covarianzas, ya no está definida de forma fija, sino que se define de forma aleatoria según la dimensión a utilizar. Comenzamos a estudiar el caso d = 2, las estimaciones son parecidas entre sí (suelen coincidir en varios decimales), si ejecutamos la función en bucle, en general se obtiene una pequeña diferencia con el estimador real, pero no se llega a disparar el *assert*. Este salta algunas veces, cuando la entropía es negativa, si los estimadores dan un resultado menor que la entropía real.

Queremos ser realmente conscientes de la magnitud del error que se está cometiendo, para ello, implementamos la función `err_entropy`. Esta función no es más que una modificación de `test_entropy` en la que se eliminan los *assert* y se calcula la diferencia entre la entropía gaussiana

teórica y las estimadas. Notamos que si aumentamos el valor de d, ambas estimaciones obtienen valores próximos, difiriendo de la entropía teórica.

Creamos la función `exp_err_ent`. En ella, se pretende obtener un idea más precisa del error obtenido para cada dimensión con ambos estimadores. Para ello, se mide el error un número (`reps`) determinado de veces y se calcula la media del error obtenido para cada dimensión. Finalmente, se dibuja una gráfica del error obtenido. Ejecutando esta función con `reps = 20`, `k = 3`[1], `d = 2, ..., 10`, se obtienen los errores que podemos observar en la Tabla 1, nótese cómo a medida que aumenta la dimensión, aumenta el error de forma similar para ambos estimadores.

Tabla 1: Errores obtenidos con ambas estimaciones al calcular la entropía de una variable aleatoria con distribución normal y compararla con la entropía teórica.

d	Error estimador 1	Error estimador 2
2	0.014212339128172302	0.014680402744924015
3	0.11105323234117655	0.1228230352418274
4	0.16456589959078247	0.1634349916264736
5	0.699756197647123	0.7335137192470153
6	1.11385415187634	1.1622332193590927
7	1.2283705866481953	1.2922942070634487
8	1.5855188720644953	1.6860491894433907
9	2.262297509187	2.436619705610153
10	2.433512217901498	2.660198051294997

Representando gráficamente los valores anteriores compararemos con más facilidad el error obtenido con ambos estimadores. En la Figura 1 encontramos esta representación. Vemos que los errores obtenidos con los dos estimadores eran prácticamente 0 para $d = 2$ y que estos van aumentando con la dimensión, siendo ligeramente superior el error obtenido con el segundo estimador, 20 repeticiones por iteración.

El tiempo de cómputo a medida que aumentamos la dimensión se incrementa considerablemente por lo que para hacernos una idea del comportamiento de los estimadores al aumentar la dimensión, repetimos el experimento con menos repeticiones por dimensión y llegamos hasta $d = 15$. Se obtienen los errores mostrados en la Figura 2. Notamos que los errores no superan al anterior en todos los casos, se puede deber a que en las 10 repeticiones se obtengan estimaciones de la entropía que fueran mejores y ayuden a disminuir el error. Además es destacable cómo, a medida que aumenta la dimensión, crece la diferencia entre las dos

[1] En [19] proponen utilizar un valor k entre 2 y 4 excepto cuando se está comprobando la independencia.

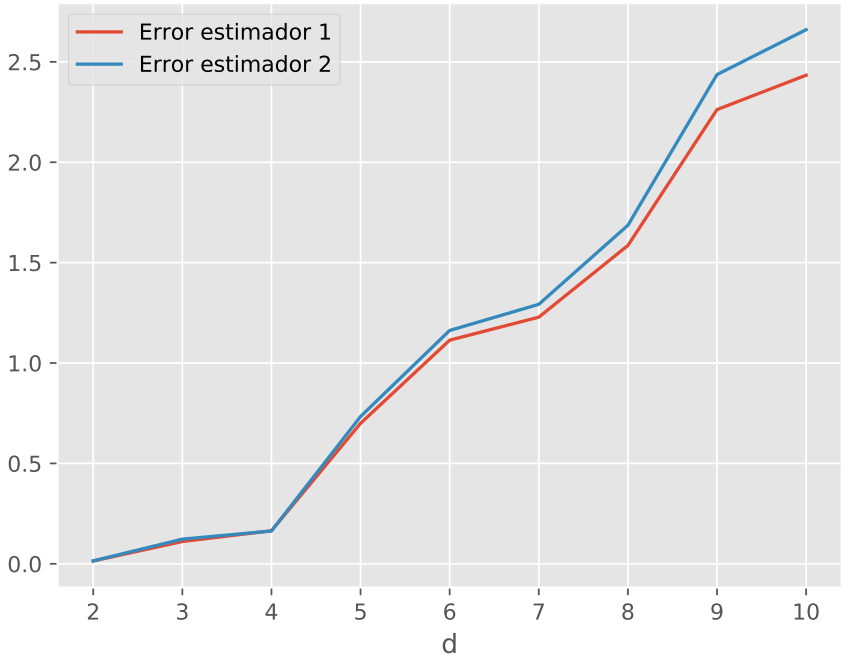


Figura 1: Errores obtenidos con ambas estimaciones al calcular la entropía de una variable aleatoria con distribución normal y compararla con la entropía teórica.

estimaciones.

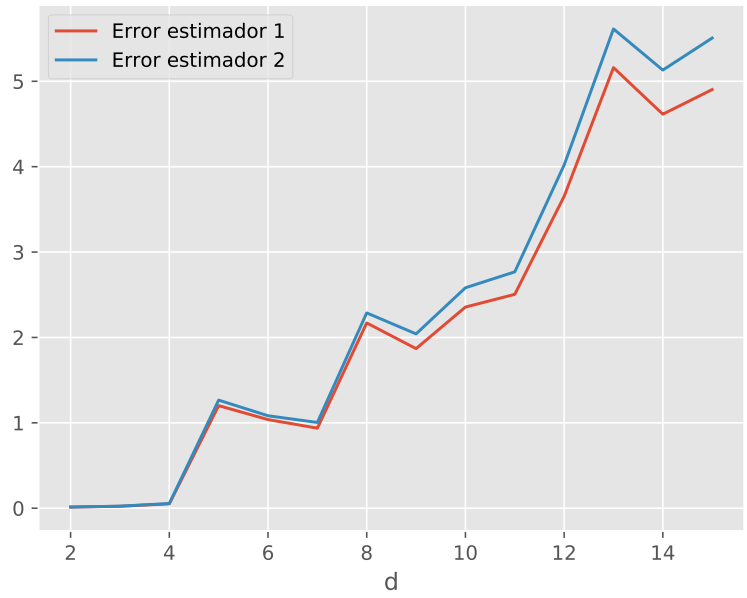


Figura 2: Errores obtenidos con ambas estimaciones al calcular la entropía de una variable aleatoria con distribución normal y compararla con la entropía teórica, 10 repeticiones por iteración.

4.2.2 *Estimadores de la información mutua*

A continuación, realizamos el estudio comparativo para los valores de las estimaciones de la información mutua. Las implementaciones realizan la estimación de la información mutua de forma diferente. La implementación 1 utiliza la fórmula que calcula la información mutua como suma de las entropías marginales de las variables y resta de la entropía conjunta. Los errores encontrados en los cálculos individuales podrían afectar al cálculo total. La implementación 2 utiliza el estimador (4) para realizar su estimación de la entropía, es posible que los errores cometidos al calcular la entropía no se vean arrastrados.

Realizamos pruebas comparando los resultados de los estimadores con los de una gaussiana (en la función `test_mutual_information_mod`) y calculamos el error como la diferencia entre información mutua teórica y estimada, calculamos la media tras repetir el experimento 100 veces, con $d = 2$ y $k = 3$. Se obtienen los siguientes resultados:

```
Error estimador 1: 0.010671581846025138
Error estimador 2: 0.007051668015342052
```

El error en el primer estimador es ligeramente superior al error en el segundo.

Al pasar a dimensión 3, los estimadores comienzan a diferir notablemente de la entropía teórica. Revisando el código notamos que no se adapta al cambio de dimensión directamente, el cálculo de la entropía teórica de la gaussiana se hace sumando las entropías marginales y restándosela a la conjunta (en este caso como la entropía se calcula correctamente no se arrastran errores), pero la función estaba originalmente desarrollada para el caso $d = 2$. Se modifica utilizando un bucle sobre la dimensión para que, efectivamente, sume todas las entropías marginales. Una vez realizada la modificación correspondiente, realizamos la prueba para $k = 3,\ d = 3$ y 100 repeticiones, con lo que obtenemos:

```
Error estimador 1: 1.6417490587527461
Error estimador 2: 1.6426795618610046
```

Al igual que en el análisis de la entropía, crearemos una función, `exp_err_im`, en la que repetiremos la prueba para las dimensiones dadas. Repitiendo el experimento para $d = 2, ..., 15$; 25 veces por dimensión (calculando la media del error), con $k = 3$ se obtienen los errores mostrados en la Tabla 2. Notamos que los errores van aumentando al ir añadiendo más dimensiones. Visualizamos los errores obtenidos en la Figura 3. Destaca que los errores obtenidos con ambas estimaciones son similares, de hecho, se superponen en la gráfica. El error en la estimación es creciente en la dimensión.

Tabla 2: Errores obtenidos con ambas estimaciones al calcular la información mutua de variables aleatorias con distribución normal y compararla con la información mutua teórica.

d	Error estimador 1	Error estimador 2
2	0.00778954666461035 86	0.00587913548125665 2
3	1.852406411791869	1.8495568835932743
4	2.328067362081687	2.330375193324987
5	4.022186430366867	4.022166466891652
6	4.552915119298815	4.555859073764772
7	5.325584640708861	5.327399887079806
8	5.696270621561146	5.696928152061884
9	6.86921976358911	6.871685971363305
10	7.492106258014951	7.4935380268611915
11	8.401680241357042	8.400306763900506
12	9.010038140658828	9.011324360858382
13	9.983816029579954	9.986518788178701
14	10.470383330007241	10.468634751932003
15	11.476246581716687	11.47645425484027

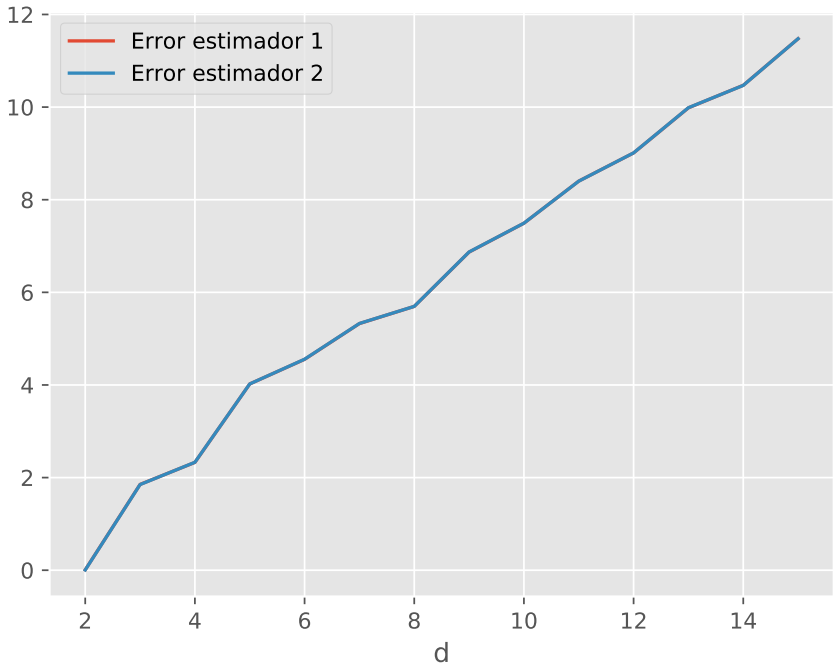


Figura 3: Errores obtenidos con ambas estimaciones al calcular la información mutua de variables aleatorias con distribución normal y compararla con la información mutua teórica.

4.3 ANÁLISIS MEDIANTE *T-TEST*

Nos preguntamos si el comportamiento en media de los estimadores de ambas implementaciones es el mismo, por ello, en el archivo `ttest.py` se escribe el código necesario para llevar a cabo un *T-Test*.

Tomaremos muestras de tamaño $N = 30$, un número superior incrementaría notablemente los tiempos de ejecución. Realizaremos el *test* para diferentes grupos de muestras, según la dimensión d y el tamaño de muestra n utilizado para realizar la estimación. Consideraremos dimensiones pequeñas, $d = 2$, medianas, $d = 10$, y grandes, $d = 25$; a su vez, tamaños de muestra pequeños, $n = 1000$, medianos $n = 30000$, y grandes, $n = 100000$[2].

Realizamos un *test* de normalidad sobre la muestra, de esta forma nos aseguramos de que nos encontramos bajo las condiciones del *T-Test*.

Recordamos que el contraste de hipótesis a realizar toma como hipótesis nula que los estimadores son similares y como hipótesis alternativa que los estimadores son significativamente diferentes. Calcularemos el

[2] El caso de dimensión grande y tamaño de muestra grande, resulta demasiado costoso computacionalmente por lo que se omite en este análisis

valor del estadístico t y su p-valor asociado en los diferentes casos. Si el p-valor es menor o igual que $0{,}05$ supondremos que tenemos evidencia suficiente de que los estimadores son distintos y rechazaremos la hipótesis nula. En caso contrario, no tendremos evidencia suficiente para rechazar la hipótesis nula y concluiremos que los estimadores en media son similares.

En la Tabla 3 se recogen los valores del estadístico t y el p-valor obtenido en los diferentes casos en los que se ha estimado la entropía. Vemos cómo el valor absoluto del estimador t aumenta a medida que aumenta la dimensión. Esto nos dice que la media muestral de las diferencias entre los estimadores crece con la dimensión. El p valor, al contrario, disminuye a medida que aumenta la dimensión. Podemos atender a la Tabla 4 para saber en qué casos rechazamos la hipótesis nula y en cuáles no. Para $d = 2$ no tenemos evidencia suficiente para rechazar la hipótesis nula, por tanto, concluimos que en media, para dimensiones pequeñas, los estimadores tienen un comportamiento similar. Al aumentar la dimensión, se observa suficiente evidencia de que los estimadores son diferentes, por lo que se rechaza la hipótesis nula y se afirma que su comportamiento es diferente en media. Notamos que estos resultados no varían según la dimensión de la muestra.

Tabla 3: Resultados del *T-test* para comparar la diferencia entre las estimaciones de la entropía para diferentes valores de d y n.

d \ n		1000	30000	100000
2	p	0,77008	0,051889	0,97277
	t	0,29502	−2,0275	−0,034430
10	p	$8{,}7473 \cdot 10^{-22}$	$1{,}2869 \cdot 10^{-22}$	$2{,}1281 \cdot 10^{-23}$
	t	−26,302	−28,169	−30,034
25	p	$8{,}6948 \cdot 10^{-43}$	$2{,}6519 \cdot 10^{-61}$	
	t	−142,08	−618,45	

Veremos ahora qué ocurre en el caso de la estimación de la información mutua. Si ambos estimadores calcularan la información mutua a partir de la entropía podríamos suponer que en los casos en que las diferencias en la estimación de la entropía fueran significativas, también lo serían en el caso de la información mutua; pero como en la implementación 2 la estimación de la información mutua no se hace a partir de la entropía, repetiremos el *test* para el caso de la información mutua para comprobar en qué casos las diferencias son significativas y en qué casos no lo son.

Tabla 4: Resultado del contraste de hipótesis para comparar la diferencia entre las estimaciones de la entropía para diferentes valores de d y n mediante el *T-Test*. Rechazo significa que rechazamos la hipótesis nula, No rechazo, que no la rechazamos.

d \ n	1000	30000	100000
2	No rechazo	No rechazo	No rechazo
10	Rechazo	Rechazo	Rechazo
25	Rechazo	Rechazo	

Los valores del estadístico t y sus p-valores asociados al calcular las diferencias entre las implementaciones de la información mutua se encuentran en la Tabla 5. En este caso, la media de las diferencias entre los estimadores normalizada que nos da el valor t aumenta a medida que crece el tamaño de la muestra y la dimensión. Los p-valores en este caso son menores y disminuyen a medida que aumenta el tamaño de muestra y la dimensión de sus elementos. Los casos en los que se rechaza o no se rechaza la hipótesis nula se encuentran en la Tabla 6. A la vista de los menores p valores no nos extraña que en este caso se rechazará la hipótesis nula en todos los casos. Las diferencias en media entre ambas implementaciones son significativas.

Tabla 5: Resultados del *T-test* para comparar la diferencia entre las estimaciones de la información mutua para diferentes valores de d y n.

d \ n		1000	30000	100000
2	p	$1{,}9822 \cdot 10^{-05}$	$3{,}5483 \cdot 10^{-14}$	$4{,}3197 \cdot 10^{-16}$
	t	5,0887	13,676	16,231
10	p	$9{,}1502 \cdot 10^{-43}$	$3{,}7462 \cdot 10^{-57}$	$1{,}5652 \cdot 10^{-61}$
	t	−141,83	−444,82	−629,79
25	p	$1{,}2286 \cdot 10^{-46}$	$6{,}7789 \cdot 10^{-73}$	
	t	−192,95	−1552,6	

Tabla 6: Resultado del contraste de hipótesis para comparar la diferencia entre las estimaciones de la información mutua para diferentes valores de d y n mediante el *T-Test*. Rechazo significa que rechazamos la hipótesis nula, No rechazo, que no la rechazamos.

d \ n	1000	30000	100000
2	Rechazo	Rechazo	Rechazo
10	Rechazo	Rechazo	Rechazo
25	Rechazo	Rechazo	

4.4 ANÁLISIS DE RENDIMIENTO

Para realizar la comparación de los tiempos de ejecución de las funciones se ha utilizado el archivo `profiling.py`.

Se ha realizado un análisis de los tiempos de ejecución de las dos implementaciones de la entropía y de las dos implementaciones de la información mutua estudiadas. Para ello, se han realizado 5 ejecuciones y nos hemos quedado con los tiempos medios de ejecución para diferentes valores del número de muestras y dimensión de las mismas. Remarcamos que el tiempo de ejecución considerado es el tiempo acumulado por la función a estudiar y todas las funciones a las que esta invoque.

Se han considerado tres rangos para el tamaño de las muestras: pequeñas (1000 muestras), medianas (30000 muestras) y grandes (100000). De forma similar, en función a la dimensión, se realizaron estimaciones con tres tipos de dimensiones: pequeñas (dimensión 2), medianas (dimensión 10) y grandes (dimensión 25).

Comenzaremos analizando los tiempos obtenidos al calcular la entropía. En la Tabla 7 podemos ver los tiempos de de ejecución, en segundos, de la función `entropy` de la primera implementación para los valores de n y d estudiados. Como es de esperar, a medida que aumenta el número de muestras y la dimensión, se incrementan los tiempos de ejecución en ambos casos. Observamos también la desviación típica de las medidas realizadas, para constatar que el número de medidas tomadas es suficiente. Consideramos que las desviaciones típicas obtenidas son lo suficientemente pequeñas en relación a los tiempos medios de ejecución.

La Tabla 8 contiene la misma información, esta vez sobre la función `entropy` de la implementación 2. En esta implementación de la entropía, los tiempos también aumentan al incrementar el número de muestras y la dimensión. Las desviaciones típicas vuelven a ser no significativas de

Tabla 7: Tiempos de ejecución de la función `entropy` de la implementación 1 para diferentes valores de n y d.

(a) Media de los tiempos obtenidos en las 5 ejecuciones.

d \ n	1000	30000	100000
2	0.0020 s	0.0774 s	0.4067 s
10	0.0149 s	2.5013 s	16.0703 s
25	0.1158 s	34.2677 s	654.9519 s

(b) Desviaciones típicas de los tiempos obtenidos.

d \ n	1000	30000	100000
2	0.0003 s	0.0064 s	0.0119 s
10	0.0005 s	0.2573 s	0.6337 s
25	0.0019 s	2.2058 s	10.1255 s

acuerdo a los datos medios.

Tabla 8: Tiempos de ejecución de la función `entropy` de la implementación 1 para diferentes valores de n y d.

(a) Media de los tiempos obtenidos en las 5 ejecuciones.

d \ n	1000	30000	100000
2	0.0017 s	0.0648 s	0.2515 s
10	0.0071 s	0.8797 s	5.6956 s
25	0.0475 s	30.6916 s	876.7703 s

(b) Desviaciones típicas de los tiempos obtenidos.

d \ n	1000	30000	100000
2	0.0001 s	0.0054 s	0.0137 s
10	0.0002 s	0.0823 s	0.2450 s
25	0.0001 s	0.1998 s	12.1683 s

Para realizar la comparación atenderemos a la Tabla 9, que contiene la información relativa a los tiempos medios de ejecución estructurada de forma que resulte más fácil comparar. Cuando la dimensión es 2, los tiempos son próximos para muestras pequeñas y medianas, diferenciándose más para muestras grandes. Sin embargo, en casi todos los casos la implementación 1 es más lenta que la 2. Al aumentar la dimensión, se

incrementan los tiempos de ejecución de ambas implementaciones, pero también las diferencias entre ellas. Para una muestra pequeña la implementación 1 invierte aproximadamente el doble de tiempo que la 2. Con muestras mayores la diferencia de tiempo se incrementa, para un tamaño de muestra grande, la implementación 1 tarda casi el triple de tiempo que la 2. Para una dimensión grande, los tiempos de ejecución son considerablemente mayores en ambas implementaciones. Destaca que para una dimensión grande y un tamaño de muestra grande se invierte la tendencia observada en el resto de combinaciones de tamaño de muestra y dimensión, en este caso la implementación 1 tarda menos que la 2.

Tabla 9: Tiempos de ejecución medios de la función `entropy` en ambas implementaciones para diferentes valores del parámetro d y n.

Implementación	d \ n	1000	30000	100000
1	2	0.0020 s	0.0774 s	0.4067 s
2	2	0.0017 s	0.0648 s	0.2515 s
1	10	0.0149 s	2.5013 s	16.0703 s
2	10	0.0071 s	0.8797 s	5.6956 s
1	25	0.1158 s	34.2677 s	654.9519 s
2	25	0.0475 s	30.6916 s	876.7703 s

Podemos concluir, por tanto, que la segunda implementación es más rápida que la primera para calcular la entropía, salvo para dimensión y tamaño de muestra grande. El cálculo que realizan ambas estimaciones es el mismo, utilizan el mismo estimador, varía únicamente la forma de implementarlo, en cuanto a tiempo de cómputo será preferible la implementación 2.

Si analizamos qué pasos emplean más tiempo en las implementaciones de la entropía nos sorprende descubrir que es en la misma función, el método `query` de `sklearn.neighbors._kd_tree.BinaryTree`. Sin embargo, aunque ambas implementaciones hacen uso de este método, la primera invierte más tiempo en ella que la segunda. Por ejemplo, en una ejecución de la implementación 1 para tamaños de muestra y dimensiones medianas que requirió $2{,}2$ s, esta función utilizó $2{,}125$ s., mientras que la implementación 2 invirtió $0{,}739$ s en esta función de una ejecución que en total consumió $0{,}776$ s, véase Tabla 10. Destaca que esta función no es solo la que más tiempo consume sino que la mayor parte del tiempo del cálculo de la entropía se emplea en realizar este cálculo, siendo los tiempos de las siguientes funciones mucho menores.

Tabla 10: Resultados `cProfiler`. Funciones que consumen más tiempo en el cálculo de la entropía para ambas implementaciones, caso $d = 10$, $n = 30000$, tiempos en segundos.

(a) Implementación 1, tiempo total de ejecución $2{,}202\,\mathrm{s}$.

ncalls	tottime	percall	filename:lineno(function)
1	2.125	2.125	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
1	0.072	0.072	_base.py:349(_fit)
10	0.002	0.000	built-in method numpy.array

(b) Implementación 2, tiempo total de ejecución $0{,}776\,\mathrm{s}$.

ncalls	tottime	percall	filename:lineno(function)
1	0.739	0.739	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
1	0.034	0.034	entropy_estimators.py:280(build_tree)
1	0.002	0.002	method 'random_sample' of 'numpy.random.mtrand.RandomState' objects

En la implementación 1 se obtienen los vecinos más cercanos utilizando la clase `sklearn.neighbors.NearestNeighbors`, invocando al método `kneighbors` [30]. Este método realiza una serie de comprobaciones antes de invocar al método `query` para obtener realmente los vecinos más cercanos y sus distancias. Además, la llamada a `query` se realiza usando métodos que proporcionan paralelismo de la biblioteca `joblib`, indicando que se utilizará paralelismo a nivel de hebra. En este caso, al crear el árbol con las opciones por defecto, se utiliza la métrica de Minkoswki con $p = 2$, es decir, la distancia euclídea. Además, el parámetro `leaf_size` se inicializa a 30.

En la implementación 2, en primer lugar, se crea un árbol, que será de tipo `sklearn.neighbors.BallTree` cuando la dimensión sea mayor que 20 y de tipo `sklearn.neighbors.KDTree` cuando la dimensión sea menor que 20; y a continuación, se calculan los vecinos más cercanos llamando directamente al método `query` de la clase correspondiente. La métrica utilizada es la de Chebyshev y `leaf_size = 40`.

El incremento de tiempo en las llamadas a `query` en la primera implementación podría provenir del coste de realizar las llamadas en paralelo y no aprovecharlo realmente, sabemos que el paralelismo a nivel de he-

bra en Python depende de si el cerrojo global del intérprete (GIL) está activado o desactivado. En la segunda implementación, se invoca al método directamente y no existe este coste.

La diferencia de tiempos también podría venir causada por el valor de `leaf_size`. La diferencia entre los valores de este parámetro en ambas implementaciones es 10, comparado con el número de muestras puede parecer pequeño, pero al influir en el cálculo de los vecinos más cercanos de todos los puntos podría afectar notablemente.

La métrica utilizada podría generar diferencias en los tiempos de ejecución. La métrica de Minkowski requiere calcular la raíz cuadrada de las diferencias al cuadrado, mientras que la de Chebyshev el máximo de las diferencias.

El tipo de árbol utilizado para buscar los vecinos más cercanos puede afectar a los tiempos de ejecución, de hecho, aunque la documentación recomiende usar `BallTree` para dimensiones mayores que 20, este parámetro no es exacto y es posible que en este caso llegue a resultar más costoso utilizarlo frente a `KDTree`.

Analizamos específicamente el caso en el que tamaño de muestra y dimensión son grandes, con el fin de averiguar qué ocurre exactamente para que la implementación 1 sea más rápida que la 2 solo en este caso. En la Tabla 11 observamos que en las dos implementaciones es el método `query` el que más tiempo emplea. Con la diferencia de que en la implementación 1 es el método `query` de `KDTree` y en la 2 el de `BallTree`. Sin embargo, sabemos que la implementación 2 utiliza el método `query` de la clase `BallTree` en todas las ejecuciones en las que $d = 25$ y en las demás sigue siendo más rápida que la implementación 1. Sospechamos que el incremento en el número de muestras provoca que esta implementación sea más lenta únicamente en este caso.

Pasamos a estudiar qué implementación es más rápida en el cálculo de la información mutua. Recordamos que en este caso las implementaciones no realizan el mismo cálculo. La primera implementación halla la información mutua a través de las entropías de las variables marginales, mientras que la segunda implementación lo hace a través de $I^{(2)}$.

En la Tabla 12 se recogen los tiempos de ejecución de la implementación 1 para los valores de n y d considerados. Al igual que para la entropía, los tiempos en todos los casos aumentan al considerar muestras más grandes y de mayor dimensión. Además, notamos que los tiempos de ejecución son superiores a los necesitados para calcular la entropía, esto no es de extrañar, en uno de los casos se calcula la entropía de varias variables y luego se opera con ellas para obtener la información mutua.

Tabla 11: Resultados `cProfiler`. Funciones que consumen más tiempo en el cálculo de la entropía para ambas implementaciones, caso $d = 25$, $n = 100000$, tiempos en segundos.

(a) Implementación 1, tiempo total de ejecución 643.479 s.

ncalls	tottime	percall	filename:lineno(function)
1	643.026	643.026	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
1	0.440	0.440	_base.py:349(_fit)
10	0.007	0.001	built-in method numpy.array

(b) Implementación 2, tiempo total de ejecución 867,12 s.

ncalls	tottime	percall	filename:lineno(function)
1	866.847	866.847	method 'query' of 'sklearn.neighbors._ball_tree.BinaryTree' objects
1	0.249	0.249	entropy_estimators.py:280(build_tree)
1	0.016	0.016	method 'random_sample' of 'numpy.random.mtrand.RandomState' objects

Las desviaciones típicas son pequeñas en relación a los valores medios.

Tabla 12: Tiempos de ejecución del cálculo de la información mutua para diferentes valores de n y d, implementación 1.

(a) Tiempos medios de ejecución.

d \ n	1000	30000	100000
2	0.0067 s	0.3754 s	1.3876 s
10	0.0681 s	24.1955 s	344.3863 s
25	0.2173 s	173.4488 s	2811.0427 s

(b) Desviaciones típicas de los tiempos de ejecución.

d \ n	1000	30000	100000
2	0.0008 s	0.0089 s	0.0833 s
10	0.0013 s	1.5809 s	10.2065 s
25	0.0013 s	4.2758 s	87.2810 s

De la misma forma, en la Tabla 13 encontramos los tiempos de ejecución de la información mutua en el caso de la implementación 2. Atendiendo a los tiempos medios de ejecución, vemos que crecen a medida que aumenta el tamaño de muestra y dimensión, siendo más altos que los tiempos de cálculo de la entropía. Las desviaciones típicas son también, en este caso, pequeñas en comparación a los tiempos que comparan, por tanto, concluimos que 5 mediciones han sido suficientes para conocer los tiempos de ejecución de las diferentes funciones con precisión.

Tabla 13: Tiempos de ejecución del cálculo de la información mutua para diferentes valores de n y d, implementación 2.

(a) Tiempos medios de ejecución.

d \ n	1000	30000	100000
2	0.0066 s	0.6363 s	2.7960 s
10	0.0485 s	33.6843 s	798.8034 s
25	0.1271 s	154.2582 s	3036.5627 s

(b) Desviaciones típicas de los tiempos de ejecución.

d \ n	1000	30000	100000
2	0.0012 s	0.0256 s	0.1403 s
10	0.0005 s	0.6881 s	15.5813 s
25	0.0010 s	0.4396 s	22.6054 s

Compararemos los tiempos de ejecución de ambas implementaciones utilizando la Tabla 14. Para un tamaño de muestra pequeña y dimensión pequeña, los tiempos son próximos, pero a medida que van aumentando, tanto dimensión como tamaño de muestra, aumenta también la diferencia entre los tiempos de ejecución de las dos implementaciones. Si nos fijamos en el tamaño de muestra pequeño, la implementación 1 resulta más lenta que la 2 para todas las dimensiones consideradas, ocurriendo lo contrario cuando nos fijamos en un tamaño de muestra grande, en los que la implementación 1 resulta más rápida que la 2 para todas las dimensiones. Para un tamaño de muestra mediano, el comportamiento varía según la dimensión, para $d = 2, d = 10$ la implementación 1 resulta más rápida que la 2, mientras que para $d = 25$ ocurre lo contrario.

En la primera implementación, vuelve a ser el método `query` de `KDTree` el que requiere más tiempo de ejecución, aunque destacamos que esta vez el número de veces que se llama a esta función es 3, ya que se calculan tres entropías. Dos de estas llamadas serán sobre puntos de dimensión d, cuando calculemos $h(X)$ y $h(Y)$, mientras que la restante es

Tabla 14: Tiempos de ejecución del cálculo de la información mutua en ambas implementaciones para diferentes valores del parámetro d y n.

Implementación	d \ n	1000	30000	100000
1	2	0.0067 s	0.3754 s	1.3876 s
2	2	0.0066 s	0.6363 s	2.7960 s
1	10	0.0681 s	24.1955 s	344.3863 s
2	10	0.0485 s	33.6843 s	798.8034 s
1	25	0.2173 s	173.4488 s	2811.0427 s
2	25	0.1271 s	154.2582 s	3036.5627 s

sobre puntos de dimensión $2d$, en el cálculo de $h(X, Y)$.

En la segunda implementación la función que más tiempo consume es también el método `query`, de la clase `KDTree` o `BallTree` según la dimensión. Notamos que en esta implementación, antes de llamar al método `query`, se unen los puntos de las variables para las que calculamos la información mutua, por tanto, para $d = 10$ se consideraría un árbol con puntos de dimensión 20. Además, la segunda función que más tiempo consume se aproxima, en tiempo total, a `query`; es la función `query_radius`, utilizada para calcular el valor de la media de $\psi(n)$. Esta función se llama dos veces, una para contar los puntos de la primera variable que están a menor distancia que el k-ésimo vecino y otra para realizar el mismo cálculo respecto de los puntos de la segunda variable. Cada una de estas llamadas individuales a `query_radius`, donde consideramos puntos de dimensión d, será más rápida que la llamada a `query`, donde se consideran puntos de dimensión $2d$.

En resumen, las funciones que más tiempo consumen son: en la implementación 1 las 2 llamadas a `query` para puntos de dimensión d y la llamada a `query` para puntos de dimensión $2d$, en la implementación 2 la llamada a `query` sobre puntos de dimensión $2d$ y las 2 llamadas a `query_radius` para puntos de dimensión d.

Comenzamos analizando qué ocurre para un número de muestras pequeño, $n = 1000$, en la Tabla 17 los resultados del *profiling* para muestras de dimensión 10. Por un lado, al estudiar el caso de la entropía vimos que los tiempos del cálculo de `query` en la implementación 1 eran más lentos que en la 2. Por otro lado, es posible que para muestras pequeñas el cálculo de `query_radius` sea más rápido que el de `query`. Estos motivos nos permiten justificar que la implementación 2 haya resultado más rápida para este tamaño de muestra.

Tabla 15: Resultados `cProfiler`. Funciones que consumen más tiempo en el cálculo de la información mutua para ambas implementaciones, caso $d = 10$, $n = 1000$, tiempos en segundos.

(a) Implementación 1, tiempo total de ejecución $0{,}069$ s.

ncalls	tottime	percall	filename:lineno(function)
3	0.064	0.021	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.003	0.001	_base.py:349(_fit)
12	0.000	0.000	method 'reduce' of 'numpy.ufunc' objects

(b) Implementación 2, tiempo total de ejecución $0{,}049$ s.

ncalls	tottime	percall	filename:lineno(function)
1	0.025	0.025	method 'query' of 'sklearn.neighbors._ball_tree.BinaryTree' objects
2	0.022	0.011	method 'query_radius' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.002	0.001	entropy_estimators.py:280(build_tree)

Para tamaños de muestras grandes, $n = 100000$, se observa el comportamiento contrario, la implementación 1 es más rápida que la 2. Si observamos en la Tabla 16 los tiempos de las funciones que emplearon más tiempo, vemos claramente el motivo. El tiempo de ejecución de `query_radius` en la implementación 2 supera a los tiempos de ejecución de `query` en la implementación 1. De hecho, el cálculo de 2 `query_radius` supera en tiempo de ejecución al cálculo de 3 `query`.

Para tamaños de muestras medianos, $n = 30000$, tendremos que distinguir según la dimensión. Para dimensiones pequeñas y medianas la implementación 1 resulta más rápida debido a que los tiempos de cómputo de `query_radius` superan a los de `query`, véase Tabla 17. Sin embargo, para dimensiones grandes, la implementación 1 resulta más lenta, esto se debe a que al cambiar el tipo de árbol con el que se realiza `query_radius` de `KDTree` a `BallTree` disminuye su tiempo de cálculo, siendo inferior que los `query` que realiza la implementación 1, como vemos en la Tabla 18.

Tabla 16: Resultados `cProfiler`. Funciones que consumen más tiempo en el cálculo de la información mutua para ambas implementaciones, caso $d = 2$, $n = 100000$, tiempos en segundos.

(a) Implementación 1, tiempo total de ejecución $1{,}348$ s.

ncalls	tottime	percall	filename:lineno(function)
3	1.125	0.375	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.205	0.068	_base.py:349(_fit)
36/22	0.006	0.000	built-in method numpy.core._multiarray_umath.implement_array_function

(b) Implementación 2, tiempo total de ejecución $02{,}650$ s.

ncalls	tottime	percall	filename:lineno(function)
2	2.010	1.005	method 'query_radius' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
1	0.424	0.424	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.204	0.068	entropy_estimators.py:280(build_tree)

Tabla 17: Resultados `cProfiler`. Funciones que consumen más tiempo en el cálculo de la información mutua para ambas implementaciones, caso $d = 2$, $n = 30000$, tiempos en segundos.

(a) Implementación 1, tiempo total de ejecución $0{,}37$ s.

ncalls	tottime	percall	filename:lineno(function)
3	0.325	0.108	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.039	0.013	_base.py:349(_fit)
36/22	0.002	0.000	built-in method numpy.core._multiarray_umath.implement_array_function

(b) Implementación 2, tiempo total de ejecución $0{,}635$ s.

ncalls	tottime	percall	filename:lineno(function)
2	0.455	0.228	method 'query_radius' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
1	0.134	0.134	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.043	0.014	entropy_estimators.py:280(build_tree)

Tabla 18: Resultados `cProfiler`. Funciones que consumen más tiempo en el cálculo de la información mutua para ambas implementaciones, caso $d = 25$, $n = 30000$, tiempos en segundos.

(a) Implementación 1, tiempo total de ejecución $172{,}812$ s.

ncalls	tottime	percall	filename:lineno(function)
3	0.325	0.108	method 'query' of 'sklearn.neighbors._kd_tree.BinaryTree' objects
3	0.039	0.013	_base.py:349(_fit)
36/22	0.002	0.000	built-in method numpy.core._multiarray_umath.implement_array_function

(b) Implementación 2, tiempo total de ejecución $154{,}347$ s.

ncalls	tottime	percall	filename:lineno(function)
1	95.822	95.822	method 'query' of 'sklearn.neighbors._ball_tree.BinaryTree' objects
2	58.309	29.154	method 'query_radius' of 'sklearn.neighbors._ball_tree.BinaryTree' objects
3	0.197	0.066	entropy_estimators.py:280(build_tree)

5

CONCLUSIONES

Este trabajo comienza con una revisión de los conceptos necesarios para abordar sus objetivos, se estudian incialmente los conceptos de entropía e información mutua, tanto en el caso discreto como en el caso continuo. Posteriormente, se han estudiado sus propiedades, conceptos relacionados y relaciones entre ellas. A continuación, se analiza teóricamente algunas propuestas de estimadores de la entropía, basadas en los k vecinos y también del tipo de Kozachenko - Leonenko, y de la información mutua, basadas en particiones y en el estimador de la entropía del tipo de Kozachenko - Leonenko.

El análisis teórico y experimental de los estimadores nos permite obtener algunas conclusiones. El análisis de su comportamiento en el caso de la distribución normal permite afirmar que los errores en ambos estimadores aumentan a medida que crece la dimensión de los datos. En el caso de la entropía es ligeramente inferior el error obtenido por la implementación 1 [12], pero en el caso de la información mutua estos errores resultan similares en ambas implementaciones.

La realización del *T-Test* sobre las implementaciones de la entropía permite concluir que el comportamiento de los estimadores es similar para dimensiones pequeñas, pero a medida que la dimensión crece, las diferencias en media entre los estimadores son significativas. En el caso de la entropía, las pruebas realizadas comparando con variables aleatorias generadas con una distribución normal, permitirían concluir que el estimador 1 [12] presenta errores menores y, por tanto, resultaría preferible al estimador 2[35]. En el caso de la información mutua, sin embargo, no concluimos nada en este sentido, ya que los errores obtenidos por ambos estimadores fueron similares.

Atendiendo al rendimiento, la implementación más rápida para realizar el cálculo de la entropía es la dada en la implementación 2 [35], excepto cuando el número de observaciones y la dimensión sean grandes, en cuyo caso resulta más rápida la implementación 1 [12]. En el estudio de la información mutua, si tenemos que elegir la estimación más rápida, dependerá del tamaño de la muestra y de la dimensión. Para muestras de tamaño pequeño resulta más veloz la implementación 2 [35], al igual que para muestras de tamaño medio y dimensión grande. En el resto de casos, muestras de tamaño grande y muestras de tamaño mediano con dimensiones pequeñas y medianas es más rápida la implementación 1 [12]. Notamos que para dimensiones grandes y tamaños de muestra grande

los tiempos de ejecución se incrementan notablemente, tiempos cercanos a unos pocos segundos de ejecución cuando la dimensión es pequeña se transforman en tiempos entre 45 minutos y una hora para dimensiones grandes y tamaños de muestra grandes. Se hace necesario, por tanto, el uso de paralelismo a nivel de ejecución si se desea obtener resultados en tiempo aceptables.

6 ESTIMACIÓN DEL COSTE Y PLANIFICACIÓN

6.1 ESTIMACIÓN DEL COSTE

Se realiza un presupuesto del coste del trabajo, en el que se incluyen las horas dedicadas a cada concepto y se realiza una estimación a precio 35€/hora, véase Tabla 19.

Para la realización del presupuesto se ha dividido el trabajo en tres módulos: módulo teórico, módulo práctico y módulo general.

En el módulo teórico se recogen las tareas asociadas al estudio y análisis de los contenidos de carácter teórico. Notamos que aquí se incluye el estudio de las diferentes nociones a partir de varias fuentes para su posterior síntesis.

En el módulo experimental se recogen las diferentes tareas asociadas al análisis de las implementaciones. No sólo se realiza un análisis de las mismas sino que es necesario la implementación y validación de código para llevar a cabo el análisis desde diferentes perspectivas. Además, se tiene en cuenta el equipo utilizado y el tiempo dedicado a la instalación del sistema operativo y las bibliotecas necesarias para llevar a cabo el análisis.

En el módulo general se encuentran las tareas relativas a la redacción de este documento, además de las correcciones que ha requerido, así como las reuniones con los tutores.

Tabla 19: Estimación del coste del trabajo.

Concepto	Cantidad (horas)	Coste unidad (€/h)	Coste total (€)
Módulo teoría			
Análisis/estudio conceptos previos	15	35	525
Análisis/estudio caso discreto	25	35	875
Análisis/estudio caso continuo	25	35	875
Análisis/estudio estimadores	60	35	2100
Módulo experimental			
Equipo de trabajo: Dell portátil Vostro 5590		1085	1085
Instalación sistema operativo, bibliotecas, entorno de desarrollo	10	35	350
Análisis implementaciones estimadores	20	35	700
Análisis/desarrollo código análisis del error	30	35	1050
Análisis/desarrollo/validación clase para generación de muestras	5	35	175
Análisis del *T-Test*, desarrollo del *T-Test*, validación del *T-Test*	25	35	875
Análisis *profilers*, implementación/integración/validación código para *profiling*, ejecución	32	35	1120
Análisis de resultados obtenidos	20	35	700
Módulo general			
Redacción memoria	50	35	1750
Corrección memoria	15	35	525
Reuniones con los tutores	30	35	1050
Total	362		13755

IVA no incluido.

6.2 PLANIFICACIÓN

Se incluye en la Figura 4 la planificación optimista del desarrollo del trabajo. En ella se han dividido las tareas también en módulo general (de color rosa), módulo teórico (de color azul) y módulo práctico (de color verde). Notamos que las tareas del módulo general en realidad se encontrarán presentes a lo largo de todo el desarrollo del proyecto, ya que la redacción de la memoria, corrección de la misma y reuniones con los tutores para comentar estas correcciones y/o posibles dudas se realizan a medida que se avanza en la realización de las tareas específicas. En la planificación se ha tenido en cuenta que el primer cuatrimestre habría más carga docente que el segundo cuatrimestre (y con ello menos tiempo para la realización de este trabajo), así como el periodo de exámenes en el que prácticamente no se avanzaría.

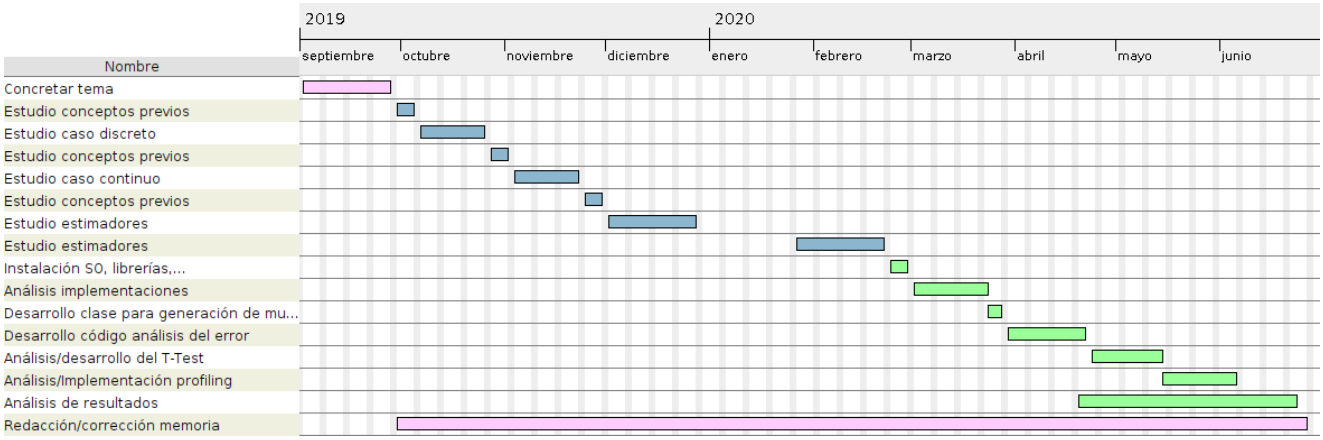


Figura 4: Planificación optimista del desarrollo del trabajo.

Sin embargo, no todo ha salido según lo previsto, por ejemplo, el tiempo en estudiar y comprender los estimadores fue superior al estimado y requirió la consulta de bibliografía adicional. Además, el estudio de los estimadores fue intercalado con algunas tareas del módulo práctico, como vemos en la Figura 5, donde encontramos una planificación más realista del trabajo.

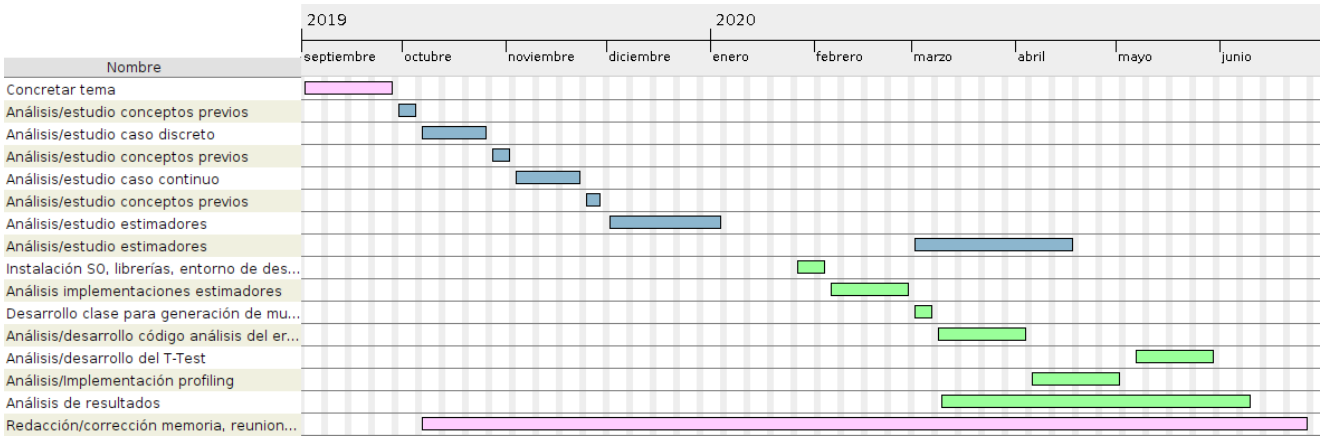


Figura 5: Planificación realista del desarrollo del trabajo.

BIBLIOGRAFÍA

[1] *1.6. Nearest Neighbors — scikit-learn 0.23.1 documentation*. URL: https://scikit-learn.org/stable/modules/neighbors.html#neighbors (Último acceso 18-06-2020).

[2] *About*. URL: https://www.isi.edu/people/gregv/about (Último acceso 26-06-2020).

[3] *About me: machine learning researcher // Gaël Varoquaux: computer / data / brain science*. URL: http://gael-varoquaux.info/about.html (Último acceso 26-06-2020).

[4] M. Aghagolzadeh y col. "A Hierarchical Clustering Based on Mutual Information Maximization". En: *2007 IEEE International Conference on Image Processing*. Vol. 1. 2007, págs. 277-280.

[5] Sofía Almeida. *Repositorio para el trabajo fin de grado*. es. URL: https://github.com/SofiaAlmeida/TFG (Último acceso 28-05-2020).

[6] *BSTJ 7: 3. July 1928: Transmission of Information. (Hartley, R.V.L.)* eng. Jul. de 1928. URL: http://archive.org/details/bstj7-3-535 (Último acceso 25-06-2020).

[7] Yen-Chi Chen. "Lecture 7: Density Estimation: k-Nearest Neighbor and Basis Approach". STAT 425: Introduction to Nonparametric Statistics. 2018.

[8] T. M. Cover y Joy A. Thomas. *Elements of information theory*. 2nd ed. OCLC: ocm59879802. Hoboken, N.J: Wiley-Interscience, 2006. ISBN: 9780471241959.

[9] *Embarrassingly parallel for loops — joblib 0.14.1.devo documentation*. URL: https://joblib.readthedocs.io/en/latest/parallel.html#parallel (Último acceso 15-06-2020).

[10] Francois Fleuret. "Fast Binary Feature Selection with Conditional Mutual Information". En: *Journal of Machine Learning Research* 5 (nov. de 2004), págs. 1531-1555.

[11] Eva Freyhult, Vincent Moulton y Paul Gardner. "Predicting RNA Structure Using Mutual Information:" en. En: *Applied Bioinformatics* 4.1 (2005), págs. 53-59. ISSN: 1175-5636. DOI: 10.2165/00822942-200504010-00006. URL: http://link.springer.com/10.2165/00822942-200504010-00006 (Último acceso 25-06-2020).

[12] GaelVaroquaux. *Estimating entropy and mutual information with scikit-learn*. en. URL: https://gist.github.com/GaelVaroquaux/ead9898bd3c973c40429 (Último acceso 01-05-2020).

[13] *Git*. URL: https://git-scm.com/ (Último acceso 23-06-2020).

[14] *Glossary — Python 3.8.3 documentation*. URL: https://docs.python.org/3/glossary.html#term-global-interpreter-lock (Último acceso 15-06-2020).

[15] M. N. Goria y col. "A new class of random vector entropy estimators and its applications in testing statistical hypotheses". en. En: *Journal of Nonparametric Statistics* 17.3 (abr. de 2005), págs. 277-297. ISSN: 1048-5252, 1029-0311. DOI: 10.1080/10485250400026815. URL: https://www.tandfonline.com/doi/full/10.1080/10485250400026815 (Último acceso 25-06-2020).

[16] Robert M. Gray. *Entropy and Information Theory*. en. Boston, MA: Springer US, 2011. ISBN: 9781441979698 9781441979704. DOI: 10.1007/978-1-4419-7970-4. URL: http://link.springer.com/10.1007/978-1-4419-7970-4 (Último acceso 01-05-2020).

[17] K Hlavackovaschindler y col. "Causality detection based on information-theoretic approaches in time series analysis". en. En: *Physics Reports* 441.1 (mar. de 2007), págs. 1-46. ISSN: 03701573. DOI: 10.1016/j.physrep.2006.12.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S0370157307000403 (Último acceso 25-06-2020).

[18] *Initialization, Finalization, and Threads — Python 3.8.3 documentation*. URL: https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock (Último acceso 15-06-2020).

[19] Alexander Kraskov, Harald Stoegbauer y Peter Grassberger. "Estimating Mutual Information". En: *Physical Review E* 69.6 (jun. de 2004). arXiv: cond-mat/0305641, pág. 066138. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.69.066138. URL: http://arxiv.org/abs/cond-mat/0305641 (Último acceso 16-06-2020).

[20] N. N. Leonenko L. F. Kozachenko. "Sample Estimate of the Entropy of a Random Vector". En: *Problems of Information Transmission* 23.2 (1987), págs. 95-101.

[21] *Matplotlib: Python plotting — Matplotlib 3.2cumentation*. URL: https://matplotlib.org/ (Último acceso 23-06-2020).

[22] Álvaro Montenegro. "Information and entropy in economics". En: *Revista de Economía Institucional* 13.25 (jul. de 2011), págs. 199-221. ISSN: 0124-5996. URL: http://www.scielo.org.co/scielo.php?script=sci_abstract&pid=S0124-59962011000200009&lng=en&nrm=iso&tlng=es (Último acceso 25-06-2020).

[23] *multiprocessing — Process-based parallelism — Python 3.8.3 documentation*. URL: https://docs.python.org/3/library/multiprocessing.html (Último acceso 15-06-2020).

[24] *NumPy*. URL: https://numpy.org/ (Último acceso 23-06-2020).

[25] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (Último acceso 23-06-2020).

[26] Liam Paninski. "Estimation of Entropy and Mutual Information". en. En: *Neural Computation* 15.6 (jun. de 2003), págs. 1191-1253. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976603321780272. URL: http://www.mitpressjournals.org/doi/10.1162/089976603321780272 (Último acceso 17-06-2020).

[27] *ParallelProcessing - Python Wiki*. URL: https://wiki.python.org/moin/ParallelProcessing (Último acceso 15-06-2020).

[28] Fernando Pérez-Cruz. "Estimation of Information Theoretic Measures for Continuous Random Variables." En: ene. de 2008, págs. 1257-1264.

[29] *random — Generate pseudo-random numbers — Python 3.8.3 documentation*. URL: https://docs.python.org/3/library/random.html (Último acceso 23-06-2020).

[30] *scikit-learn/scikit-learn, kneighbors method*. en. URL: https://github.com/scikit-learn/scikit-learn/blob/fd237278e/sklearn/neighbors/_base.py#L533 (Último acceso 18-06-2020).

[31] *scipy.stats.ttest_rel — SciPy v1.5.0 Reference Guide*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html (Último acceso 22-06-2020).

[32] C. E. Shannon. "A Mathematical Theory of Communication". en. En: *Bell System Technical Journal* 27.3 (jul. de 1948), págs. 379-423. ISSN: 00058580. DOI: 10.1002/j.1538-7305.1948.tb01338.x. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6773024 (Último acceso 25-06-2020).

[33] Shashank Singh y Barnabás Póczos. "Finite-Sample Analysis of Fixed-k Nearest Neighbor Density Functional Estimators". En: *arXiv:1606.01554 [cs, math, stat]* (jun. de 2016). arXiv: 1606.01554. URL: http://arxiv.org/abs/1606.01554 (Último acceso 22-06-2020).

[34] *sklearn.neighbors.NearestNeighbors — scikit-learn 0.23.1 documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors.kneighbors (Último acceso 18-06-2020).

[35] Greg Ver Steeg. *gregversteeg/NPEET*. original-date: 2014-10-10T19:57:02Z. Abr. de 2020. URL: https://github.com/gregversteeg/NPEET (Último acceso 01-05-2020).

[36] *The Python Profilers — Python 3.8.3 documentation*. URL: https://docs.python.org/3/library/profile.html#pstats.Stats (Último acceso 15-06-2020).

[37] *threading — Thread-based parallelism — Python 3.8.3 documentation*. URL: https://docs.python.org/3/library/threading.html?highlight=parallelism (Último acceso 15-06-2020).

[38] *Welcome to Python.org*. en. URL: https://www.python.org/ (Último acceso 23-06-2020).