

## **Trabajo de Laboratorio N°3:**

### ***Algoritmos de Clasificación -***

### ***KNN y Clasificador Bayesiano***

- **Materia:** Inteligencia Artificial.
- **Carrera:** Ingeniería en Informática.
- **Docente:** Ing. Federico Gabriel D'Angiolo.
- **Estudiante:** Calonge, Federico.

# **Índice.-**

|   |                  |
|---|------------------|
| <b>1- Objetivo.</b>                             | <b>(Pág. 3)</b>  |
| <b>2- Introducción: Conceptos previos.</b>      | <b>(Pág. 4)</b>  |
| 2.1-Clasificación.                              | <b>(Pág.4)</b>   |
| 2.2-KNN.  | <b>(Pág.5)</b>   |
| 2.3-Clasificador Bayesiano – Naive Bayes.       | <b>(Pág.9)</b>   |
| <b>3- Desarrollo.</b>                           | <b>(Pág.16)</b>  |
| 3.1-KNN:  |                  |
| Implementación Ejemplo N°1                      | <b>(Pág.16)</b>  |
| 3.1.1-Data set.                                 | <b>(Pág.16)</b>  |
| 3.1.2-¿Qué queremos predecir?                   | <b>(Pág.17)</b>  |
| 3.1.3-Implementación del modelo de KNN          | <b>(Pág.17)</b>  |
| 3.1.4-Resultados.                               | <b>(Pág.18)</b>  |
| 3.1.5-Diferencias entre KNN y K-means.          | <b>(Pág.19)</b>  |
| 3.1.6-Casos de uso de KNN y K-means.            | <b>(Pág.20)</b>  |
| 3.2-Clasificador Bayesiano – Naive Bayes:       |                  |
| Implementación Ejemplo N°2                      | <b>(Pág. 21)</b> |
| 3.2.1-Data set.                                 | <b>(Pág.21)</b>  |
| 3.2.2-¿Qué queremos predecir?                   | <b>(Pág.21)</b>  |
| 3.2.3-Implementación modelo Bayes.              | <b>(Pág.21)</b>  |
| 3.2.4-Resultados.                               | <b>(Pág.22)</b>  |
| 3.3-Cómo evaluar la Eficacia de cada algoritmo. | <b>(Pág.23)</b>  |
| <b>4-Conclusiones.</b>                          | <b>(Pág.24)</b>  |
| <b>5-Mejoras a desarrollar.</b>                 | <b>(Pág.25)</b>  |
| <b>6-Bibliografía.</b>                          | <b>(Pág.26)</b>  |

## 1- Objetivo.

Este Trabajo de Laboratorio consistirá en estudiar los distintos algoritmos de Clasificación, obteniendo los resultados que se adquieren de cada uno de ellos.

Para esto analizaremos 2 Data Sets y demostraremos el funcionamiento de distintas técnicas de Clasificación mediante ejemplos realizados en Python con el ambiente Anaconda. Los algoritmos de Clasificación que analizaremos serán:

- **KNN.**
- **Clasificador Bayesiano.**

Explicaremos qué significa usar cada una de estas técnicas, y a partir de estas, obtendremos modelos y visualizaciones de los datos que nos permitan predecir el comportamiento de distintas variables **discretas**.

## 2- Introducción: Conceptos previos.

En esta Sección describiremos los temas teóricos y matemáticos para tratarlos a lo largo del desarrollo del Informe y llevarlos a cabo mediante los Algoritmos de Python en el ambiente de Anaconda.

### 2.1-Clasificación.

**Clasificación** es una técnica que utilizamos para **predecir respuestas / valores discretos** (a diferencia de las técnicas de **Regresión**, donde tratamos de **predecir respuestas continuas**). Los algoritmos de clasificación están dentro de los **algoritmos de aprendizaje supervisado**, donde los modelos de machine learning que construimos se basan en datos de entrenamiento **etiquetados**; de esta manera le damos parámetros a nuestro algoritmo y a nuestros datos, y, basado en estos parámetros, el algoritmo aprenderá a clasificar los datos.

Una respuesta discreta que puede predecir un algoritmo de clasificación es, por ejemplo, si un mail es Spam o no o si un objeto es color blanco, negro o azul, etc. Es decir, esta técnica se utiliza, como mencionamos anteriormente, cuando los datos pueden ser etiquetados o categorizados en grupos o clases.

Como en los casos de Regresión que vimos anteriormente, acá también tendremos variables “features” o variables predictoras “Xs” (que corresponden a características de nuestros datos) y una variable “target” o variable a predecir “Y” (es nuestra variable objetivo, la que queremos predecir). De esta manera las variables “features” pueden ser continuas o discretas pero la variable “target” deberá ser si o si una variable discreta.

Hay muchas técnicas de Clasificación, entre ellas las más utilizadas son:

1. **Vecinos Más Cercanos (KNN).**
2. **Naive Bayes – Clasificador Bayesiano.**
3. Árboles de decisión/clasificación,
4. Máquinas de Vectores de Soporte.
5. Regresión logística.
6. Linear discriminant analysis (LDA) o normal discriminant analysis (NDA).

**Nosotros nos vamos a enfocar en este Trabajo de Laboratorio en las dos primeras técnicas.**

## 2.2-KNN.

Como mencionamos anteriormente, **KNN (K Nearest Neighbors o K Vecinos más Cercanos)** es una técnica de aprendizaje supervisado y de clasificación. Es uno de los modelos más simples de implementar para tareas de clasificación. Permite clasificar datos **basándose en la similitud con otros datos de un set de prueba**: lo que hace KNN es simplemente **calcular las distancias de un nuevo punto de datos a todos los demás puntos de datos de entrenamiento**. Es un algoritmo de “lazy learning”, ya que no tiene una fase especializada de entrenamiento. KNN es un algoritmo NO paramétrico, lo que significa que NO asumen nada acerca de los datos.

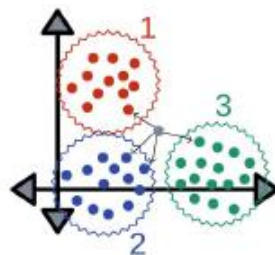
El algoritmo KNN tiene 2 principales desventajas:

- Tiene un muy alto costo de predicción trabajando con **datasets muy grandes**. Esto es porque en grandes datasets calcular las distancias entre un nuevo punto y los miles de puntos existentes resulta muy dificultoso para el algoritmo.
- Tampoco funciona muy bien para datos que tengan **muchas dimensiones**, ya que a mayor número de dimensiones, mayor será la dificultad del algoritmo de calcular la distancia para cada dimensión.

Por estas razones **KNN tiende a funcionar mejor en datasets pequeños y con pocas features (columnas)**.

### Ejemplo de funcionamiento de KNN:

Supongamos que tenemos 3 grupos ya armados/etiquetados. La idea es tirar un nuevo elemento y que nuestro algoritmo KNN lo clasifique dentro de esos 3 grupos. Como ejemplo supongamos que nuestros 3 grupos son simples esferas de colores (rojo=1, azul=2 y verde=3). Ver **imagen 1**. De esta manera nosotros tiraremos una nueva esfera (representada en la imagen por el color gris, pero realmente NO sabemos su color) y KNN nos lo clasificará en verde, roja o azul en base a sus características (si se asemejan a las características de las esferas rojas, azules o verdes).



**Imagen 1**

Al implementar KNN se deben seguir los siguientes 3 pasos:

1. Se calcula la **distancia euclidiana**  $d(p, q)$  entre el **ítem a clasificar** (en nuestro ejemplo la esfera gris) **y el resto de puntos** (en nuestro ejemplo las demás esferas rojas, azules y verdes). Para calcular  $d(p, q)$  utilizamos la **Fórmula 1**.

$$\begin{aligned} d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}. \end{aligned}$$

**Fórmula 1**

Donde:

- $p$  = es nuestra muestra; en nuestro ejemplo es nuestra esfera gris.  $p$  está formado por varios coeficientes  $n$ : por ejemplo, en un caso muy simple podríamos tener que  $p=(1,2)$ . Esto representaría que estaríamos ante un caso de únicamente 2 dimensiones  $X$  e  $Y$ ; y nuestros cálculos serían mucho más sencillos.
- $q$  = son los demás puntos (esferas rojas, azules y verdes). Al igual que  $p$ ,  $q$  está formado por varios coeficientes  $n$ .
- $d(q, p)$  = distancia euclidiana entre el único punto  $p$  y cada punto  $q$ .

Esta fórmula para sacar la distancia euclidiana la hacemos N veces dependiendo de nuestros N valores del DS, ya que necesitamos cada una de las distancias de nuestra muestra gris p a cada punto de nuestro dataset q (formado por las esferas azules, verdes y rojas).

Por ejemplo, si tenemos 30 puntos en nuestro dataset (donde 10 son rojos, 10 azules y 10 verdes) entonces la **Fórmula 1** la debiéramos de hacer 30 veces comparando esas distancias q con nuestra distancia p que siempre va a ser la misma.

Cuando tenemos más de 2 features (X e Y) en nuestro modelo, ahí se empieza a complicar y no podemos graficar visualmente la relación entre nuestras variables.

2. Luego de tener estas distancias se seleccionan los **K** elementos más cercanos (osea aquellos con **menor distancia euclidiana**). Por ejemplo, si estamos parados en la esfera gris vemos que tenemos 4 esferas de los otros colores más cercanas (las que están marcadas con las 4 flechas grises en la **Imagen 1**).

Entonces, ahí seleccionamos las K=4 más cercanas. Para darnos cuenta que eran las más cercanas previamente tuvimos que calcular las distancias. Podíamos haber elegido 10, 20, 15 de K... pero elegimos 4 → **Esta elección de K es un punto importante para el algoritmo.**

3. Y por último, se hace una **votación de mayoría** entre esos **K** puntos más cercanos (en nuestro caso eran 4). Entonces de estos 4.... los de la clase que **predomine** (si son rojo, verde o azul) decidirá su **clasificación final** → en el ejemplo que estamos considerando de la **Imagen 1**, nuestra esfera gris será clasificada / predicha como **azul**, ya que tenemos 2 azules más cercanos (>1 verde y >1 rojo). En caso de existir un empate, la clase se elige aleatoriamente.

La pelota gris que usamos como ejemplo podría ser cualquier elemento que quisiéramos clasificar en base a nuestros datos; en los ejemplos de implementación que haremos en este Trabajo, estos datos serán animales de un zoológico. De esta manera, “tirando” el animal a nuestro algoritmo KNN ya entrenado, este nos dirá/predecirá a qué clase/tipo pertenece el mismo.

Anteriormente, con Regresión de Ridge resolvíamos el problema de Regresión Lineal ajustando el parámetro “alfa”; ahora, en KNN nuestra incógnita es el valor de K a elegir. Por esto, el proceso de **elección del número K** es vital para obtener un **buen rendimiento del algoritmo** → el **K óptimo**. No hay una manera exacta para determinar el mejor valor de K, por lo cual es necesario **iterar con algunos valores** antes de seleccionar uno en particular.

No hay una fórmula o manera exacta para obtener y determinar el mejor valor de K. Lo que hay que hacer es **iterar y evaluar el score del algoritmo** con algunos valores y quedarnos con un RANGO de valores de K que nos asegure un gran porcentaje de eficiencia del algoritmo (>90% sería un buen porcentaje). Ya que, van a haber rangos de valores de K donde el algoritmo trabajará mal y otros rangos de valores K donde el algoritmo trabajará bien. Esto lo implementamos en nuestro algoritmo utilizado en **3.1-KNN**, donde **llegamos a determinar un K óptimo** con el que entrenaremos nuestro modelo y realizaremos predicciones.



### 2.3-Clasificador Bayesiano – Naive Bayes.

Al igual que KNN, el **Clasificador Bayesiano**, o también llamado **Naive Bayes**, es una técnica de aprendizaje supervisado y de clasificación. Permite construir clasificadores **utilizando el teorema de Bayes**. Naive Bayes es un poderoso algoritmo. El término “naive” (ingenuo) viene por el hecho de que este algoritmo toma unos pequeños atajos, que revisaremos más adelante, para calcular las probabilidades de clasificación. Es muy flexible y se puede usar en diferentes tipos de data sets fácilmente y sin restricciones, ya que solo se deben usar las características numéricas. Por lo tanto, **es el más apropiado para tareas como Clasificación de texto (NPL)**. Naive Bayes trabaja en un modelo probabilístico que se basa en la interpretación de las estadísticas bayesianas.

#### Teorema de Bayes:

El enfoque de las estadísticas de aprendizaje se divide en dos partes/significados: el **enfoque “frecuentista”** (frequentist) y el **enfoque bayesiano**.

La mayoría de nosotros estamos familiarizados con el **enfoque frecuentista** cuando nos encontramos por primera vez con estadísticas. Según este enfoque, se supone que los datos provienen de alguna distribución y luego nuestro objetivo es determinar cuáles son los parámetros para esa distribución en particular. Sin embargo, se supone que esos parámetros son fijos (quizás incorrectamente). Usamos nuestro modelo para describir los datos, y luego incluso probamos para garantizar que los datos se ajustan a nuestro modelo.

El **enfoque bayesiano**, mediante las estadísticas bayesianas, en cambio, modelan cómo razonan realmente las personas cuando se les da con algunos datos. Tenemos un conjunto de datos y los usamos para actualizar nuestro modelo con la probabilidad de que ocurra algo. **En las estadísticas bayesianas, usamos los datos para describir el modelo en lugar de usar un modelo primero y confirmarlo más tarde con datos como el enfoque frecuentista.**

El teorema de Bayes calcula el valor de  $P(A | B)$ , que denota cuál es la probabilidad de que ocurra A, dado que B ya ha ocurrido. En la mayoría de los casos, B sería un evento observado por ej. si llovió ayer, y A sería una predicción: ¿hoy va a llover?. En nuestro campo de estudio, **B** es generalmente lo que denominamos **datos observados para entrenar nuestro modelo** y **A** son

los **datos a predecir** (permiten descubrir a qué clase pertenece un nuevo punto de datos). La ecuación para el teorema de Bayes es la descrita en la **Fórmula 2**.

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

**Fórmula 2**

Donde:

$P(A | B)$ : probabilidad “a posteriori”

$P(A)$ : probabilidad “a priori”.

$P(B | A)$ : probabilidad de B en la hipótesis A.

#### Ejemplo basado en nuestra implementación:

Para comprender mejor Naive Bayes, estudiaremos y analizaremos un ejemplo de clasificación de animales del zoológico, donde clasificamos a los animales según su clase/tipo. En nuestro dataset utilizamos todas las features posibles (16) para utilizar los modelos de Bayes y KNN y predecir en base a esto a qué clase pertenecen dichos animales (de las 7 clases disponibles). Pero ahora, **para simplificar y explicar sencillo Bayes, suponemos que queremos encontrar la probabilidad de que un animal con un conjunto de características X1 y X2 sea de Tipo 1, es decir, vertebrados.**

$P(A)$ , en este contexto, es la probabilidad de que este animal sea un vertebrado. Podemos calcular  **$P(A)$** , probabilidad a priori, observando directamente nuestro conjunto de datos de entrenamiento y calculando de ahí el porcentaje de animales son vertebrados. Si nuestro conjunto de datos de entrenamiento contiene 30 vertebrados por cada 100 animales,  $P(A)$  es 30/100 o 0.3.

$P(B)$ , en este contexto, es la probabilidad de que este animal contenga las características 'X1' y 'X2'. Del mismo modo, podemos calcular  **$P(B)$**  calculando el porcentaje de animales en nuestro conjunto de datos de entrenamiento que contiene estas características específicas. Si 10 animales de cada 100 de nuestro conjunto de datos de entrenamiento contienen las

características 'X1' y 'X2', P (B) es 10/100 o 0.1. Tener en cuenta que NO nos importa si el animal es vertebrado o no al calcular este valor.

**P(B | A)** es la probabilidad de que un animal contenga las características 'X1' y 'X2' SI es un vertebrado. También es fácil de calcular desde nuestro conjunto de datos de entrenamiento. Examinamos nuestro conjunto de entrenamiento para animales vertebrados y calculamos el porcentaje de ellos que contienen las características X1 y X2. De nuestros 30 vertebrados, si 6 contienen las características 'X1' y 'X2', entonces P (B | A) se calcula como 6/30 o 0.2.

Y por último utilizamos el teorema de Bayes para calcular **P (A | B)**, que es la probabilidad de que un animal que tiene las características particulares 'X1' y 'X2' sea un vertebrado. Usando la **Fórmula 2**, vemos que el resultado es 0.6:

$$P(A|B) = (0.2 \times 0.3) / 0.1 = 0.6$$

Esto nos indica que **si un animal tiene las características 'X1' y 'X2', hay un 60 por ciento de posibilidades de que sea vertebrado.**

De esta manera, la probabilidad la calculamos mediante una evidencia directa en nuestro conjunto de datos de entrenamiento, no mediante alguna supuesta distribución. En contraste, un enfoque frecuentista de esto dependería de que creáramos una distribución de la probabilidad de diferentes características en los animales para calcular ecuaciones similares.

### Ejemplo más detallado del funcionamiento de Naive Bayes basado en nuestra implementación:

Como vimos anteriormente, la ecuación del teorema de Bayes se puede usar **para calcular la probabilidad de que una muestra dada pertenezca a una clase dada**. Por lo tanto, **podemos usar la ecuación como un algoritmo de clasificación**.

Usando **C** como una **clase dada** y **D** como un **punto de datos de muestra en nuestro dataset**, creamos los elementos necesarios para el teorema de Bayes y Naive Bayes. **Naive Bayes es un algoritmo de clasificación que utiliza el teorema de Bayes para calcular la probabilidad de que una nueva muestra de datos pertenezca a una clase en particular**.

De esta manera tenemos nuestra **Fórmula 2** se transforma en la **Fórmula 3**.

$$P(C | D) = (P(D | C) \times P(C)) / P(D)$$

**Fórmula 3.**

Donde:

- **P (C)** es la **probabilidad de una clase**, que se calcula a partir del conjunto de datos de entrenamiento (como vimos con el ejemplo de vertebrados). Simplemente calculamos el porcentaje de muestras en nuestro conjunto de datos de entrenamiento que pertenecen a la clase dada.
- **P (D)** es la **probabilidad de un punto de datos dado de muestra en nuestro dataset**. Podría ser difícil para nosotros calcular esto, ya que la muestra puede contener muchas características diferentes, pero dado que es una constante en todas las clases y no necesitamos calcularla en absoluto. **Más adelante veremos cómo solucionar este problema**.
- **P (D | C)** es la **probabilidad de que la muestra de datos D pertenezca a la clase C**. Esto también podría ser difícil de calcular debido a las diferentes características de D. Sin embargo, aquí es donde entra en juego la parte “ingenua” (naive) del algoritmo Naive Bayes.

Naive Bayes nos permite asumir ingenuamente que **cada característica es independiente la una de la otra**. En lugar de calcular la probabilidad total de  $P(D | C)$ , calculamos la probabilidad de cada característica  $D_1, D_2, D_3, \dots$  y así sucesivamente. Luego, los multiplicamos juntos:  $P(D | C) = P(D_1 | C) \times P(D_2 | C) \dots \times P(D_n | C)$  como si fueran independientes entre sí. Cada uno de estos valores es relativamente fácil de calcular; simplemente calculamos el porcentaje de veces que es igual en nuestro conjunto de datos de muestra.

De esta manera lo simplificamos a la **Fórmula 4**.

Considerando 2 características D:

$$P(C | D) = (P(D1 | C) \times P(D2 | C) \times P(C)) / P(D)$$

**Fórmula 4.**

Por el contrario, si quisiéramos realizar una **versión Bayes no ingenua** para esta parte, tendríamos que calcular las correlaciones entre las diferentes características para cada clase. **Tal cálculo es demasiado complejo y, por lo tanto, inviable**, sin grandes cantidades de datos o modelos de análisis de lenguaje suficientes.

De esta manera, con Naive Bayes el algoritmo es simple: **calculamos  $P(C | D)$  para cada clase posible, ignorando el término  $P(D)$ . Luego elegimos la clase con la mayor probabilidad. Como el término  $P(D)$  es constante en cada una de las clases, ignorarlo no tiene impacto en la predicción final.** De esta manera, aplicamos la **Fórmula 5** a cada una de las clases.

$$P(C | D) = (P(D1 | C) \times P(D2 | C) \times P(C)) / \cancel{P(D)}$$

$$\rightarrow P(C | D) = P(D1 | C) \times P(D2 | C) \times P(C)$$

**Fórmula 5.**

### Ejemplo con valores del funcionamiento de Naive Bayes:

Supongamos que tenemos los siguientes valores de características/features **binarios** de una muestra en nuestro conjunto de datos: [0, 0, 0, 1]. Nuestro conjunto de datos de entrenamiento contiene dos clases con un 25 por ciento (1 de 4) pertenecientes a la clase 1 y un 75 por ciento (3 de 4) de muestras pertenecientes a la clase 0.

Las probabilidades de los valores de características para cada clase son las siguientes:

Para la clase 0: [0.3, 0.4, 0.4, 0.7]

Para la clase 1: [0.7, 0.3, 0.4, 0.9]

La interpretación de estos valores sería: para la característica 1, es un 1 en el 30 por ciento de los casos para la clase 0.

Ahora, podemos **calcular la probabilidad de que esta muestra pertenezca a la clase 0**.  $P(C = 0) = 0.75$ , que es la probabilidad de que la clase sea 0.  $P(D)$  no es necesaria para el algoritmo Naive Bayes. Observamos este cálculo:

$$P(D | C = 0) = P(D1 | C = 0) \times P(D2 | C = 0) \times P(D3 | C = 0) \times P(D4 | C = 0)$$

$$P(D | C = 0) = 0.3 \times 0.6 \times 0.6 \times 0.7 = \mathbf{0.0756}$$

El segundo y tercer valor son 0.6, ya que el valor de esas feature en la muestra fue 0. Las probabilidades enumeradas son para valores de 1 para cada característica. Por lo tanto, la probabilidad de un 0 es su inverso:  $P(0) = 1 - P(1)$ .

Ahora, podemos calcular la probabilidad de que la muestra de datos pertenezca a esta clase  **$P(C=0|D)$** . Un punto importante a tener en cuenta es que **no tenemos que calcular  $P(D)$**  para esto, por lo que esta no es una probabilidad real. Sin embargo, es lo suficientemente bueno para comparar con el mismo valor para la probabilidad de la clase 1. Veamos el cálculo:

$$P(C=0|D) = P(C=0) P(D|C=0) = 0.75 * 0.0756 = \mathbf{0.0567}$$

Ahora calculamos los mismos valores para la clase 1:

$$P(C=1) = 1 - P(C=0) = 0.25$$

Como sabemos el cálculo de  $P(D)$  no es necesario para Naïve Bayes. Vemos el cálculo entonces:

$$P(D|C=1) = P(D1|C=1) \times P(D2|C=1) \times P(D3|C=1) \times P(D4|C=1) = 0.7 \times 0.7 \times 0.6 \times 0.9 = \mathbf{0.2646}$$

$$P(C=1|D) = P(C=1)P(D|C=1) = 0.25 \times 0.2646 = \mathbf{0.06615}$$

Cabe destacar que las probabilidades de calcular  $P(D|C)$  fueron mucho, mucho más altas para la clase 0. Esto se debe a que introdujimos una creencia previa de que la mayoría de las muestras generalmente pertenecen a la clase 0.

De esta manera **como  $P(C=1|D) > P(C=0|D)$  el algoritmo clasificará como perteneciente a la clase 1** y NO a la 0. Sin embargo la decisión final fue muy cercana,

#### Conclusión Naïve Bayes:

Como conclusión, podemos decir que como Naive Bayes se basa en el teorema de Bayes (lo que significa que utiliza datos anteriores para actualizar el modelo); nos permite que el modelo sea flexible y actualice nuevos datos e incorpore una creencia previa.

Además, la parte “ingenua” (naive) permite calcular fácilmente las probabilidades sin tener que lidiar con complejas correlaciones entre las características (utilizamos las características de los animales como leche, plumas, pelo, huevos, etc.).

En la mayoría de las tareas basadas en texto (NLP), Naive Bayes puede funcionar bastante bien. Como otra ventaja, el clasificador Naive Bayes no tiene ningún parámetro que deba establecerse como el  $k$  de KNN o el “alfa” de Ridge (aunque hay algunos parámetros por si se desean hacer algunos ajustes).

### 3- Desarrollo.

#### 3.1-KNN: Implementación Ejemplo N°1

##### 3.1.1-Data set.

En este ejemplo, el Data Set que cargamos será de un archivo csv de entrada (zoo.csv) obtenido mediante un data set de la página de Kaggle. Este DS consiste en 101 animales de un zoológico. Hay 16 features para describir a los animales y cada animal ya está "etiquetado"/ clasificado en 7 distintas clases: Mammal, Bird, Reptile, Fish, Amphibian, Bug e Invertebrate.

Como características / features de entrada (nuestras columnas) tendremos:

- animal\_name: Unique for each instance
- hair: Boolean
- feathers: Boolean
- eggs: Boolean
- milk: Boolean
- airborne: Boolean
- aquatic: Boolean
- predator: Boolean
- toothed: Boolean
- backbone: Boolean
- breathes: Boolean
- venomous: Boolean
- fins: Boolean
- legs: Numeric (set of values: {0,2,4,5,6,8})
- tail: Boolean
- domestic: Boolean
- catsize: Boolean
- class\_type: Numeric (integer values in range [1,7])



Ejemplo de los primeros 10 registros con sus 16 features (+animal name y class type):

| animal_name | hair | feathers | eggs | milk | airborne | aquatic | predator | toothed | backbone | breathes | venomous | fins | legs | tail | domestic | catsize | class_type |
|-------------|------|----------|------|------|----------|---------|----------|---------|----------|----------|----------|------|------|------|----------|---------|------------|
| aardvark    | 1    | 0        | 0    | 1    | 0        | 0       | 1        | 1       | 1        | 1        | 0        | 0    | 4    | 0    | 0        | 1       | 1          |
| antelope    | 1    | 0        | 0    | 1    | 0        | 0       | 0        | 1       | 1        | 1        | 0        | 0    | 4    | 1    | 0        | 1       | 1          |
| bass        | 0    | 0        | 1    | 0    | 0        | 1       | 1        | 1       | 1        | 0        | 0        | 1    | 0    | 1    | 0        | 0       | 4          |
| bear        | 1    | 0        | 0    | 1    | 0        | 0       | 1        | 1       | 1        | 1        | 0        | 0    | 4    | 0    | 0        | 1       | 1          |
| boar        | 1    | 0        | 0    | 1    | 0        | 0       | 1        | 1       | 1        | 1        | 0        | 0    | 4    | 1    | 0        | 1       | 1          |
| buffalo     | 1    | 0        | 0    | 1    | 0        | 0       | 0        | 1       | 1        | 1        | 0        | 0    | 4    | 1    | 0        | 1       | 1          |
| calf        | 1    | 0        | 0    | 1    | 0        | 0       | 0        | 1       | 1        | 1        | 0        | 0    | 4    | 1    | 1        | 1       | 1          |
| carp        | 0    | 0        | 1    | 0    | 0        | 1       | 0        | 1       | 1        | 0        | 0        | 1    | 0    | 1    | 1        | 0       | 4          |
| catfish     | 0    | 0        | 1    | 0    | 0        | 1       | 1        | 1       | 1        | 0        | 0        | 1    | 0    | 1    | 0        | 0       | 4          |
| cavy        | 1    | 0        | 0    | 1    | 0        | 0       | 0        | 1       | 1        | 1        | 0        | 0    | 4    | 0    | 1        | 0       | 1          |

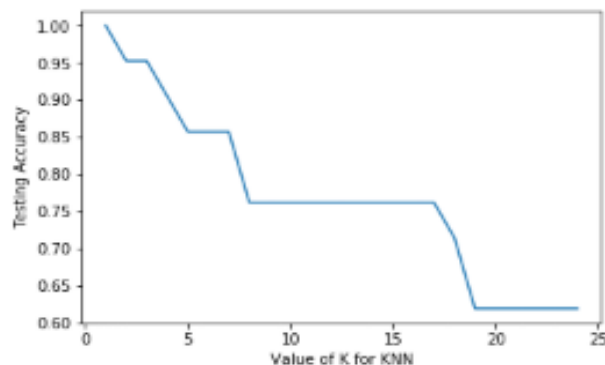
### 3.1.2-¿Qué queremos predecir?

Nuestro objetivo es, mediante 16 distintas features numéricas y booleanas de nuestro data set, **predecir la variable class\_type**. De esta manera, podremos predecir la clasificación de un animal (esté o no en este zoológico, ya que puede ser un animal nuevo).

### 3.1.3-Implementación del modelo de KNN

Antes de implementar nuestro modelo KNN, debemos seleccionar el K óptimo para utilizar nuestro algoritmo. Para esto entrenamos al algoritmo K veces y nos quedamos con el modelo que nos dé la mejor predicción (el mejor score).

De esta manera realizamos un barrido para los distintos valores de K (del 1 al 25). Esto se describe en la siguiente imagen:



La imagen anterior nos describe que para  $K < 5$  nuestro algoritmo predecirá correctamente (Ya que tenemos una eficiencia mayor al 90% para estos casos).

El K óptimo que utilizamos nosotros es 2. De esta manera generamos nuestro modelo KNN con K=2, lo entrenamos y obtenemos su score:


```
: k_Optimo=2
knn = KNeighborsClassifier(n_neighbors=k_Optimo) #Le asignamos este k=2.
knn.fit(X_train,y_train) #Se va a entrenar al algoritmo para ese K=2 con Los x_train
#e Y_train que elegimos previamente.
y_pred=knn.predict(X_test) #Y ahora hacemos una predicción con Los X_Test.
score = metrics.accuracy_score(y_test,y_pred) #Cálculo del score para k=2

print("Precisión de KNN con K optimo:",score) #printamos el score.

Precisión de KNN con K optimo: 0.9523809523809523
```

Luego realizamos predicciones sobre ese modelo y analizamos los resultados:


```
print(dicClasses[y_predict[0]])
print(dicClasses[y_predict[1]])
print(dicClasses[y_predict[2]])
```



Mammal  
Bird  
Reptile

Por último, para probar, elegimos un valor de K NO óptimo para realizar predicciones. Para esto utilizamos un K=20 y lo colocamos en un nuevo modelo KNN. Y realizamos las mismas predicciones que hicimos con nuestro k óptimo = 2. Estas predicciones nos dieron así:

```
print(dicClasses[y_predict[0]])
print(dicClasses[y_predict[1]])
print(dicClasses[y_predict[2]])
```



Mammal  
Bird  
Mammal

### 3.1.4-Resultados.

Observamos que las 2 primeras predicciones coinciden en ambos modelos. En la última predicción los modelos difieren: para el modelo con el k óptimo = 2 dice que es un Reptil y para nuestro modelo con k = 20 dice que es un mamífero.

Esta última predicción seguramente va a ser la incorrecta ya que nuestro score para K=20 es muy bajo (0,61). Y de esta manera, para este modelo, nuestro algoritmo no es fiable, predice mal.

### 3.1.5-Diferencias entre KNN y K-means.

A diferencia de KNN que se encuentra dentro de los algoritmos de aprendizaje supervisado, K-Means es un algoritmo NO supervisado, de esta manera tenemos 2 diferencias principales:

- Con **KNN**, al ser un algoritmo de aprendizaje **supervisado** de **clasificación**, los modelos de machine learning que construimos se basan en datos de entrenamiento etiquetados. En nuestro ejemplo tenemos varias características de los animales del zoológico y ya están etiquetados en distintas clases (mamíferos, invertebrados, etc.); ya están etiquetados nuestros datos. Al hacer esto, le estamos diciendo a nuestro algoritmo qué parámetros corresponden a determinados animales, y, basado en este mapeo, el algoritmo aprenderá a clasificar nuevos animales utilizando esos parámetros que le proporcionamos.
- En cambio, con **K-means**, al ser un algoritmo de aprendizaje **NO supervisado** de **clustering**, los modelos de machine learning que construimos se basan en datos de entrenamiento que NO están etiquetados. Dado que no hay etiquetas disponibles, se debe extraer información basada únicamente en los datos que se proporcionan. Para nuestro caso de animales de zoológicos la propia máquina utilizará sus propios criterios de clasificación y de acuerdo a estos clasificará a los animales de acuerdo a sus features /características. En este caso NO le damos clases pre-armadas, sino que el algoritmo de aprendizaje no supervisado trata de separar el conjunto de datos dado en varios grupos de la mejor manera posible.

**K-means tiene una forma muy similar de trabajar como KNN, pero la desventaja, como dijimos anteriormente, es que los datos NO están etiquetados.**

#### Funcionamiento de K-Means:

K-Means permite agrupar objetos en k grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster (para esto se suele usar la distancia cuadrática). De esta manera, se crean zonas o “clusters” que tienen puntos centrales llamados “centroides”. Y, dentro de ese cluster, metemos a todas nuestras muestras. El cluster lo crea el algoritmo a su manera, ya que como vimos anteriormente, la propia máquina utilizará sus propios criterios para clasificar las muestras.

El algoritmo consta de tres pasos:

1. **Inicialización:** una vez elegido el número de grupos,  $k$ , se establecen  $k$  centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente.
2. **Asignación objetos a los centroides:** cada objeto de los datos es asignado a su centroide más cercano.
3. **Actualización centroides:** se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo.

→ Luego se repiten los pasos 2 y 3 hasta que los centroides no se mueven, o se mueven por debajo de una distancia umbral en cada paso.

El algoritmo *k-means* resuelve un **problema de optimización**, siendo la función a optimizar (minimizar) la suma de las distancias cuadráticas de cada objeto al centroide de su cluster.

### 3.1.6-Casos de uso de KNN y K-means.

Como KNN es un algoritmo de clasificación, entonces se lo usa por ejemplo para:

- Sistemas de recomendación.
- Predecir acciones de la bolsa de valores va a subir o bajar, utilizando los valores históricos.
- Cualquier tipo de problema donde se requiera clasificar elementos en clases dadas (por ejemplo para clasificar las góndolas del supermercado o estantes de libros, clasificar planetas, etc.).

En cambio, como K-means es un algoritmo de Clustering, se lo usa por ejemplo para:

- Biotecnología: mediante la identificación de la función de genes y proteínas.
- Ciencias sociales: Patrones en los modelos de comportamiento, patologías criminales,...
- Economía: **encontrar aspectos y patrones en común dentro de los clientes** para encontrar grupos y enfocar productos o servicios mediante campañas.

## 3.2-Clasificador Bayesiano: Implementación Ejemplo N°2

### 3.2.1-Data set.

Utilizaremos el mismo DS de **3.1.1-Data set** (animales de zoológico).

### 3.2.2-¿Qué queremos predecir?

Nuestro objetivo es predecir lo mismo que predecimos para el modelo de KNN (clasificación de animales dentro de las 7 clases según sus 16 features), con la diferencia de que ahora tendremos un modelo de Bayes, el cual veremos si tendrá mayor o menor score que el modelo de KNN.

### 3.2.3-Implementación modelo Bayes.

Implementamos el modelo de Bayes con los mismos datos mediante “GaussianNB()” e imprimimos su score:

```
X_train,X_test,y_train,y_test = train_test_split(x_data,y,test_size=0.2,random_state=4)
nb = GaussianNB() #Definimos el algoritmo de Naive Bayes.
nb.fit(X_train,y_train) #Entrenamos el algoritmo con los x_train e Y_train que elegimos previamente.
y_pred=nb.predict(X_test) #Y ahora hacemos una predicción con los X_Test.

from sklearn.metrics import accuracy_score
print("Precisión de Naive Bayes:",metrics.accuracy_score(y_test, y_pred))

Precisión de Naive Bayes: 0.9847619847619848
```

Luego realizamos predicciones y vemos los resultados:

```
print(dicClasses[y_predict[0]])
print(dicClasses[y_predict[1]])
print(dicClasses[y_predict[2]])
```

```
Mammal
Bird
Mammal
```

Y como análisis “extra” obtenemos el reporte de clasificación y nuestra matriz de confusión (explicados en 3.3- Cómo evaluar la Eficacia de cada algoritmo) para tener una evaluación más precisa de nuestro modelo:

|             | precision | recall | f1-score | support |
|-------------|-----------|--------|----------|---------|
| 1           | 1.00      | 1.00   | 1.00     | 7       |
| 2           | 1.00      | 1.00   | 1.00     | 5       |
| 4           | 1.00      | 1.00   | 1.00     | 1       |
| 5           | 1.00      | 1.00   | 1.00     | 1       |
| 6           | 1.00      | 0.33   | 0.50     | 3       |
| 7           | 0.67      | 1.00   | 0.80     | 4       |
| avg / total | 0.94      | 0.90   | 0.89     | 21      |

```
[[ 7 0 0 0 0 0]
 [ 0 5 0 0 0 0]
 [ 0 0 1 0 0 0]
 [ 0 0 0 1 0 0]
 [ 0 0 0 0 1 2]
 [ 0 0 0 0 0 4]]
```

#### 3.2.4-Resultados.

Vemos que los valores de precisión, recall, y f1-score dieron valores cercanos a 1, de esta manera podemos confiar en las predicciones de nuestro modelo.

Podemos observar que el 3ro lo clasificó como Mammal. Como tenemos un buen valor de Score confiaremos en su predicción. El modelo que nosotros creamos **GaussianNB()** implementa esta fórmula para obtener las probabilidades para el valor que le damos y cada clase:

$$P(C | D) = (P(D | C) \times P(C)) / P(D)$$

Y cuando hacemos la predicción **solo nos devuelve la clase que obtuvo la mayor de estas probabilidades.**

### 3.3- Cómo evaluar la Eficacia de cada algoritmo.

Anteriormente, para algoritmos de Regresión teníamos el coeficiente de determinación  $R^2$  para medir/determinar la calidad del modelo para replicar los resultados; este coeficiente lo utilizábamos cuando teníamos modelos lineales (o no) de regresión. En el caso de los algoritmos de clasificación tenemos diferentes métricas de evaluación. Scikit-learn nos provee de funciones tales como 'accuracy', 'true-positive', 'false-positive', (TP,FP,TN,FN), 'precision', 'recall', 'F1 score', etc. para evaluar la performance del clasificador.

Pero, si medimos la efectividad de nuestro modelo solo por la cantidad de aciertos que tuvo, sólo teniendo en cuenta a la clase mayoritaria podemos estar teniendo una falsa sensación de que el modelo funciona bien.

Por esto, tenemos que utilizar la llamada “Confusión matrix”, la cual está “resumida” en el “Report Clasification”, que nos ayudará a comprender el funcionamiento y precisión de nuestro algoritmo. Los elementos de análisis que tenemos que tener en consideración son:

- La **Accuracy** del modelo es el número total de predicciones correctas dividido por el número total de predicciones.
- La **Precisión** de una clase define cuan confiable es un modelo en responder si un punto pertenece a esa clase.
- El **Recall** de una clase expresa cuan bien puede el modelo detectar a esa clase.
- El **F1 Score** de una clase es dada por la media armonía de precisión y recall ( $2 \times \text{precision} \times \text{recall} / (\text{precision} + \text{recall})$ ) digamos que combina precisión y recall en una sola métrica.

→ Los 3 últimos los sacamos mediante el Report Clasification. Y **Accuracy** lo obtenemos mediante “**metrics.accuracy\_score**”.

Tenemos cuatro casos posibles para cada clase:

- **Alta precision y alto recall:** el modelo maneja perfectamente esa clase
- **Alta precision y bajo recall:** el modelo no detecta la clase muy bien, pero cuando lo hace es altamente confiable.
- **Baja precisión y alto recall:** La clase detecta bien la clase pero también incluye muestras de otras clases.
- **Baja precisión y bajo recall:** El modelo no logra clasificar la clase correctamente.

Cuando tenemos un dataset con desequilibrio (muchos datos en una clase y pocos en otra), suele ocurrir que obtenemos un alto valor de precisión en la clase Mayoritaria y un bajo recall en la clase Minoritaria.

#### 4-Conclusiones.

Esta práctica nos permitió analizar 2 de los más famosos algoritmos de Clasificación: KNN y Naive Bayes, analizando su funcionamiento teórico y práctico mediante una implementación en Python para clasificar animales del zoológico.

Ambos modelos que entrenamos e implementamos (KNN con  $k$  óptimo = 2 y Naive Bayes) nos dieron un buen score, así que podríamos confiar en las predicciones de ambos modelos.

Además, este trabajo nos permitió tener una idea más clara de **cómo evaluar la eficacia de algoritmos de clasificación** (que es distinta que el score que usábamos para algoritmos de Regresión) y cómo influye la elección del  $K$  en el algoritmo de KNN.



## 5-Mejoras a desarrollar.

- Buscar ejemplos de clusterización para k-means e implementarlo en código.
- Ejemplificar mejor la sección “**3.3- Cómo evaluar la Eficacia de cada algoritmo**” agregando las cuentas para sacar cada métrica con los valores VP, VN, FN y FP explicando cada uno. Y agregar conceptos de **especificidad** y **sensibilidad**.
- Analizar especificidad y sensibilidad en KNN.
- Analizar la matriz de confusión, classification\_report, sensibilidad y especificidad para el modelo de KNN (lo mismo que se analizó para Bayes).
- Podríamos evaluar la **Curva Rock** para cada modelo.

## 6-Bibliografía.

- KNN:
  - <https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/>
- K-Means:
  - [https://www.uniovi.es/compnum/laboratorios\\_py/kmeans/kmeans.html](https://www.uniovi.es/compnum/laboratorios_py/kmeans/kmeans.html)
- Clasificador Bayesiano:
  - <https://github.com/Aniruddha-Tapas/Applied-Machine-Learning/blob/master/Zoo%20Animal%20Classification%20using%20Naive%20Bayes.ipynb>
- Información acerca de cómo calcular los Score en Clasificadores:
  - <https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>
  - <https://bookdown.org/content/2274/metodos-de-clasificacion.html>