Universidad Autónoma de Querétaro

Facultad de Ingeniería
Maestría en Inteligencia Artificial
Machine Learning
Sheila Leyva López
Cecilia Gabriela Rodríguez Flores

Clasificación de la base de datos Mushroom mediante diferentes algoritmos de aprendizaje automático.

20 de enero de 2023.

Objetivo

El objetivo principal de este proyecto final es implementar diferentes algoritmos de aprendizaje automático para conocer cuál de ellos tiene un mejor rendimiento al clasificar la base de datos de mushroom.

Introducción

La base de datos de mushroom es una base de datos proporcionada por la Universidad de Irvine, California [1]. Esta base de datos cataloga hongos según sus características físicas en dos clases, venenosos y comestibles. Cuenta con un total 8124 instancias y 22 atributos.

Debido a la gran cantidad de atributos está base de datos puede ser un buen ejercicio para aplicar el algoritmo de análisis de componentes principales. Y al presentar datos faltas es una base de datos a las cual se le debe de realizar una imputación. Haciendo de esto un ejercicio completo para aplicar los conocimientos adquiridos durante la asignatura de aprendizaje máquina.

Marco Teórico

Es importante definir conceptos básicos del aprendizaje máquina que permitan entender el contexto en el que se trabaja, como son las etapas fundamentales para la implementación de técnicas de aprendizaje máquina:

Preparación de datos

Es una etapa fundamental en cualquier proyecto, debido a que por medio de ella se pueden inicializar correctamente los datos para su posterior procesamiento y análisis.

Imputación

Los métodos de imputación consisten en estimar los valores ausentes en base a los valores válidos de otras variables y/o casos de la muestra. La estimación se puede hacer a partir de la información del conjunto completo de variables o bien de algunas variables especialmente seleccionadas. Usualmente los métodos de imputación se utilizan con variables métricas (de intervalo o de razón), y deben aplicarse con gran precaución porque pueden introducir relaciones inexistentes en los datos reales.

Idealmente las técnicas de imputación no deberían cambiar la distribución de los datos. Así que si originalmente se tenía una distribución normal (con forma de campana), entonces después de la imputación se debería mantener esta distribución original.

En general, las técnicas de imputación se pueden clasificar de la siguiente manera:

- Técnicas por información externa o deductiva, se suele utilizar cuando los datos faltantes se pueden deducir de otras instancias completas.
- Técnicas deterministas, las cuales funcionan cuando, de acuerdo con las mismas condiciones de los datos se producen las mismas respuestas. Entre ellas, se encuentran la Imputación por regresión, Imputación de la media (o moda), Imputación por media de clases, Imputación por vecino más cercano, imputación por algoritmo EM.

- Técnicas estocásticas, son aquellas que cuando, se repite el método bajo las mismas condiciones, producen resultados diferentes. Entre estas técnicas se encuentran: Imputación aleatoria y la Imputación secuencial (hot deck).

Análisis de componentes principales (PCA)

El análisis de componentes principales (PCA) simplifica la complejidad de los datos de alta dimensión al tiempo que conserva tendencias y patrones, en otras palabras, es básicamente un procedimiento estadístico para convertir un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de valores de variables linealmente no correlacionadas, en donde primeramente se realiza una transformación lineal de los datos en un nuevo sistema de coordenadas ortogonales. En este nuevo sistema de coordenadas las componentes principales están ordenadas automáticamente según la varianza de la proyección de datos, es decir, según la cantidad de información que contengan. Debido a esto, se puede reducir la dimensión de los datos resultantes en el nuevo espacio eliminando las componentes principales que presenten una menor varianza, es decir, que aporten menos información.

Para poder realizar esta reducción de dimensionalidad con PCA, se tienen que realizar los siguientes pasos:

- Obtener lo datos, en todas sus dimensiones
- Restar la media. Para que PCA funcione de manera correcta, se requiere restar la media de cada dimensión.
- Calcular la matriz de covarianza
- Calcular los eigenvalores y los eigenvectores de la matriz de covarianza
- Escoger los componentes y formar el vector de características, en donde el elemento con el número mayor es el componente principal del set de datos.
- En general, una vez que los eigenvectores se encuentran en la matriz de covarianza, se ordenan de mayor a menor dando estos componentes principales en orden

Así bien, PCA ofrece múltiples beneficios, pero también adolece de ciertas deficiencias.

Ventajas:

1. Fácil de calcular. PCA se basa en álgebra lineal, que es computacionalmente fácil de resolver por computadoras.

2. Acelera otros algoritmos de aprendizaje automático. Los algoritmos de aprendizaje automático convergen más rápido cuando se entrenan en los componentes principales en lugar del conjunto de datos original.

3. Contrarresta los problemas de datos de alta dimensión. Los datos de alta dimensión hacen que los algoritmos basados en regresión se sobre ajusten fácilmente. Al usar PCA de antemano para reducir las dimensiones del conjunto de datos de entrenamiento, evitamos que los algoritmos predictivos se sobre ajusten.

Desventajas:

1. Baja interpretabilidad de los componentes principales. Los componentes principales son combinaciones lineales de las características de los datos originales, pero no son tan fáciles de interpretar. Por ejemplo, es difícil saber cuáles son las características más importantes del conjunto de datos después de calcular los componentes principales.

2. La compensación entre la pérdida de información y la reducción de la dimensionalidad. Aunque la reducción de la dimensionalidad es útil, tiene un costo. La pérdida de información es una parte necesaria de PCA.

3. PCA asume una correlación entre características. Si las características no están correlacionadas, PCA no podrá determinar los componentes principales.

4. PCA es sensible a la escala de las características. Al suponer que se cuenta con dos características: una toma valores entre 0 y 1000, mientras que otra toma valores entre 0 y 1. PCA estará extremadamente sesgado hacía que la primera característica sea el primer componente principal, independientemente de la variación máximo real dentro de los datos. Por lo cual, es importante estandarizar los valores primeros.

5. PCA no es robusto frente a valores atípicos. Similar al punto anterior, el algoritmo estará sesgado en conjuntos de datos con fuertes valores atípicos.

6. Las implementaciones técnicas a menudo no asumen valores perdidos. Al calcular PCA utilizando herramientas de software estadístico, a menudo asumen que el conjunto de características no tiene valores perdidos.

Analizar

En esta sección se utilizarán (que dato se van a utilizar) o dividirá los datos ya preparados para el siguiente paso, para este trabajo se optó por crear subconjuntos de datos:

Regla 80-20

Este método consiste en la creación de subconjuntos, en los cuales el 80% de los datos de considera para el entrenamiento, mientras que un 20% corresponde a pruebas del modelo de clasificación.

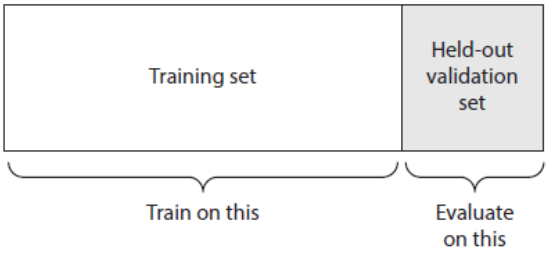


Ilustración 1. Regla 80-20.

Validación cruzada K-FOLD

Existen situaciones en las que no se cuenta con un gran número de datos para entrenamiento y prueba. La validación cruzada permite dividir el conjunto de datos, de tal forma que, los datos se pueden utilizar en el entrenamiento y en prueba. Específicamente en la validación cruzada de k iteraciones (kfold cross validation) se divide el conjunto de datos original en subconjuntos, de modo que, durante el entrenamiento se toma cada k subconjunto como un conjunto de prueba total, mientras que los demás subconjuntos se toman como entrenamiento, esto se repetirá k veces, y cada vez el conjunto de prueba será diferente y el resto de los datos se utilizará para

entrenamiento [2]. En cada iteración se busca el mayor valor de precisión y se elige la iteración que haya tenido el mejor desempeño.

Ilustración 2. Validación cruzada kfold.

Modelos

A continuación, se presentan los diversos modelos de machine learning a probar para la realización de este trabajo:

Árboles de decisión

Los árboles de decisión son una técnica de aprendizaje automático supervisado y ha sido utilizada en múltiples ámbitos desde la inteligencia artificial hasta la economía. Esta técnica va tomando las decisiones siguiendo la estructura de un árbol. Los nodos intermedios representan soluciones y las hojas la predicción o clasificación que se está buscando [3].

Las principales ventajas de este algoritmo es que es fácil de entender y explicar a personas que no están familiarizadas con técnicas de inteligencia artificial, se adapta a cualquier tipo de datos y descubre cuales son los atributos relevantes. Por otro lado, sus desventajas son que no extrapolan bien fuera de su rango de entrenamiento y tienden al sobreajuste [3].

ID3

ID3 es el acrónimo de “Iterative Dichotomiser 3” o dicotimizador iterativo en español. Fue desarrollado por J. Ross Quinlan. La salida del algoritmo se puede presentar como como un grafo en forma de árbol cuyos componentes son:

- Nodo raíz: Es el nodo principal localizado en la parte superior. De este nodo parten ramas hacia otros nodos inferiores, de los cuales pueden salir más ramas hacia otros nodos.
- Nodos terminales: Como su nombre lo indica, son nodos donde termina el flujo y que ya no son raíz de ningún otro nodo, también puede denominarse hoja. Estos nodos terminales contienen la clasificación a la cual pertenece el objeto que ha conducido hasta dicho nodo.
- Nodos intermedios: representan preguntas con respecto al valor de uno de los atributos.
- Ramas: representan las posibles respuestas que los atributos pueden tomar.

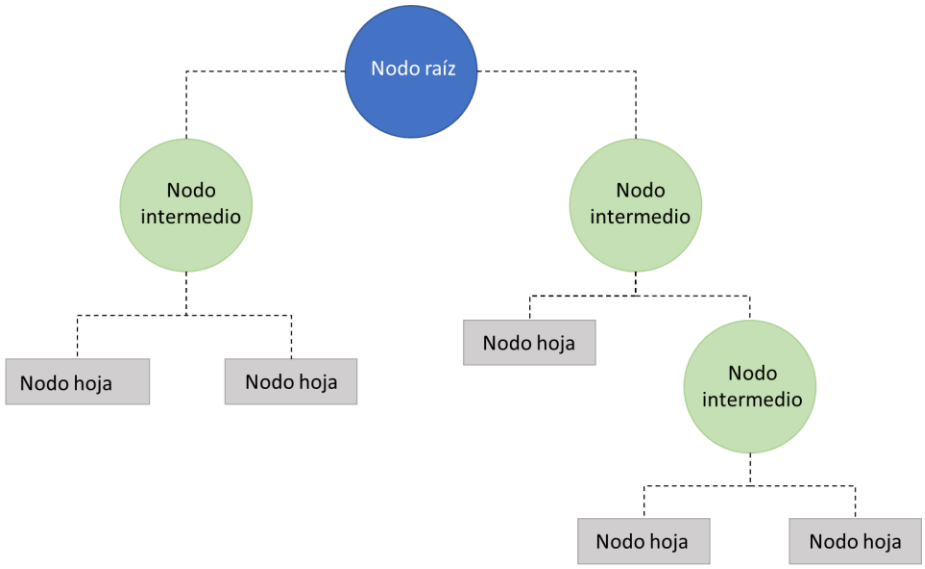


Ilustración 3. Estructura general de un árbol de decisión.

El objetivo principal del algoritmo ID3 es determinar, para un conjunto de datos, el atributo más importante, es decir, aquel que posea el mayor poder discriminatorio para dicho conjunto, el cual será denominado nodo raíz; este atributo es usado para la clasificación de la lista de objetos, basados en los valores asociados con él mismo. Después de haber hecho la primera prueba de

atributo se arrojará un resultado, el cual es en sí mismo un nuevo problema de aprendizaje de árbol de decisión, con la diferencia que contará con menos ejemplos y un atributo menos, por lo que, cada atributo que se selecciona se descarta para la siguiente prueba [3].

Este proceso se realiza mediante el apoyo de dos operaciones de probabilidad, la entropía y la ganancia. La entropía es una medida de incertidumbre y es usado para ayudar a decidir qué atributo debe ser el siguiente en seleccionarse. Se calcula de la siguiente forma:

$$\text{Entropia}(S) = \sum_{i=1}^{n} -p_i \log_2 p_i \tag{1}$$

Mientras que la ganancia de información es una medida de discriminación e indica el siguiente atributo que debe ser seleccionado y se calcula de la siguiente forma:

$$\text{Gan Inf}(S, A) = \text{Entropia}(S) - \sum_{v \in V(A)} \frac{|Sv|}{|S|} \text{Entropia}(Sv) \tag{2}$$

Prueba Cobertura

La prueba de cobertura es una de las pruebas más utilizadas en la evaluación del desempeño de los árboles de decisión, dado que, se pretende conocer si se recorren todos los posibles caminos pasando por todos los nodos intermedios y finales. Idealmente se espera una cobertura igual o cercana al 100%[3].

Perceptrón multicapa

El perceptrón multicapa es una red neuronal artificial formada por múltiples capas, de tal manera que tiene capacidad para resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón. El perceptrón multicapa puede estar total o localmente conectado.

Algoritmo de K vecinos más cercanos

El algoritmo K-vecinos más cercanos (KNN) es un tipo de algoritmo de aprendizaje automático supervisado que se utiliza para la clasificación, la regresión y la detección de valores atípicos. Es extremadamente fácil de implementar en su forma más básica, pero puede realizar tareas bastante complejas. Es un algoritmo de aprendizaje perezoso ya que no tiene una fase de

entrenamiento especializada. Más bien, utiliza todos los datos para el entrenamiento mientras clasifica (o retrocede) un nuevo punto de datos o instancia [4].

KNN es un algoritmo de aprendizaje no paramétrico, lo que significa que no asume nada sobre los datos subyacentes. Esta es una característica extremadamente útil ya que la mayoría de los datos del mundo real no siguen ningún supuesto teórico, por ejemplo, separabilidad lineal, distribución uniforme, etc.

Como primera etapa se debe seleccionar un número k de vecinos, posteriormente se calcula la distancia de cada uno de los puntos hacia todos los demás. Se toman los k vecinos más cercanos y se le atribuye al punto que se está analizando la clase más frecuente en los vecinos que se analizan. Finalmente se selecciona el mejor número de vecinos.

Regresión logística

La regresión logística es una técnica matemática utilizada en aprendizaje automático para la clasificación de datos. Básicamente, es una neurona. Recibe este nombre dado que, primero se realiza una combinación lineal y después se aplica una función logística, aunque no es una regresión sino un modelo de clasificación [4]. El proceso se describe a través del diagrama

de la Ilustración 1. Básicamente, lo que en regresión lineal se denominaba hipótesis $\boldsymbol{h\theta}$ ahora es Z y $\sigma(Z)$ es la función de transferencia definida por la función sigmoidea.

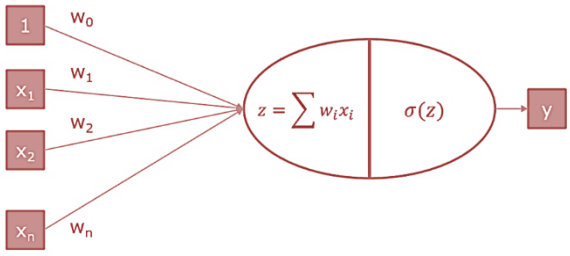


Ilustración 4. Esquema regresión logística.

Donde:

- X: atributos
- W: coeficientes o pesos.

- $\sigma(Z)$: función logística sigmoide

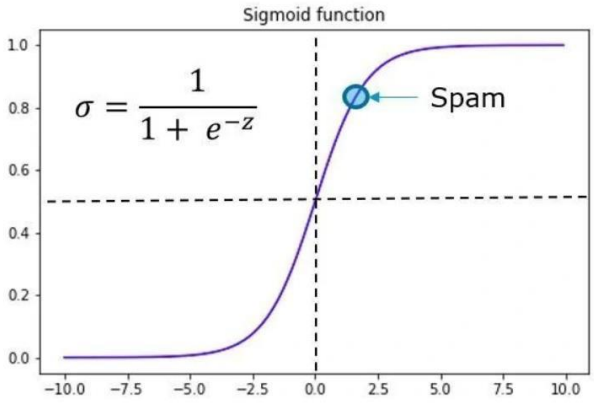


Ilustración 5. Función sigmoide.

Learning rate

En el aprendizaje automático y las estadísticas, la tasa de aprendizaje o learning rate (lr) es un parámetro de ajuste en un algoritmo de optimización que determina el tamaño del paso en cada iteración mientras avanza hacia un mínimo de una función de pérdida. Dado que influye en qué medida la información recién adquirida anula la información anterior, representa metafóricamente la velocidad a la que "aprende" un modelo de aprendizaje automático.

K-means ++

El algoritmo de k-means es la técnica más popular, la cual se basa en centroides. Sin embargo, presenta una desventaja, depende en gran medida de la inicialización de los centroides[5]. Es decir, si el centroide se inicializa en un punto alejado podría terminar sin puntos asociados a él o en caso contrario podría terminar con más de un clúster asociado, esto y otros factores pueden ocasionar un clustering deficiente.

Debido a la desventaja que presenta la técnica Kmeans, surge K-means++, que intenta solucionar la situación asegurando que los centroides se inicialicen de mejor manera, de tal forma que se mejore la calidad del clustering; esencialmente esta es la única diferencia entre k-means y K-means ++, el resto del método es idéntico[6].

Principalmente, la inicialización consiste en seleccionar de forma aleatoria el primer centroide los puntos de datos, para posteriormente, calcular la distancia de cada punto de datos al centroide más cercano elegido en el paso uno. Después, se selecciona el siguiente centroide de los puntos de datos, de tal manera que el punto que se desea elegir como centroide tiene la distancia máxima medida desde el centroide más cercano y tiene la mayor probabilidad de ser elegido, y así, sucesivamente se repiten los dos pasos anteriores hasta que se cumplan para todos los k-ésimos centroides[5].

$$J = \sum_{k \in K} \sum_{i} z_k^i |x_i - \mu_k|^2$$

Donde:
xi: representa al objeto.
μk: media del cluster

A continuación, se enumeran algunas limitaciones de la agrupación de esta técnica de agrupamiento.

- El resultado está muy influenciado por la entrada original, es decir, por el número de agrupaciones.
- En algunos casos, las agrupaciones muestran vistas espaciales complejas, por lo cual la esta técnica no es buena elección para estos casos.

Por otra parte, se presentan algunas de las desventajas de este algoritmo de agrupamiento:

- El algoritmo exige la especificación inferida del número de agrupaciones.
- El algoritmo no funciona con conjuntos de datos no lineales y no es capaz de tratar datos ruidosos y valores atípicos.
- No es directamente aplicable a datos categóricos
- La distancia euclidiana puede ponderar de forma desigual los factores subyacentes.

El algoritmo no es variante de la transformación no lineal, es decir, proporciona resultados diferentes con distintas representaciones de los datos.

Métricas

Exactitud

La exactitud es la relación entre los simples correctamente clasificados y el número total de simples en el conjunto de datos de evaluación. Esta métrica es conocida por ser engañosa en el caso de diferentes proporciones de clases, ya que asignar simplemente todos los simples a la clase predominante es una forma fácil de lograr una alta precisión [3].

$$ACC = \frac{\#\ correctly\ classified\ samples}{\#\ all\ samples} = \frac{TP+TN}{TP+FP+TN+FN} \quad (3)$$

Precisión

La precisión es la capacidad de un modelo para identificar sólo objetos relevantes, es decir, es el porcentaje de predicciones positivas correctas entre todas las verdades básicas dadas [3].

$$P_r = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{N-S} FP_n} = \frac{\sum_{n=1}^{S} TP_n}{all\ detection} \quad (4)$$

Sensitividad

La sensitividad o recall se define como el porcentaje de predicciones correctas tomadas de la clase de predicciones correctas sin tomar en cuenta los falsos positivos [3].

$$REC = \frac{\#\ true\ positive\ samples}{\#\ samples\ classified\ positive} = \frac{TP}{TP + FN} \quad (5)$$

F1 Score

La F1 Score calcula la media armónica de la precisión y recall de forma que se enfatiza al valor más bajo entre ambas [3].

$$F1 = 2\ x\ \frac{precision\ x\ recall}{precision + recall} = \frac{2\ x\ TP}{2\ x\ TP + FP + FN} \quad (6)$$

Materiales y métodos

Herramientas utilizadas

- Google Collaboratory
- Conjunto de datos: *Indicadores personales clave de enfermedad cardíaca*
- AMD Ryzen 9 5900HS with Radeon Graphics 3.30 GHz
- RAM 16.0 GB
- 64-bit operating system, x64-based processor

Conjunto de datos

En este trabajo, se utilizará la base de datos Mushroom. Este conjunto de datos incluye descripciones de muestras hipotéticas correspondientes a 23 especies de hongos sin branquias en la familia Agaricus y Lepiota. Cada especie se identifica como definitivamente comestible, definitivamente venenosa o de comestibilidad desconocida y no recomendada. Esta última clase se combinó con la venenosa. La Guía establece claramente que no existe una regla simple para determinar la comestibilidad de un hongo.

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	b	c	l
8120	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	b	v	l
8121	e	f	s	n	f	n	a	c	b	n	...	s	o	o	p	o	o	p	b	c	l
8122	p	k	y	n	f	y	f	c	n	b	...	k	w	w	p	w	o	e	w	v	l
8123	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	o	c	l

8124 rows × 23 columns

Ilustración 6. Visualización de los atributos e instancias del conjunto de datos Mushroom.

La base de datos de mushroom es una base de datos proporcionada por la Universidad de Irvine, California. Esta base de datos cataloga hongos según sus características físicas en dos clases, venenosos y comestibles. Cuenta con un total 8124 instancias y 22 atributos.

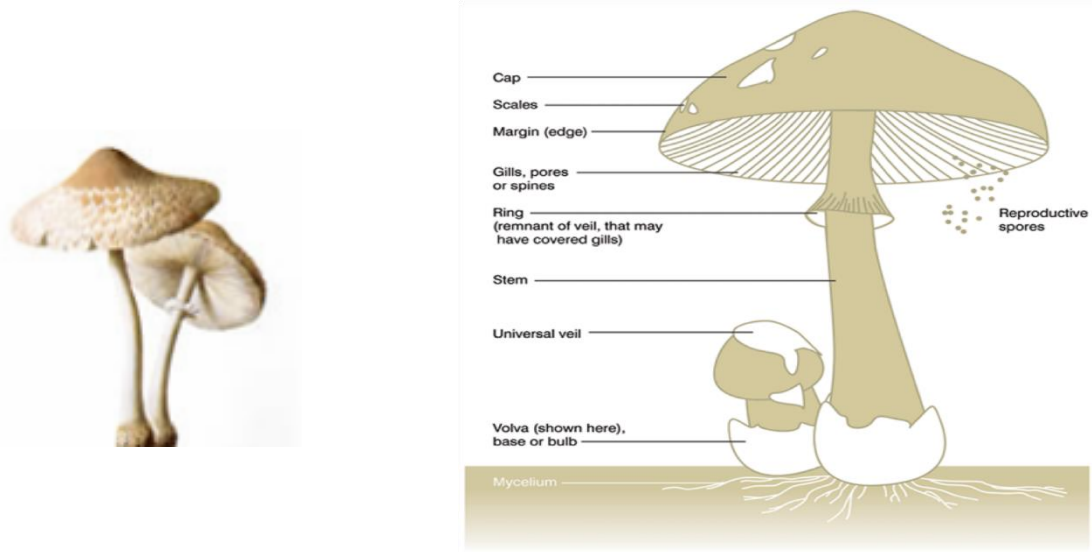


Ilustración 7. Representación de un hongo. [7].

El análisis exploratorio de datos, o EDA, es una parte integral de la comprensión del conjunto de datos de Mushroom. Antes de avanzar hacia la clasificación, es vital familiarizarse con las diferentes relaciones dentro de los datos. El análisis de estas relaciones proporcionará información sobre cómo interpretar los resultados de los modelos anteriores. Hacer preguntas sobre estas relaciones de antemano también podría proporcionar conocimientos adicionales sobre relaciones que tal vez no sabíamos que existían. Esta sección investigará más a fondo la distribución de datos y hará preguntas específicas sobre la información que se encuentra dentro del conjunto de datos. A continuación, se agrega una breve información sobre las observaciones de cada uno de los atributos incluidos en este conjunto de datos [1]:

- **cap-shape:** Ciertamente, no hay muchas características de la forma del sombrero que puedan decidir definitivamente si el hongo es venenoso o comestible. Si el hongo tiene nudos, la mayoría de las veces podría ser venenoso. Una tapa en forma de campana es más probable que sea comestible.
  Observaciones: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s.

- **cap-surface:** Si el hongo tiene una superficie más fibrosa, es más probable que sea comestible. Los hongos lisos y escamosos tienen un poco más de probabilidad de ser venenosos.
  Observaciones: fibrous=f, grooves=g, scaly=y, smooth=s.
- **cap-color:** Si el hongo tiene una superficie más blanca o gris, es probable que sea comestible. Los colores de sombrero rojo o amarillo tienden a ser más venenosos.
  Observaciones: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
- **bruises:** Si el hongo tiene magulladuras, es probable que sea comestible. Si el hongo no tiene magulladuras, es más probable que sea venenoso.
  Observaciones: bruises=t, no=f
- **odor:** Si el hongo no tiene olor, es muy probable que sea comestible. Es muy probable que cualquier olor desagradable o acre sea venenoso.
  Observaciones: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
- **gill-attachment:** No hay características que definitivamente clasifiquen el hongo como comestible o venenoso.
  Observaciones: attached=a, descending=d, free=f, notched=n
- **gill-spacing:** Si las branquias están muy juntas, es como si fuera venenoso. Si están abarrotados, es más probable que sean comestibles.
  Observaciones: close=c, crowded=w, distant=d
- **gill-size:** Si las branquias son estrechas, es como si fuera venenoso. Si son anchos, es más probable que sean comestibles.
  Observaciones: broad=b, narrow=n
- **gill-color:** Es casi seguro que los colores beige sean venenosos. También es probable que los colores gris y chocolate sean venenosos. Es probable que los colores blanco, morado y marrón sean comestibles.
  Observaciones: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

- **stalk-shape:** Los tallos son difíciles de analizar para determinar su comestibilidad. **Observaciones: enlarging=e, tapering=t**
- **stalk-root:** Es muy probable que las raíces faltantes sean venenosas. Club y raíces iguales son probablemente comestibles.
  Observaciones: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
- **stalk-surface-above-ring:** Los tallos lisos por encima del anillo probablemente serán comestibles. Es probable que los tallos sedosos sean venenosos.
  Observaciones: fibrous=f, scaly=y, silky=k, smooth=s
- **stalk-surface-below-ring:** Los tallos lisos por encima del anillo probablemente serán comestibles. Es probable que los tallos sedosos sean venenosos.
  Observaciones: fibrous=f, scaly=y, silky=k, smooth=s
- **stalk-color-above-ring:** Es probable que los colores blanco y gris sean comestibles. Es probable que el beige, el marrón y el morado sean venenosos.
  Observaciones: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
- **stalk-color-below-ring:** Es probable que los colores blanco y gris sean comestibles. Es probable que el beige, el marrón y el morado sean venenosos.
  Observaciones: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
- **veil-type:** Los velos son difíciles de analizar para la comestibilidad.
  Observaciones: partial=p, universal=u
- **veil-color:** Los velos son difíciles de analizar para la comestibilidad.
  Observaciones: brown=n, orange=o, white=w, yellow=y
- **ring-number:** Los números de anillo son difíciles de analizar para la comestibilidad. **Observaciones: none=n,one=o,two=t**
- **ring-type:** Es muy probable que los tipos de anillos colgantes sean comestibles. Es probable que los tipos de anillos grandes y evanescentes sean venenosos. **Observaciones: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z**

- **spore-print-color:** Es muy probable que el negro y el marrón sean comestibles. Es muy probable que los colores tostado y blanco sean venenosos.
  Observaciones: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
- **population:** Es probable que varios hongos que se encuentran en la población sean venenosos. Es probable que numerosos y abundantes sean comestibles.
  Observaciones: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
- **habitat:** Los hongos que se encuentran en pastos y áreas boscosas probablemente sean comestibles. Es probable que los hongos que se encuentran en los caminos y las hojas sean venenosos. **Observaciones: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d**

Métodos

Para la realización de esta tarea, fue necesario importar las librerías de Pandas, numpy, etc., así como el montaje de Google Drive en el entorno de ejecución a fin de poder hacer uso de la base de datos propuesta.

Posteriormente, se optó por crear diversas funciones a fin de distribuir las tareas de una manera más eficiente como se muestra a continuación:

Funciones para la preparación y analisis de los datos

- Función Numero_atributos: obtiene el número de atributos y los atributos que comprende la base de datos en cuestión. Esta función requiere como datos de entrada únicamente la base de datos.

```
def Numero_atributos(df_na):
    num_atributos_ = df_na.shape[1]    #número de columnas
    atributos_ = list(df_na.columns)   #extrae columnas-->lista

    print(f'El dataset tiene {num_atributos_} atributos:')

    for i in range(num_atributos_):
        print(f'{i+1}.- {atributos_[i]} ')
    return num_atributos_, atributos_
```

Ilustración 8. Función para calcular el número y el contenido de cada uno de los atributos.

- Función Observaciones: obtiene los atributos, el tipo de atributo y sus correspondientes observaciones que comprende cada uno de los atributos de la base de datos en cuestión. Esta función requiere como datos de entrada la base de datos y la lista de atributos.

```
def Observaciones(df_o, atributos_o):
    observaciones_ = {}
    for i, atributo in enumerate(atributos_o):
        obs_ = df_o[atributo].value_counts() #Return a Series containing counts of unique values.
        obs_l_ = list(obs_.index)            #Lista de los valores únicos encontrados
        obs_ = pd.DataFrame(obs_.index)

        tipo = df_o[atributo].dtypes

        if tipo == 'object': tipo = 'categórico'
        elif tipo == 'int': tipo = 'entero'
        elif tipo == 'float64': tipo = 'flotante'

        print(f'''El atributo {atributo} es de tipo {tipo}, contiene {len(obs_l_)} observaciones y son las siguientes:
        {obs_l_}
              ''')
        observaciones_.update({atributo:obs_})
    return observaciones_
```

Ilustración 9. Función para obtener el tipo de atributo y las observaciones de cada uno de los atributos.

- Función Numero_instancias: calcula el número de instancias de cada uno de los atributos que comprende la base de datos en cuestión. Esta función requiere como datos de entrada únicamente la base de datos.

```
def Numero_instancias(df_ni):
    num_instancias_ = df_ni.shape[0]
    print(f'El dataset tiene {num_instancias_} instancias')
    return num_instancias_
```

Ilustración 10. Función para calcular el número de instancias.

- Función Datos_Faltantes: obtiene el número de datos faltantes de cada uno de los atributos que comprende la base de datos en cuestión. Esta función requiere como datos de entrada únicamente la base de datos.

```
def Datos_Faltantes(df_):
    faltantes_df = df_.isnull().sum()
    print(f'Datos faltantes por atributo:\n {faltantes_df}')
```

Ilustración 11. Función para conocer el número de datos faltantes.

- Función moda: obtiene el valor de la moda del atributo en cuestión. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def moda(atributo_):
    moda_1 = atributo_.value_counts()
    moda_ = list(moda_1.index)
    return moda_[0]
```

Ilustración 12. Función para calcular la moda.

- Función maximo: obtiene el valor de máximo del atributo en cuestión. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def maximo(columna_):
    r = len(columna_)
    valor_maximo_ = columna_[0]
    for n in range(1, r):
        k = columna_[n]
        if valor_maximo_ < k:
            valor_maximo_ = k
    return valor_maximo_
```

Ilustración 13. Función para calcular el valor máximo.

- Función minimo: obtiene el valor mínimo del atributo en cuestión. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def minimo(columna_):
    r = len(columna_)
    valor_minimo_ = columna_[0]
    for n in range(1, r):
        k = columna_[n]
        if valor_minimo_ > k:
            valor_minimo_ = k
    return valor_minimo_
```

Ilustración 14. Función para conocer el valor mínimo.

- Función media: obtiene el valor de la media del atributo en cuestión. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def media(columna_):
    r = len(columna_)
    sumatoria = 0
    for i in range(r):
        sumatoria += columna_[i]
    media_ = sumatoria / r
    return media_
```

Ilustración 15. Función para conocer la media.

- Función desviación estándar: obtiene la desviación estándar del atributo en cuestión. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def desviacion_estandar(columna_): #s
    r = len(columna_)
    media_columna = media(columna_)
    sumatoria = 0
    for i in range(r):
        s = columna_[i] - media_columna
        s = s**2
        sumatoria += s
    s = sumatoria / (r - 1)
    s = s ** (1/2)
    return s
```

Ilustración 16. Función para conocer la desviación estándar.

- Función varianza: obtiene el valor de varianza del atributo en cuestión. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def varianza(columna_): #s2
    r = len(columna_)
    media_columna = media(columna_)
    sumatoria = 0
    for i in range(r):
        s = columna_[i] - media_columna
        s = s**2
        sumatoria += s
    s_2 = sumatoria / (r - 1)
    return s_2
```

Ilustración 17. Función para conocer la varianza.

- Función estadística: se encarga de obtener el análisis estadístico de la base de datos en cuestión por medio de la obtención de los valores máximos y mínimos, moda, media, desviación estándar y varianza. Esta función requiere como datos de entrada únicamente el atributo en cuestión.

```
def estadistica(atributo_):
    mod = moda(atributo_)
    max = maximo(atributo_)
    min = minimo(atributo_)
    med = media(atributo_)
    std = desviacion_estandar(atributo_)
    var = varianza(atributo_)
    return mod, max, min, med, std, var
```

Ilustración 18. Función general para obtener el análisis estadístico.

- Función cuartiles: obtiene los valores de los cuartiles de acuerdo con el atributo en cuestión. Esta función requiere como datos de entrada la base de datos y el atributo en cuestión.

```
def cuartiles(df_, atributo_):
#1. 25% de los datos es menor que o igual a este valor.
#2. La mediana. 50% de los datos es menor que o igual a este valor.
#3. 75% de los datos es menor que o igual a este valor.
#rango intercuartil. La distancia entre el primer 1er cuartil y el 3er
#cuartil (Q3-Q1); de esta manera, abarca el 50% central de los datos.

#ordenar los elementos de la columna
    v = df_[atributo_]
    v = np.asarray(v.sort_values())
    n = len(v)
    if n % 2 == 0:
        #calcular cuartil uno
        a = int(n/4)
        q1 = v[a]
        #calcular cuartil dos
        b = (n)/2
        c = b + 1
        q2 = (v[b] + v[c])/2
        #calcular cuartil tres
        d = int((3*n)/4)
        q3 = v[d]
    else:
        #calcular cuartil dos
        b = int(n/2)
        q2 = v[b]

        if b % 2 == 0:
            #calcular cuartil uno
            a = int(n/4)
            q1 = (v[a] + v[a-1])/2
            #calcular cuartil tres
            d = int((3*(n))/4)
            q3 = (v[a] + v[d-1])/2
        else:
            #calcular cuartil uno
            a = int(n/4)
            q1 = v[a]
            #calcular cuartil tres
            d = int((3*n)/4)
            q3 = v[d]
    print(f'Q1: {q1}, Q2: {q2}, Q3: {q3}')
    return q1, q2, q3
```

Ilustración 19. Función para obtener los cuartiles.

- Función outlier: indica si el atributo en cuestión cuenta o no con outliers por medio de la utilización de la función cuartiles. Esta función requiere como datos de entrada la base de datos y el atributo en cuestión.

```
def outlier(df_,atributo_):
    v = df_[atributo_]
    q1, q2, q3 = cuartiles(df_,atributo_)
    out = []
    for valor in v:
        if valor < q1:
            if abs(valor-q1) > 1.5*(q3-q1):
                out.append(valor)
        if valor > q3:
            if abs(valor-q3) > 1.5*(q3-q1):
                out.append(valor)

    if len(out) == 0:
        print(f'No existen valores atípicos en el atributo {atributo_}')
    else:
        print(f'Se encontraron {len(out)} valores atípicos')
        print(f'Los valores atípicos encontrados son: {np.asarray(out)}')
    return out
```

Ilustración 20. Función para obtener los outliers.

- Función Balance_clases: obtiene la cantidad de cada clase del atributo de decisión de la base de datos en cuestión. Esta función requiere como datos de entrada únicamente la base de datos.

```
def Balance_clases(df):
    atributo_d = input('¿Cuál es tu atributo de decisión?:')
    clases = df[atributo_d]
    cl = clases.value_counts()
    cl_ = list(cl.index)
    porcentajes_ = (cl*100)/sum(cl)
    print(porcentajes_)
    print(f'El dataset cuenta con {len(cl_)} clases y son las siguientes: {cl_}')
```

Ilustración 21. Función para calcular el balance de clases.

- Función valores_faltantes: obtiene el porcentaje total de los valores faltantes, así como el porcentaje de valores faltantes para ca uno de los atributos que comprende la base de datos en cuestión. Esta función requiere como datos de entrada únicamente la base de datos.

```
def valores_faltantes(dataset):
    missing_values_count = dataset.isnull().sum()
    total_missing = missing_values_count.sum()
    #Porcentaje de datos faltantes
    total_missing_porcent = total_missing/(np.product(dataset.shape))*100
    print('Porcentaje total de valores faltantes:',total_missing_porcent,'%')
    print('')
    print('Porcentaje de valores faltantes de cada atributo:')
    for col in dataset.columns:
        VP_missing = np.mean(dataset[col].isnull())
        print('{} - {}%'.format(col,round(VP_missing*100)))
```

Ilustración 22. Función para calcular los valores faltantes.

- Función imputación_aleatoria: realiza la imputación del atributo con datos faltantes de la base de datos en cuestión. Esta función requiere como datos de entrada la base de datos y el atributo en cuestión.

```
def imputacion_aleatoria(df_aleat, atributo_):
    df_a = df_aleat.copy()
    faltantes_df = df_a[atributo_].isnull().sum()
    no_faltantes = df_a[df_a[atributo_].notnull()]

    min = no_faltantes[atributo_].min()
    max = no_faltantes[atributo_].max()

    tipo = df_a[atributo_].dtypes
    print(tipo)
    if tipo == 'O':
        random_string_list = []
        for i, atributo in enumerate(atributo_):
            obs_ = df_a[atributo_].value_counts() #Return a Series containing counts of unique values.
            obs_l_ = list(obs_.index)            #Lista de los valores únicos encontrados
        for i in range(faltantes_df):
            x = random.choices(obs_l_)
            # random_string_list.append(x)
            #df_a.loc[df_a[atributo_].isnull(),atributo_] = random_string_list
            df_a.loc[df_a[atributo_].isnull(),atributo_] = x

    elif tipo == 'int64':
        random_int_list = []
        for i in range(faltantes_df):
            x = random.radint(min,max)
            random_int_list.append(x)
        df_a.loc[df_a[atributo_].isnull(),atributo_] = random_int_list

    elif tipo == 'float64':
        random_float_list = []
        for i in range(faltantes_df):
            x = round(random.uniform(min,max),2)
            random_float_list.append(x)
        df_a.loc[df_a[atributo_].isnull(),atributo_] = random_float_list

    return df_a
```

Ilustración 23. Función para realizar la imputación aleatoria.

- Función imputación_HotDeck: realiza la imputación del atributo con datos faltantes de la base de datos en cuestión. Esta función requiere como datos de entrada la base de datos y el atributo en cuestión.

```
def Imputacion_HotDeck(columna,Copy_hotdeck):
    missing_values_count = Copy_hotdeck[columna].isnull().sum()
    total_missing = missing_values_count.sum()
    print(total_missing)
    Total = len(Copy_hotdeck[columna])
    numero = np.random.randint(0, Total)
    V=[] #valores de la columna
    IA = [] #index aleatorio
    I =[]
    for index, row in Copy_hotdeck.iterrows():
        V.append(row[columna])
    Indice= recorrido(Copy_hotdeck, columna,numero)
    IA.append(Indice) #Guarda el indice aleatorio
    print('Lista de valores de la columna:')
    print(V)
    print('')
    for i in range(len(V)): #Recorrido de los indices de la columna
        if i == IA[0]: #Si el indice coincide con el indice aleatorio
            #print(V[i]) #comprobar que el valor coincida con el valor del registro
            for i in (V[i:]): #Recorrido de los valores desde el indice aleatorio
                I.append(i) #Lista de valores secuenciales al valor aleatorio propuesto
    while len(I)<(int(total_missing)*2): #Si la cantidad de valores no es mayor al doble de P50
        for i in (V[:Total]): #Recorrido de los valores desde el indice aleatorio
            I.append(i) #Lista de valores secuenciales al valor aleatorio propuesto
    print('Lista de valores secuenciales al valor aleatorio propuesto:')
    print(I)
    I.pop(0) #Eliminación del primer valor de la lista
    print('Lista de valores secuenciales al valor aleatorio propuesto:')
    print(I)
    print('')
    newlist = [x for x in I if pd.isnull(x) == False] #Lista de valores secuenciales al valor ale
    print('Lista de valores secuenciales al valor aleatorio propuesto sin valores faltantes:')
    print(len(newlist), newlist)
    print('')
    for n, i in enumerate(V):
        if i is np.nan: #Encontrar la posición que contenga valor nan
            V[n] = newlist[0] #Reemplazar el valor nan por el primer valor de lista newlist
            newlist.pop(0) #Eliminar el primer valor de la lista de valores secuenciales al valor
    print('Sustitución de los valores en la lista original')
    print(V)
    print('')
    Copy_hotdeck[columna] = V
    return Copy_hotdeck
```

Ilustración 24. Función para realizar la imputación HotDeck.

- Función norm_min_max: realiza la normalización de la base de datos. Esta función requiere como datos de entrada la base de datos en cuestión.

```
def norm_min_max(datos):
  lim_sup = []
  lim_inf = []
  rangoDatos =[]
  maxNorm = 1
  minNorm = 0
  rango = maxNorm - minNorm
  for i in range (0,datos.columns.size):
    lim_sup.append(datos.iloc[:,i].max())
    lim_inf.append(datos.iloc[:,i].min())
    rangoDatos.append(lim_sup[i] - lim_inf[i])
  nombres = datos.columns.values.tolist()
  datosNorm = pd.DataFrame(columns = nombres)

  for j in range(len(datos.columns)):
    varNorm = []
    var = datos.iloc[:,j]
    for i in range(len(datos)):
      D = var[i] - lim_inf[j]
      DPct = D/rangoDatos[j]
      dNorm = rango*DPct
      varNorm.append(minNorm+dNorm)
    datosNorm.iloc[:,j] = varNorm
  datos = datosNorm
  return datos
```

Ilustración 25. Función para normalizar los datos.

- Función matriz_cov: se encarga de obtener la matriz de covarianza de los elementos de la base de datos en cuestión.

```
def matriz_cov(data):
    atributos = data.columns
    n = len(atributos)
    m = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            X = data[atributos[i]]
            Y = data[atributos[j]]
            m[i][j] = (((X-X.mean())*(Y-Y.mean())).sum())/(len(X)-1)
    return m
```

Ilustración 26. Función para calcular la matriz de covarianza.

- Función PCA: brinda los porcentajes de cada uno de los atributos de acuerdo con su relevancia dentro de la base de datos, a fin de discernir los atributos con mayor peso.

```
def PCA(datos,col_decision):
    datos1 = datos.drop([col_decision],axis=1) #Eliminando el atributo de decisión
    #Ajustar los datos restando la media a cada atributo
    datos_A = pd.DataFrame(columns=datos1.columns,index=range(len(datos1)))
    for i in datos_A.columns:
        datos_A[i] = datos1[i] - datos1[i].mean()
    #datos_A
    matrix = matriz_cov(datos_A)
    #sns.heatmap(matrix)
    L,V =  np.linalg.eig(matrix)
    #Obtener el porcentaje de covarianza de cada uno de los atributos
    total = L.sum()
    p = (L/total)*100
    pca =[]
    columnas1 = datos_A.columns.values
    for index, row in enumerate(p):
        print(columnas1[index] + ':',row)
```

Ilustración 27. Función de PCA.

- Función division_80_20: realiza la separación del conjunto de datos de acuerdo al porcentaje de 80% para el conjunto de entrenamiento y 20% para el conjunto de prueba. Esta función requiere como entradas los datos y las etiquetas.

```
def method_8020(x,y):
    train_x = x[0 : int(len(x)*0.8)]
    train_y = y[0 : int(len(y)*0.8)]
    test_x = x[int(len(x)*0.8) : ]
    test_y = y[int(len(y)*0.8) : ]
    return train_x, train_y, test_x, test_y
```

Ilustración 28. Función propuesta para la división del conjunto de datos.

- Función kfold: realiza la separación del conjunto de datos a fin de utilizar todos los datos en la etapa de entrenamiento y prueba. Esta función requiere como entradas los valores de x, y, k y i.

```
def kfold(x,y,k,i):
    section = int(len(x)/k)
    x_test = x.iloc[i * section : (i+1) * section, :]
    x_train = x_.drop([i * section,((i+1) * section)-1 , 1],axis=0)

    y_test = y.iloc[i * section : (i+1) * section]
    y_train = y_.drop([i * section,((i+1) * section)-1 , 1],axis=0)

    return x_test, y_test, x_train, y_train
```

Ilustración 29. Función propuesta para kfold.

Funciones para KNN

- Función euclidean_distance: se encarga de obtener la distancia entre cada uno de los datos. Esta función requiere como entrada los atributos.

```
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):                    #for k in range(23)
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)
```

Ilustración 30. Función para calcular la distancia euclidiana.

- Función get_neighbors: realiza la extracción de los n vecinos más cercanos, de acuerdo con el valor de k propuesto. Esta función requiere como entradas el número de n vecinos, la lista de distancias ordenadas.

```
def get_neighbors(k,distances_ord):
  neighbors = list()
  indices = list()
  for i in range(k):
    neighbors.append(distances_ord[i][0])
    indices.append(distances_ord[i][1])
  return neighbors,indices
```

Ilustración 31. Función propuesta para obtener los vecinos cercanos.

- Función most_common: realiza el conteo de la cantidad de ceros y unos. Esta función requiere los valores de la columna de etiquetas.

```
def most_common(output_values):
    return max(set(output_values), key=output_values.count)
```

Ilustración 32. Función propuesta most_common.

- Función predict_classification: realiza el método de knn. Esta función requiere como entradas los datos, las etiquetas y el número de n vecinos.

```
def predict_classification(x,y, k):
    predict = []
    for j in range(0,len(x)):
        # Inicializacion de las distancias.
        distances = []
        x = np.array(x)
        for i ,example in enumerate(x):
            distance = euclidean_distance(example,x[j])# Calculate the Euclidean distance between two vectors
            distances.append((distance, i))
        distances.pop(j)
        distances_ord = sorted(distances)
        neighbors,indices = get_neighbors(k,distances_ord)
        output_values = y.iloc[indices]
        output_values = output_values.to_numpy().tolist()
       # print(type(output_values))
        #print(output_values)
        predict.append(most_common(output_values))
    return predict
```

Ilustración 33. Función propuesta para el método de knn.

Funciones para Regresión Logística

- Función regre_logistic: realiza la actualización del cálculo de los pesos de acuerdo con el número de épocas. Esta función requiere como entradas los valores de x, y, theta, alpha y épocas.

```
def regre_logist(x, y, theta, alpha, epocas,tolerance):
    iterations = 1
    for i in tqdm(range(epocas)):
        Z = np.dot(x,theta)
        ft = 1 / (1 + np.exp(-Z))
        dJ = (1/len(x)) * np.dot(np.transpose(x),(ft - y))
        mse.append(MSE(x, y, theta))
        vc.append(cross_entropy(x, y, theta))
        new_theta = theta - alpha * dJ
        #n_theta = [item for lista in new_theta for item in lista]
        it.append(iterations)
        # Stopping Condition
        if np.sum(abs(new_theta - theta)) < tolerance:
            print('Gradient Descent has converged')
            break
        iterations += 1
        theta = new_theta
    return theta, it,mse,vc
```

Ilustración 34. Función propuesta para el modelo de regresión logística.

- Función clasificacion: se enfoca en la obtención de la predicción de los valores de y. Esta función requiere de como entradas los valores de x y los pesos obtenidos de la etapa de entrenamiento.

```
def clasificacion(x,theta):
    Z = np.dot(x,theta)
    ft = 1 / (1 + np.exp(-Z))
    clas = np.round(ft)
    return clas
```

Ilustración 35. Función propuesta para la obtención de la clasificación.

Función accuracy: se enfoca en la obtención de la métrica de exactitud. Esta función requiere de como entradas los valores de y calculados y las etiquetas.

```
def accuracy(ycalculada, yreal):
    coincidencias = np.equal(ycalculada, yreal)
    totalcoin = np.sum(coincidencias)
    porcentaje = (totalcoin/len(ycalculada))*100
    return porcentaje
```

Ilustración 36. Función propuesta para la obtención de la métrica accuracy.

- Función MSE: se enfoca calcular la función de costo. Esta función requiere de como entradas los valores de y, x y thetas.

```
def MSE(x, y, theta):
    h = np.dot(x,theta)
    J = np.sum((h-y)**2)/(2 * (len(h)))
    return J
```

Ilustración 37. Función propuesta para la función de costo.

- Función cross_entropy: se enfoca calcular la función de costo. Esta función requiere de como entradas los valores de y, x y thetas.

```
def cross_entropy(x, y, theta):
    z = np.dot(x,theta)
    p = 1 / (1 + np.exp(-z))
    J = - np.mean(y*np.log(p) + (1-y)*np.log(1-p)) #Cross-entropy cost function
    return J
```

Ilustración 38. Función propuesta para la función de costo.

Funciones para Perceptrón multicapa

- capa_neuronal, es una clase que recibe el número de entradas, el número de capas que tendrá la capa y la función de activación con la que trabajarán. Tiene como propiedades la función de activación, un bias que es determinado de forma aleatoria y los pesos de cada una de las neuronas igualmente inicializados de forma aleatoria.

```
class capa_neuronal():
    def __init__(self, num_entradas, num_neuronas, funcion_activacion):
        self.funcion_activacion = funcion_activacion
        self.bias = np.random.rand(1, num_neuronas) * 2 - 1
        self.w = np.random.rand(num_entradas, num_neuronas) * 2 - 1
```

Ilustración 39. Función propuesta para la capa neuronal.

Función compilar_red, como lo dice el nombre esta función compila la estructura de la red neuronal con un ciclo for que va iterando entre el número de capas.

```
def compilar_red(neuronas_por_capa, funcion_activacion):
    red_neuronal = []
    for n, capa in enumerate(neuronas_por_capa[:-1]):
        red_neuronal.append(capa_neuronal(neuronas_por_capa[n], neuronas_por_capa[n + 1], funcion_activacion))
    return red_neuronal
```

Ilustración 40. Función propuesta para la compilación de la red neuronal.

- Función entrenar_red, recibe como parámetros los datos x, y, la estructura de la red neuronal, la derivada de la función de activación y la derivada de la función de coste. En esta función se van modificando los pesos de cada una de las neuronas, así como sus bias para poder obtener una mejor función para clasificar la información.

```
def entrenar_red(x, y, red_neuronal, sigmoide_derivada, l2_derivada):
    salidas = [(None, x)]
    for n, capa in enumerate(red_neuronal):
        I = np.dot(salidas[-1][1], red_neuronal[n].w) + red_neuronal[n].bias
        _y = red_neuronal[n].funcion_activacion(I)
        salidas.append((I, _y))
    deltas = []
    for n in reversed(range(0, len(red_neuronal))):
        I = salidas[n + 1][0]
        _y = salidas[n + 1][1]
        if n == len(red_neuronal) - 1:
            deltas.insert(0, l2_derivada(_y, y) * sigmoide_derivada(_y))
        else:
            deltas.insert(0, np.dot(deltas[0], _w.T) * sigmoide_derivada(_y))
    _w = red_neuronal[n].w

    red_neuronal[n].bias = red_neuronal[n].bias - (np.mean(deltas[0],
                                                           axis = 0,
                                                           keepdims = True) *
                                                   learning_rate)
    red_neuronal[n].w = red_neuronal[n].w - (np.dot(salidas[n][1].T, deltas[0]) *
                                             learning_rate)

    return salidas[-1][1]
```

Ilustración 41. Función propuesta para la etapa de entrenamiento.

- Función precision_mlp: se encarga de obtener la métrica de precisión de la etapa de entrenamiento del modelo perceptron. Recibe como parámetros los datos predichos del entrenamiento y las etiquetas.

```
def precision_mlp(m_entrenamiento, m_esperada):
    m1 = np.round(m_entrenamiento)
    m1[:,0] = m1[:,0]*1000
    m1[:,1] = m1[:,1]*100
    #m1[:,2] = m1[:,2]*10
    m1 = np.sum(m1, axis=1)
    m2 = zeros_like(m_esperada)
    np.copyto(m2, m_esperada)
    m2[:,0] = m2[:,0]*1000
    m2[:,1] = m2[:,1]*100
    #m2[:,2] = m2[:,2]*10
    m2 = np.sum(m2, axis=1)
    errores = []
    for v1, v2 in zip(m1,m2):
        if v1 == v2: errores.append(0)
        else: errores.append(1)
    porcentaje = 100 * sum(errores) / len(errores)
    return sum(errores), porcentaje
```

Ilustración 42. Función propuesta para obtener la precisión del modelo.

- Función predecir: se evalúa la información de entrada para poder obtener una clasificación. Recibe como parámetros los datos x y la estructura de la red neuronal.

```
def predecir(x, red_neuronal):
    salidas = [(None, x)]
    for n, capa in enumerate(red_neuronal):
        I = np.dot(salidas[-1][1], red_neuronal[n].w) + red_neuronal[n].bias
        _y = red_neuronal[n].funcion_activacion(I)
        salidas.append((I, _y))
    return salidas[-1][1]
```

Ilustración 43. Función propuesta para la función de costo.

Funciones para Kmeans ++

- Función fit: realiza el entrenamiento del algoritmo de k-means ++. Esta función requiere como entrada la base de datos en cuestión.

```
def fit(self, dataset):
    self.x_data = dataset.iloc[:, 0]
    self.y_data = dataset.iloc[:, 1]
    self.X = dataset.iloc[:, [0, 1]]  # not use feature labels
    self.m = self.X.shape[0]  # number of training examples
    self.n = self.X.shape[1]  # number of features.
    self.initial_centroids = []

    first_cen = self.get_random_centroid()
    initial_centroids = self.select_the_others_centroid(first_cen)

    self.plot_initial_centroids(initial_centroids)
    c = self.clustering(initial_centroids)
    return c
```

Ilustración 44. Función para ajustar el modelo.

- Función get_random_centroid: realiza la obtención aleatoria del primer centroide.

```
def get_random_centroid(self):
    return np.random.randint(len(self.x_data))
```

Ilustración 45. Función para obtener centroide.

- Función select_the_others_centroid: se encarga de obtener los demás centroides de acuerdo con el primer centroide obtenido. Esta función requiere como entrada el centroide inicial.

```
def select_the_others_centroid(self, first_cen):
    self.initial_centroids.append((self.x_data[first_cen], self.y_data[first_cen]))
    for i in range(self.n_cluster - 1):
        dis_max = 0
        index_max = 0
        for j in range(len(self.x_data)):
            if j != first_cen and (self.x_data[j], self.y_data[j]) not in self.initial_centroids:
                dis_temp = 0
                for k in self.initial_centroids:
                    pt1 = k
                    pt2 = (self.x_data[j], self.y_data[j])
                    dis_temp += self.euclidean_distance(pt1, pt2)
                if dis_temp > dis_max:
                    dis_max = dis_temp
                    index_max = j
        self.initial_centroids.append((self.x_data[index_max], self.y_data[index_max]))
    #print(self.initial_centroids)
    return np.array(self.initial_centroids)
```

Ilustración 46. Función para obtener centroides.

- Función clustering: realiza la agrupación de los datos de acuerdo con las posiciones de los centroides en cuestión, la cual se va a detener cuando las posiciones de los centroides no varíen. Esta función requiere como entrada los centroides.

```
def clustering(self, centroids):

    old_centroids = np.zeros(centroids.shape)
    stopping_criteria = 0.0001
    self.iterating_count = 0
    self.objective_func_values = []

    while self.euclidean_distance(old_centroids, centroids) >= stopping_criteria:
        clusters = np.zeros(len(self.X))
        C = []
        # Assigning each value to its closest cluster
        for i in range(self.m):
            distances = []
            for j in range(len(centroids)):
                distances.append(self.euclidean_distance(self.X.iloc[i, :], centroids[j]))
            cluster = np.argmin(distances)
            clusters[i] = cluster

        # Storing the old centroid values to compare centroid moves
        old_centroids = centroids.copy()

        # Finding the new centroids
        for i in range(self.n_cluster):
            points = [self.X.iloc[j, :] for j in range(len(self.X)) if clusters[j] == i]
            centroids[i] = np.mean(points, axis=0)

        # calculate objective function value for current cluster centroids
        self.objective_func_values.append([self.iterating_count, self.objective_func_calculate(clusters, centroids)])
        self.plot_centroids(centroids, clusters)
        self.iterating_count += 1

    self.plot_objective_function_values()
    return centroids
```

Ilustración 47. Función para clasificar/agrupar los datos.

- Función predict: se encarga de realizar la etapa de prueba del algoritmo de k-means++, en donde se obtendrá la asignación de cada uno de los datos de acuerdo con la cercanía entre los agrupamientos realizados. Esta función requiere como entradas los centroides y el conjunto de datos de prueba.

```
def predict(self, centroids,X):
    self.D = X.iloc[:, [0, 1]]  # not use feature labels
    self.b = self.D.shape[0]  # number of training examples
    clusters = np.zeros(len(self.D))
    C = []
    # Assigning each value to its closest cluster
    for i in range(self.b):
        distances = []
        for j in range(len(centroids)):
            distances.append(self.euclidean_distance(self.D.iloc[i, :], centroids[j]))
        cluster = np.argmin(distances)
        clusters[i] = cluster

    # calculate objective function value for current cluster centroids
    #C = self.plot_centroids(centroids, clusters)
    return clusters
```

Ilustración 48. Función para la predicción.

- Función euclidean_distance: se encarga de obtener la distancia entre cada uno de los datos. Esta función requiere como entrada los datos.

```
def euclidean_distance(self, a, b):
    return np.sqrt(np.sum((np.array(a) - np.array(b))**2))
```

Ilustración 49. Función para la predicción.

Además, una vez realizado el entrenamiento y prueba de los datos, es necesario visualizar el comportamiento de los modelos por medio de las métricas de precisión, exactitud, sensitividad, f1 score, las cuales se muestran en la Ilustración 13.

```
def metricas(claseP,true_train):
    TP = 0
    FP = 0
    TN = 0
    FN = 0
    #Determinar los TP,TN,FP y FN para la matriz de confusión del entrenamiento
    for i in range(len(claseP)):
        if (true_train[i] == claseP[i]) and true_train[i] == 1:
            TP += 1
        elif (true_train[i] == claseP[i]) and true_train[i] == 0:
            TN += 1
        elif (true_train[i] != claseP[i]) and true_train[i] == 0:
            FP += 1
        else:
            FN += 1

    accuracy = ((TP+TN)/(TP+TN+FP+FN))

    if TP + FP != 0:
        precision = TP/(TP+FP)
    else:
        precision = 0
    if TP + FN != 0:
        sensitividad = TP/(TP+FN)
    else:
        sensitividad = 0
    if precision != 0 and sensitividad != 0:
        f1 = (2*TP)/(2*TP+FP+FN)
    else:
        f1 = 0
    return [accuracy, precision, sensitividad, f1]
```

Ilustración 50. Función propuesta para la evaluación de los modelos.

Diagrama de metodología

Para el análisis de cualquier base de datos se deben seguir los siguientes pasos:

1. Recolección: Definir y cargar la base de datos a utilizar.
2. Preparar: Comprobar la no existencia de datos faltantes, identificación de outliers, así como la normalización de los datos.
3. Analizar: Conocer la distribución de la información por cada atributo, reducción de dimensionalidad, división de los conjuntos de entrenamiento y prueba del modelo por medio de la validación cruzada kfolds 5 y k-fold 10, y la división del 80% de los datos para el entrenamiento y el 20% restante a la etapa de prueba.
4. Etapa de entrenamiento y prueba: Implementación del modelo de KNN, k-means++, Regresión Logística, Perceptrón Multicapa y Árbol de decisión. Así bien, se calcula la métrica de exactitud, precisión, sensitividad y F1 Score para conocer el desempeño obtenido.

A continuación, se muestra el diagrama de flujo que representa el proceso de desarrollo, el cual fue implementado en un programa basado en lenguaje Python.

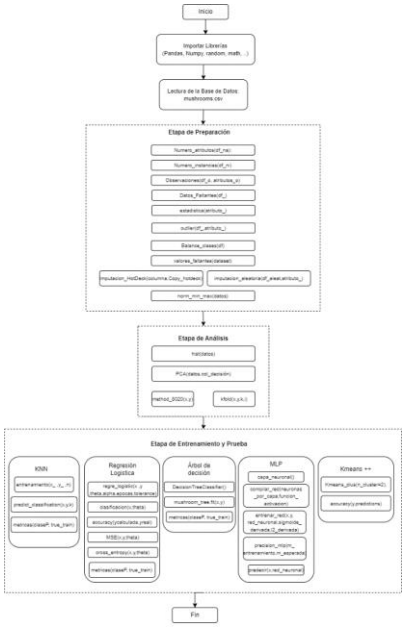


Ilustración 51. Diagrama de flujo de metodología.

Resultados y discusión

A continuación, se presentan las distribuciones de los atributos de la base de datos en cuestión, después de haberle realizado una preparación, la cual comprende los siguientes puntos: la conversión de características lingüísticas a valores categóricos, la comprobación de la no existencia de datos faltantes, la identificación de outliers y la normalización de los datos.

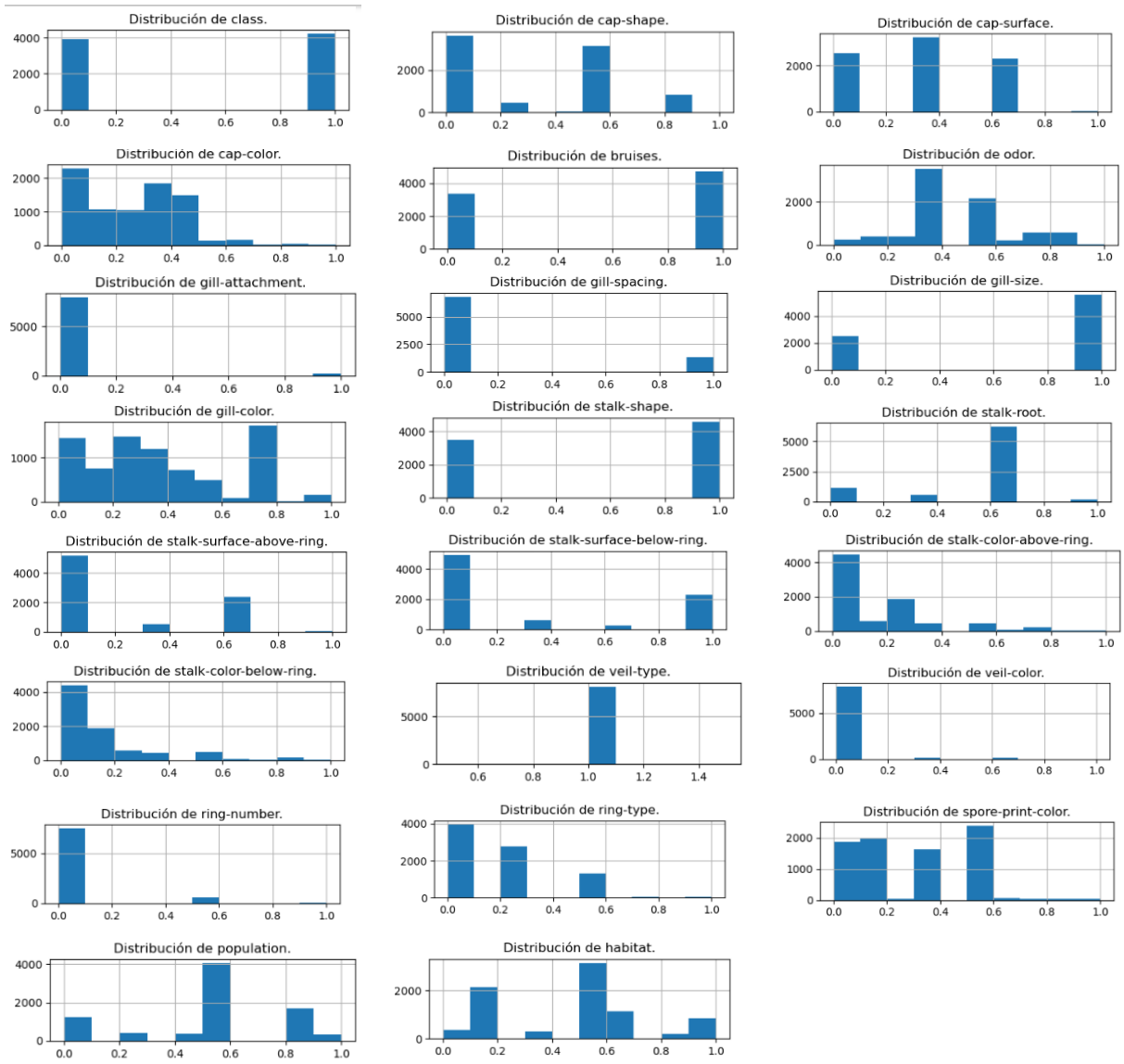


Ilustración 52.Distribución de los atributos de la base de datos.

Así bien, es importante mencionar la implementación de los métodos de imputación aleatoria y HotDeck en el atributo 'stalk-root' de la base de datos, esto con el fin de eliminar la condición de valores falntes en el atributo mencionado anteriormente. A continuación, se muestra la comparación entre la distribución del atributo antes y después de realizar las diferentes imputaciones.

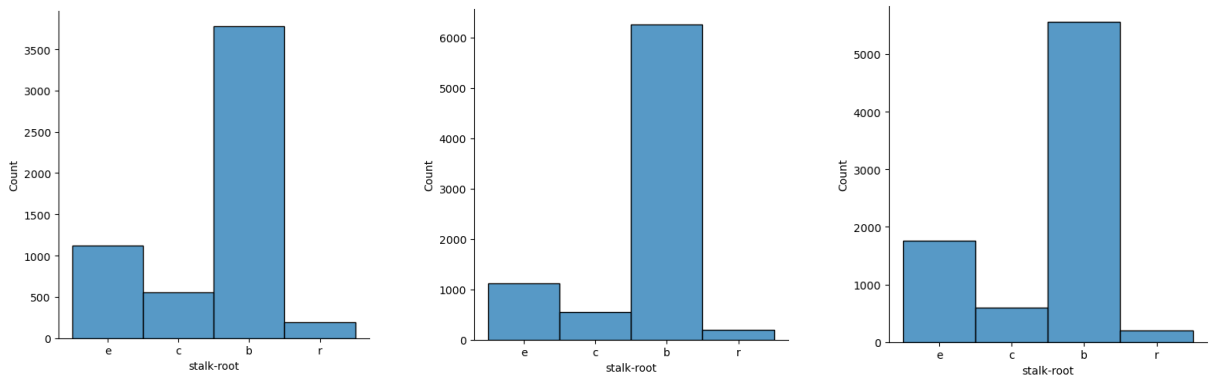


Ilustración 53. Comparación de la distribución del atributo stalk-root antes y despues de la imputación Aleatoria y HotDeck.

Posteriormente, se implementó la técnica de análisis de componentes principales (PCA) en la base de datos en cuestión, en donde se logró reducir la cantidad de atributos de 23 a 9, entre los atributos que contenían la mayor información se encuentran: cap-shape, cap-surface, cap-color, odor, gill-color, stalk-root, stalk-surface-above-ring, stalk-surface-below-ring y stalk-color-above-ring. Así bien, para los siguientes pasos se incluye a estos atributos seleccionados el atributo de decisión 'class'.

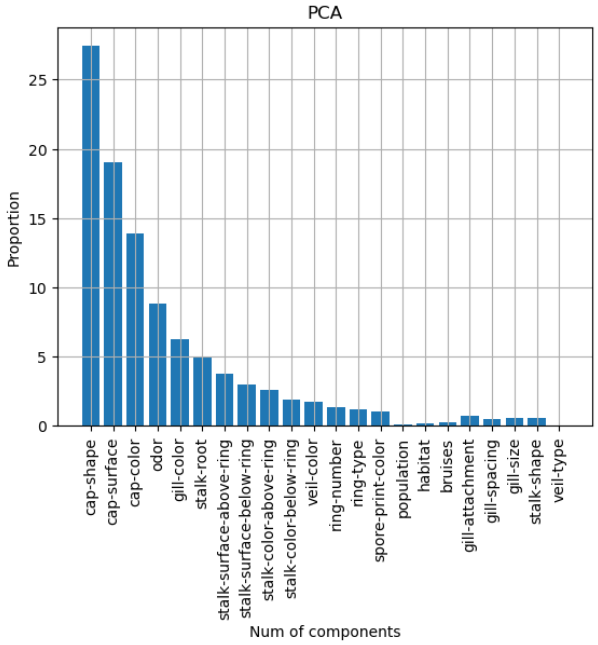
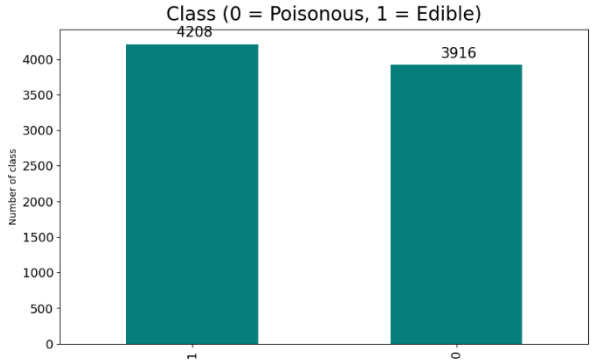


Ilustración 54. Análisis de Componentes Principales (PCA).

En la siguiente Ilustración se muestra la distribución del atributo de decisión: class, cuyas observaciones son: comestible(e) y venenoso(p), o bien, 1 y 0, respectivamente. A partir de la función Balance_clases se puede obtener el porcentaje de instancias para cada una de las clases del dataset, en donde, “e” tiene el 51.80%, mientras que a “p” le corresponde el porcentaje restante, el cual es 48.20% del total. Lo cual indica un muy ligero desbalance entre clases, tan ligero que se podría considerar que la base de datos se encuentra balanceada.




```
¿Cuál es tu atributo de decisión?:class
e    51.797144
p    48.202856
Name: class, dtype: float64
El dataset cuenta con 2 clases y son las siguientes: ['e', 'p']
```

Ilustración 55. Porcentaje de cada una de las clases: Venenoso y Comestible.

Por otra parte, se muestran las distribuciones de cada uno de los atributos correspondientes a cada una de las clases de la base datos en cuestión, las cuales son: Comestible(e) y Venenoso(p). Esto con el fin de conocer que observaciones tienen más afinidad a una de las clases en particular, como es el caso del atributo de 'odor', en donde la clase Comestible(e) le corresponden las observaciones almond(a), anise(l) y none(n), mientras que la clase Venenoso(p) le corresponden las observaciones pungent(p), none(n), foul(f), creosote(c), fishy(y), spicy(s) y musty(m). Así bien, es posible que las clases cuenten con las mismas observaciones en un atributo, como es el caso del atributo 'cap-surface', en donde la clase Comestible(e) y Venenoso(p) les corresponden las observaciones smooth(s), scaly(y) y fibrous(f).

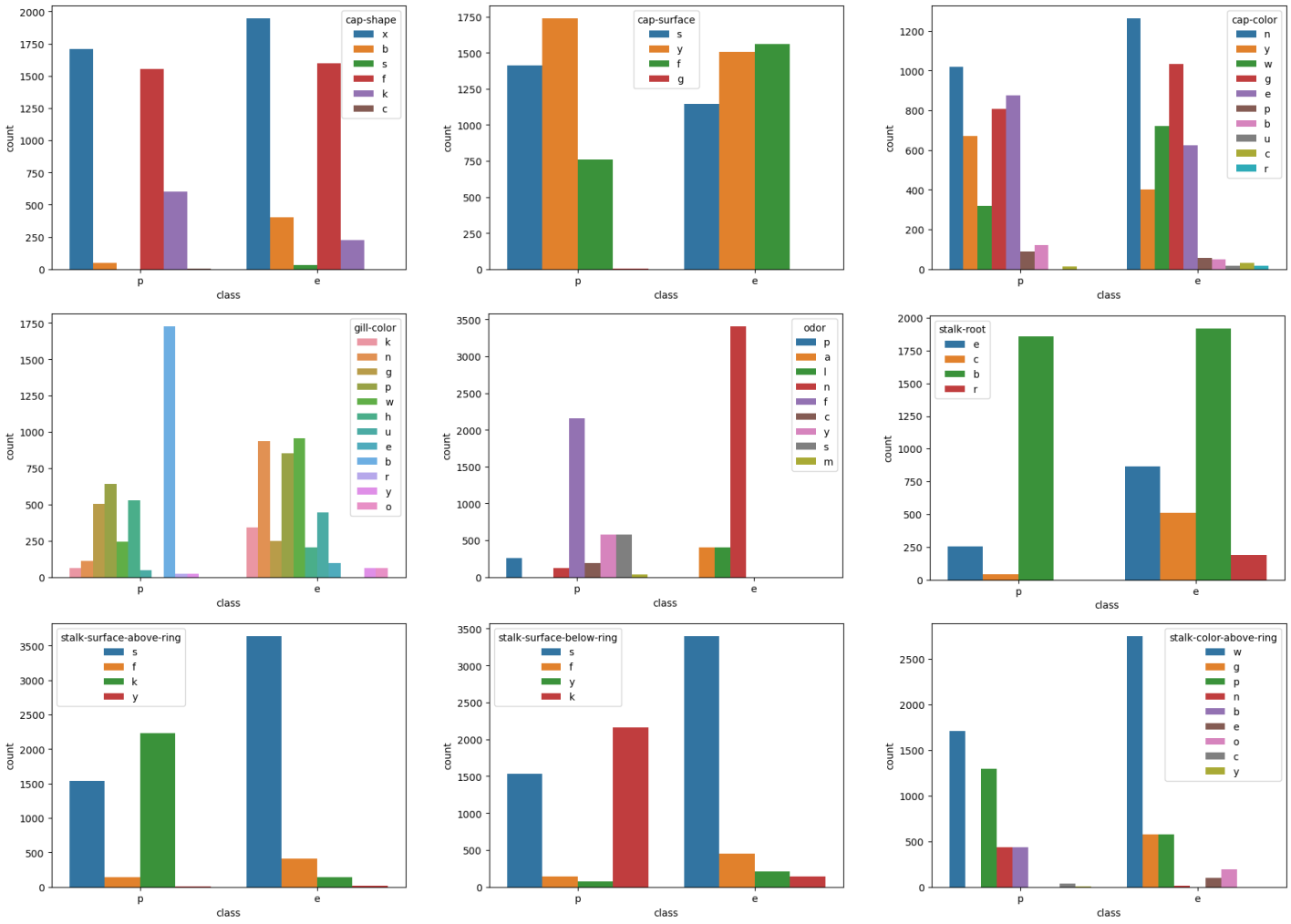


Ilustración 56. Distribución de atributos por clase.

Antes de continuar con la implementación del algoritmo de clasificación será necesario dividir el conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba. En la Tabla 1 se presentan el total de los datos para cada uno de los conjuntos, en donde se utilizó un k-fold de 5, un k-fold de 10 y validación lineal 80x20.

Tabla 1. División del conjunto de Entrenamiento y Prueba para un k-fold 5, k-fold 10 y validación lineal 80x20.

Total de datos	Conjunto de Entrenamiento	Conjunto de Prueba
Validación cruzada k-fold 5	8121	1624
Validación cruzada k-fold 10	8121	812
Validación lineal 80x20	6499	1625

A continuación, se procede en mostrar los resultados obtenidos en cada uno de los algoritmos implementados para este trabajo, los cuales son: KNN, Regresión Logística, Árbol de Decisión, Perceptrón Multicapa y Kmeans ++.

Tabla 2. División del conjunto de Entrenamiento y Prueba para un k-fold 5, k-fold 10 y validación lineal 80x20.

Algoritmos	Plan de pruebas
KNN	Clasificación con PCA con validación lineal 80x20 Clasificación sin PCA con validación lineal 80x20 Clasificación con PCA con validación cruzada k-fold 5 Clasificación con PCA con validación cruzada k-fold 10
Regresión Logística	Con PCA con validación lineal 80x20 Hiperparámetros: Épocas = 100, Alpha = 0.03, Tolerancia = 1e-3 Épocas = 100, Alpha = 0.3, Tolerancia = 1e-3 Épocas = 100, Alpha = 0.06, Tolerancia = 1e-3 Épocas = 100, Alpha = 0.6, Tolerancia = 1e-3
Árbol de Decisión	Con PCA con validación lineal 80x20 Hiperparámetros: criterion = ["gini","entropy"], max_depth = np.arange(1,11), max_features = [None, "auto", "sqrt", "log2"]
Perceptrón Multicapa	Con PCA con validación lineal 80x20 Hiperparámetros: learning_rate = 0.1, epocas_entrenamiento = 30000, neuronas_por_capa = [22,12, 6, 3, 2]
Kmeans ++	Con PCA con validación lineal 80x20

KNN

Como resultado de un barrido de valores de k para encontrar el valor, tal que, disminuya la tasa de error en la clasificación del algoritmo KNN después de un análisis de componentes principales, se obtuvieron las siguientes gráficas, en donde se puede observar el incremento del error cuando aumenta el número de k vecinos.

Clasificación con PCA con validación 80x20

En la Ilustración 57, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución de la validación 80x20.

```
100%|██████████| 6499/6499 [05:28<00:00, 19.80it/s]
Tiempo de entrenamiento:328.24896025657654
```

Ilustración 57. Tiempo de entrenamiento.

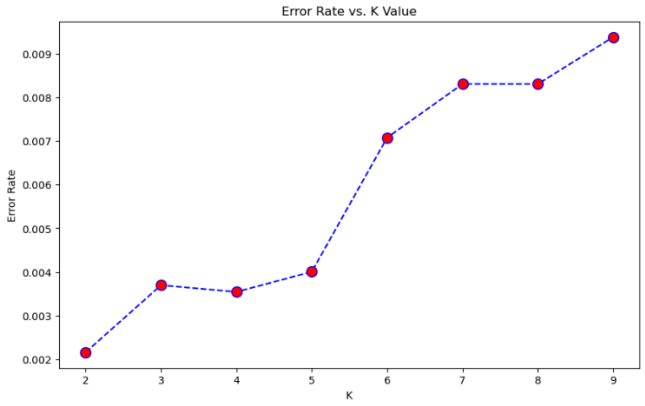


Ilustración 58. Visualización k mejor.

Una vez seleccionado el valor de k = 5, se continuó con la etapa de prueba del modelo, obteniendo los resultados mostrados en la Tabla 3.

Tabla 3. Métricas de evaluación del rendimiento del conjunto de entrenamiento y prueba.

	Conjunto de Entrenamiento	Conjunto de Prueba
Exactitud	0.99599	0.99815
Precisión	0.99675	0.99411
Sensitividad	0.99621	1.0
F1 Score	0.99648	0.99705

Clasificación sin PCA con validación 80x20

En la Ilustración 59, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución de la validación 80x20.

```
100%|██████████| 6499/6499 [21:47<00:00,  4.97it/s]
Tiempo de entrenamiento:1307.1597526073456
```

Ilustración 59. Tiempo de entrenamiento.

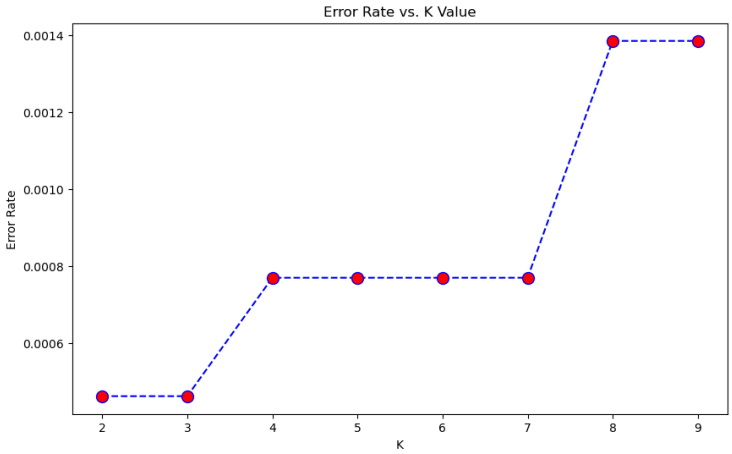


Ilustración 60. Visualización k mejor.

Una vez seleccionado el valor de k = 3, se continuó con la etapa de prueba del modelo, obteniendo los resultados mostrados en la Tabla 4.

Tabla 4. Métricas de evaluación del rendimiento del conjunto de entrenamiento y prueba.

	Conjunto de Entrenamiento	Conjunto de Prueba
Exactitud	0.99923	1.0
Precisión	1.0	1.0
Sensitividad	0.99864	1.0
F1 Score	0.99932	1.0

Clasificación con PCA con validación kfold 5

En la Ilustración 61, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución de la validación cruzada kfold 5.

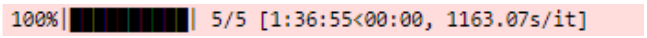


Ilustración 61. Tiempo de entrenamiento y prueba.

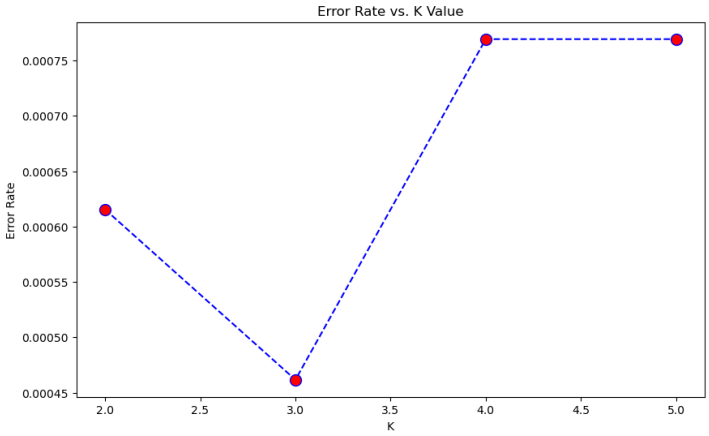


Ilustración 62. Visualización k mejor.

Una vez seleccionado el valor de k = 3, se continuó con la etapa de prueba del modelo, obteniendo los resultados mostrados en la Tabla 5.

Tabla 5. Métricas de evaluación del rendimiento del modelo con kfold5.

	Exactitud kfold-cross	Precisión kfold-cross	Sensitividad kfold-cross	F Score kfold-cross
Conjunto de Prueba	0.99433	0.98709	0.98447	0.98575

Clasificación con PCA con validación kfold 10

En la Ilustración 63, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución de la validación cruzada kfold 10.

```
100%|██████████| 10/10 [3:21:35<00:00, 1209.56s/it]
```

Ilustración 63. Tiempo de entrenamiento y prueba.

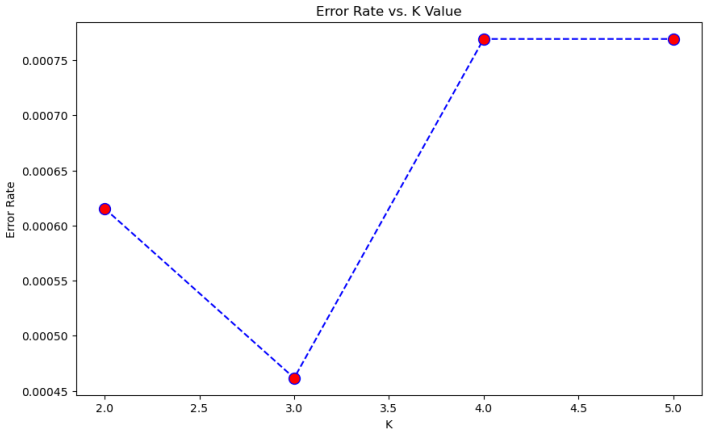


Ilustración 64. Visualización k mejor.

Una vez seleccionado el valor de k = 3, se continuó con la etapa de prueba del modelo, obteniendo los resultados mostrados en la Tabla 6.

Tabla 6. Métricas de evaluación del rendimiento del modelo con kfold 10.

	Exactitud kfold-cross	Precisión kfold-cross	Sensitividad kfold-cross	F Score kfold-cross
Conjunto de Prueba	0.98953	0.96972	0.96368	0.96604

Regresión Logística con PCA

A continuación, se pueden observar los parámetros propuestos para la implementación del modelo de Regresión Logística. Cabe mencionar que los valores iniciales de theta se obtuvieron de manera aleatoria.

Épocas = 100, Alpha = 0.03, Tolerancia = 1e-3

En la Ilustración 65, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento. Así bien, en esta ocasión el modelo no convergió antes de la época 100.

```
100%|██████████| 100/100 [00:00<00:00, 2032.99it/s]
```

Ilustración 65. Tiempo de entrenamiento del modelo de regresión logística.

En la Ilustración 66, se puede visualizar como se va modificando el valor de las funciones de costo con respecto al número de iteración en cuestión.

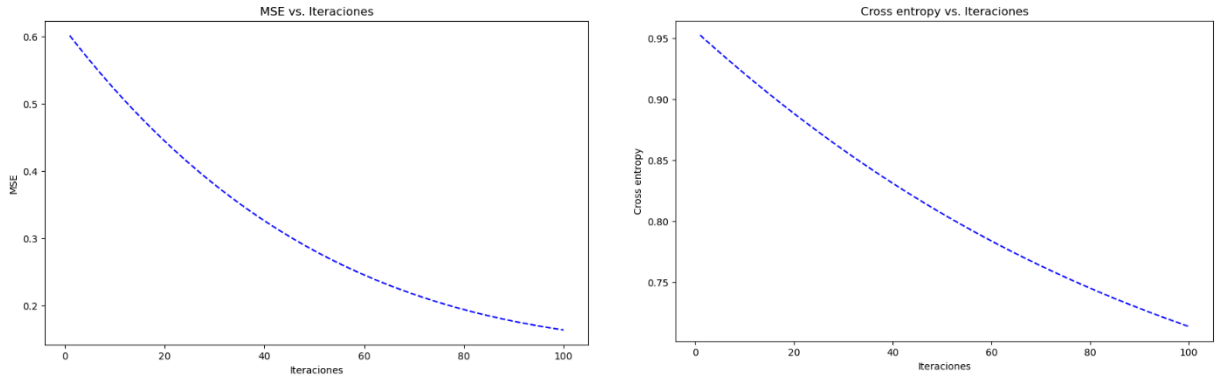


Ilustración 66. Funciones de costo vs Iteraciones.

Por otra parte, cabe mencionar que los valores obtenidos en la métrica de exactitud para el conjunto de prueba no supera el 50 por ciento. Mientras que los valores obtenidos en las funciones de costo MSE y Cross Entropy se encuentran cercanos al cero.

Tabla 7. Resultados obtenidos en la métrica de exactitud y las funciones de costo.

Exactitud	MSE	Cross Entropy
0.312	0.12217	0.46430

Épocas = 100, Alpha = 0.3, Tolerancia = 1e-3

En la Ilustración 67, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento. Así bien, en esta ocasión el modelo no convergió antes de la época 100.

```
100%|██████████| 100/100 [00:00<00:00, 2011.44it/s]
```

Ilustración 67. Tiempo de entrenamiento del modelo regresión logística.

En la Ilustración 68, se puede visualizar como se va modificando el valor de las funciones de costo con respecto al número de iteración en cuestión.

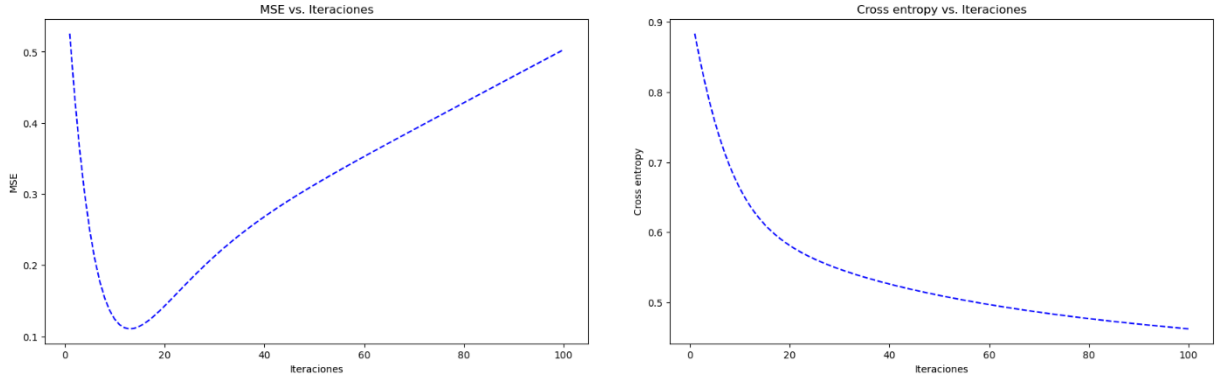


Ilustración 68. Funciones de costo vs Iteraciones.

Así mismo, los valores obtenidos en la métrica de exactitud para el conjunto de prueba no supera el 70 por ciento. Mientras que los valores obtenidos en las funciones de costo MSE y Cross Entropy se encuentran cercanos al cero.

Tabla 8.Resultados obtenidos en la métrica de exactitud y las funciones de costo.

Exactitud	MSE	Cross Entropy
0.67630	0.70861	0.36891

Épocas = 100, Alpha = 0.06, Tolerancia = 1e-3

En la Ilustración 69, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento. Así bien, en esta ocasión el modelo no convergió antes de la época 100.

```
100%|██████████| 100/100 [00:00<00:00, 1021.73it/s]
```

Ilustración 69. Tiempo de entrenamiento del modelo regresión logística.

En la Ilustración 70, se puede visualizar como va modificando el valor de la función de costo con respecto al número de iteración en cuestión, en donde se puede observar que al incrementar el número de iteraciones el valor de la función de costo va decreciendo.

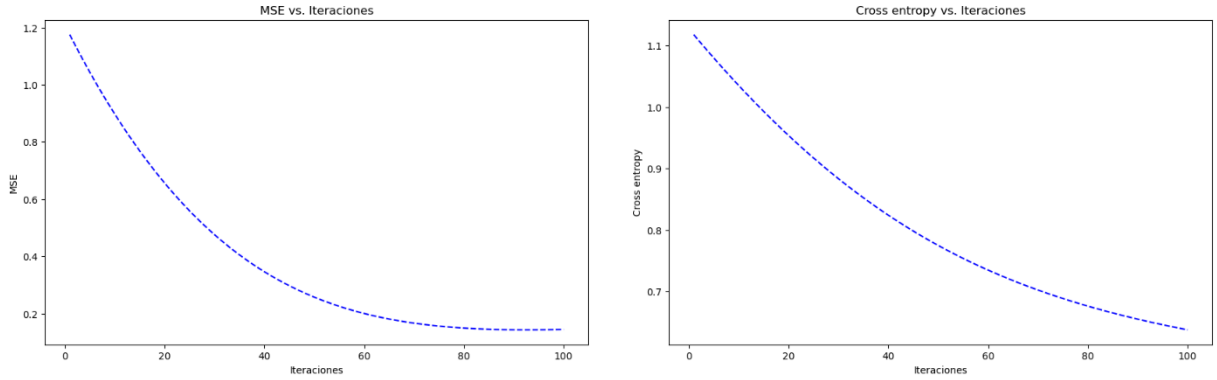


Ilustración 70. Funciones de costo vs Iteraciones.

Así mismo, los valores obtenidos en la métrica de exactitud para el conjunto de prueba no supera el 60 por ciento. Mientras que los valores obtenidos en las funciones de costo MSE y Cross Entropy se encuentran cercanos al cero.

Tabla 9. Métricas de evaluación del conjunto de prueba.

Exactitud	MSE	Cross Entropy
0.59384	0.15896	0.56647

Épocas = 100, Alpha = 0.6, Tolerancia = 1e-3

En la Ilustración 71, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento. Así bien, en esta ocasión el modelo no convergió antes de la época 100.

```
100%|██████████| 100/100 [00:00<00:00, 2165.95it/s]
```

Ilustración 71. Tiempo de entrenamiento del modelo de regresión logística.

En la Ilustración 72, se puede visualizar como se va modificando los valores de las funciones de costo con respecto al número de iteración en cuestión.

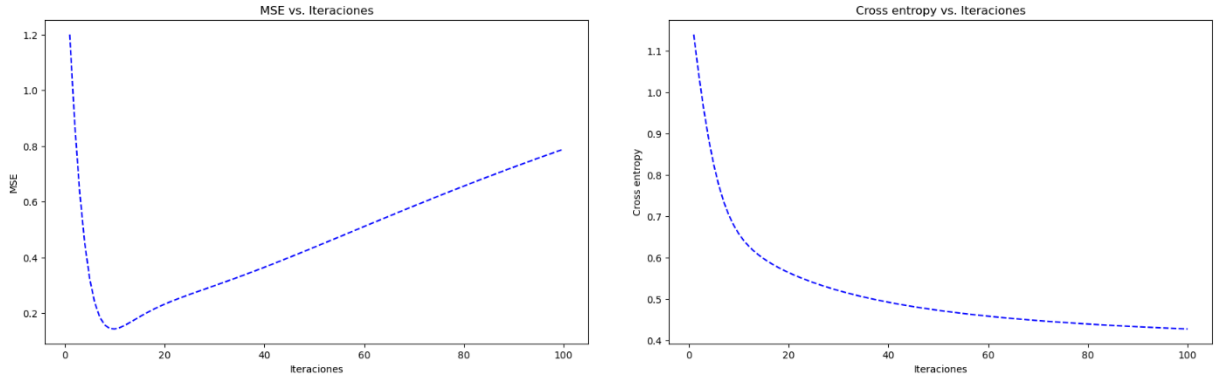


Ilustración 72. Funciones de costo vs Iteraciones.

Así mismo, los valores obtenidos en la métrica de exactitud para el conjunto de prueba, el cual no supera el 70 por ciento. Mientras que los valores obtenidos en la función de costo Cross Entropy sigue siendo muy cercana al cero.

Tabla 10. Resultados obtenidos en la métrica de exactitud y las funciones de costo.

Exactitud	MSE	Cross Entropy
0.66523	1.10343	0.32696

En la siguiente tabla, se puede realizar una comparación entre las diferentes pruebas realizadas de acuerdo con la variación en el valor del learning rate, en donde se puede observar que el mejor porcentaje de exactitud se obtuvo en el learning rate de 0.3 para el conjunto de prueba, el cual no logró superar el 70 por ciento.

Tabla 11. Comparación de los resultados obtenidos con learning rate = 0.03,0.3,0.06 y 0.6.

Learning rate	Exacitud	MSE	Cross Entropy
0.03	0.312	0.12217	0.46430
0.3	0.67630	0.70861	0.36891
0.06	0.59384	0.15896	0.56647
0.6	0.66523	1.10343	0.32696

Árbol de Decisión

A partir de la utilización de la librería de sklearn se obtuvo como resultado un árbol de decisión capaz de clasificar el conjunto de datos en cuestión, el cual tuvo un tiempo de entrenamiento de 0.00300 segundos. En la Ilustración 73 se puede observar la arquitectura del árbol de decisión obtenido, su nodo raíz, nodos intermedios y hojas.

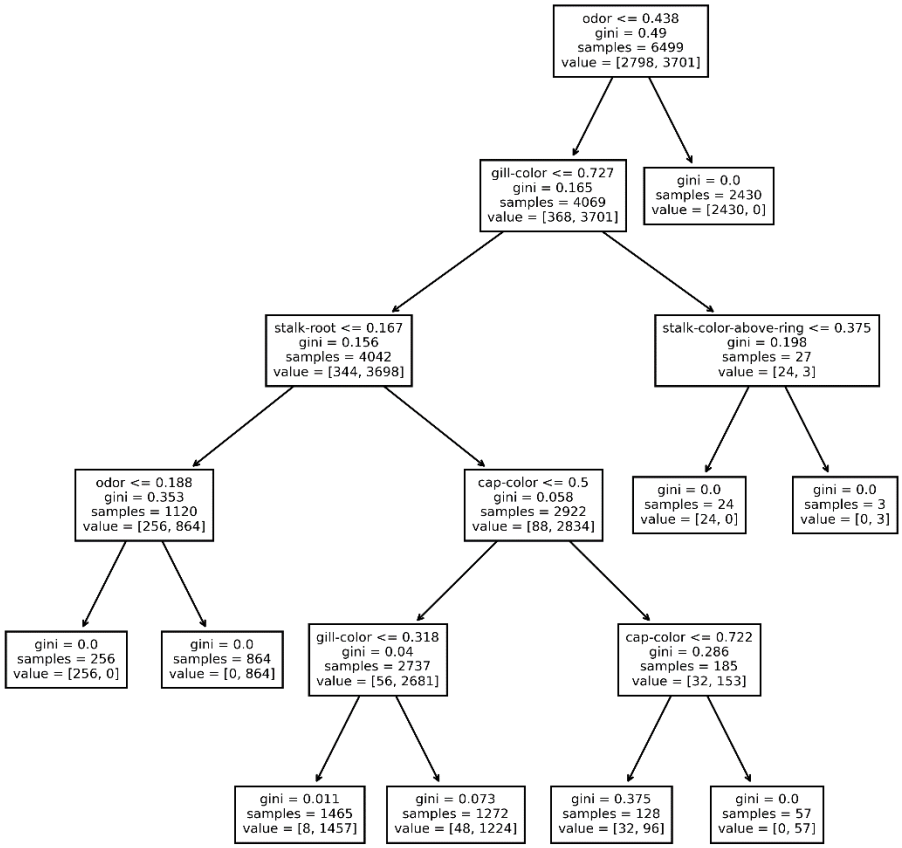


Ilustración 73. Arquitectura del árbol de decisión.

En la Tabla 12 se presentan los valores de las métricas de evaluación a fin de conocer el desempeño del modelo de árbol de decisión.

Tabla 12. Métricas de evaluación del rendimiento del conjunto de prueba.

	Exactitud	Precisión	Sensitividad	F1 Score
Conjunto de Prueba	0.99507	0.98446	1.0	0.99217

MLP

En la Ilustración 74, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento.

```
100%|██████████| 30000/30000 [01:53<00:00, 263.70it/s]
```

Ilustración 74. Tiempo de entrenamiento.

Mientras que, en la Ilustración 75 se muestra el historial de la función de pérdida durante el entrenamiento del perceptrón multicapa.

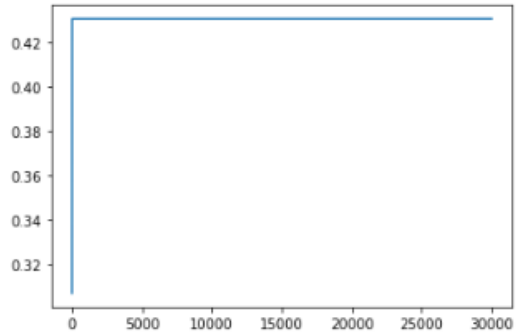


Ilustración 75. Visualización del comportamiento del modelo.

En la Tabla 13 se presentan los valores de la métrica de evaluación y el total de errores para el conjunto de entrenamiento y prueba, a fin de conocer el desempeño del modelo.

Tabla 13. Valores de precisión durante el entrenamiento y prueba.

	Conjunto de entrenamiento	Conjunto de Prueba
Precisión	0.56947	0.31200
Total de errores	2798	2798

KMeans ++

A continuación, se pueden observar los resultados que el algoritmo de K-Means ++ brindó, en donde se utilizaron los atributos con mayor relevancia en la base de datos: cap-shape y cap-surface para su implementación.

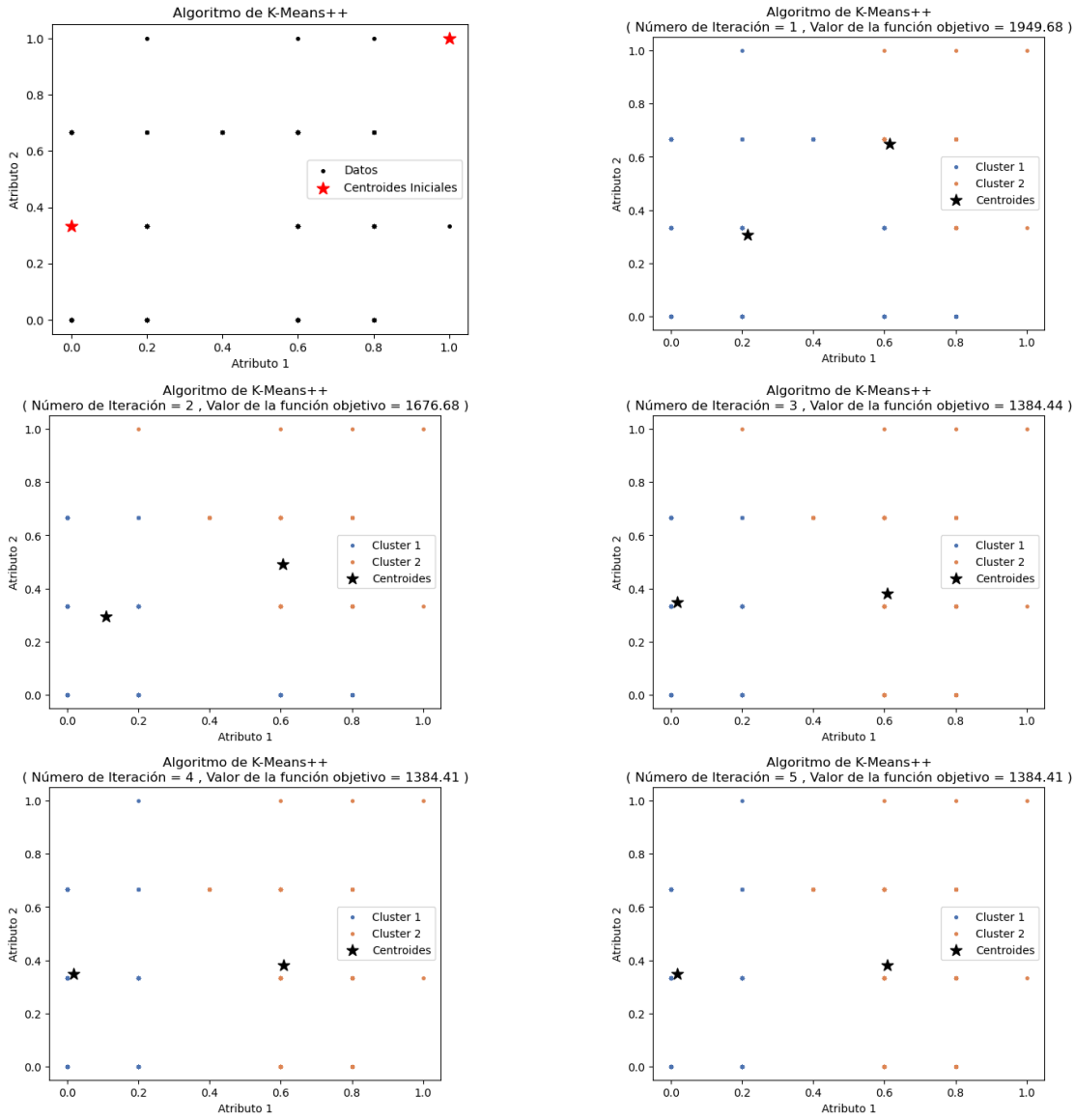


Ilustración 76. Número de iteración vs valor de la función objetivo.

En la Ilustración 76 se puede observar cómo se van reubicando los centroides, así como la reasignación de los datos con respecto a las nuevas ubicaciones de los centroides. Así bien, el tiempo de ejecución de la etapa de entrenamiento fue 6.7233150005 segundos.

En la Ilustración 77 se pueden observar las variaciones en la función objetivo entre cada iteración, en donde a partir de la iteración 3 el valor de la función objetivo varia por milésimas con respecto a las iteraciones 4 y 5.

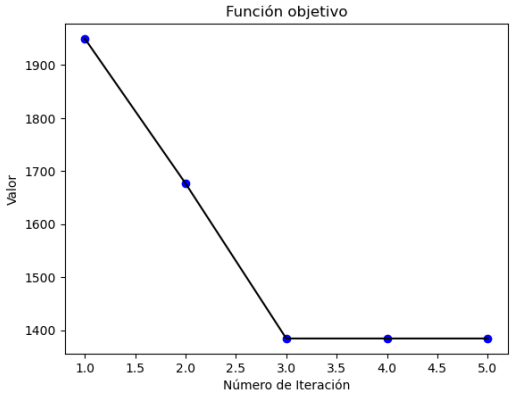


Ilustración 77. Comportamiento del modelo con cada iteración: gráfica de codo.

En la Ilustración se puede observar del lado derecho el resultado final posterior a la implementación de la técnica de K-Means++, en donde cada una de las agrupaciones corresponde a la predicción propuesta con respecto al atributo de decisión. Sin embargo, dichas agrupaciones no son del todo parecidas a las originales.

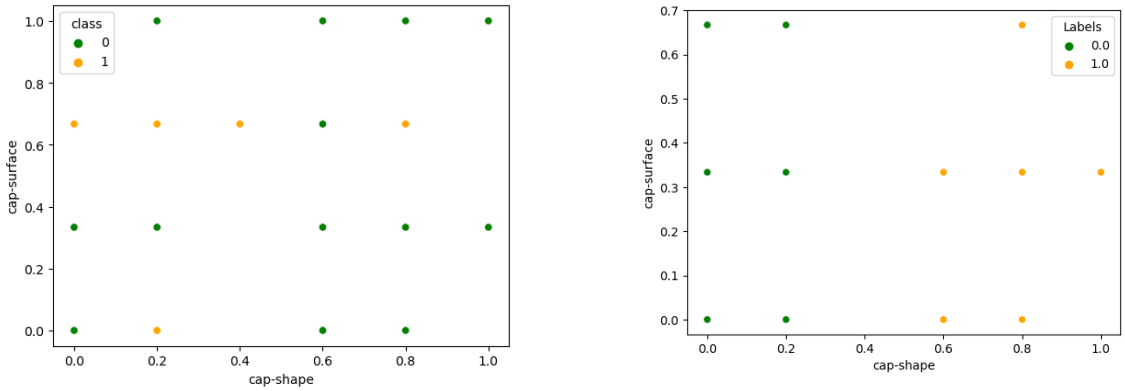


Ilustración 78.Comparación de la clasificación brindada del atributo de decisión de la base de datos y las agrupaciones propuestas por el algoritmo de K-Means ++.

En la Tabla 14 se presenta los resultados obtenidos en la etapa de prueba al utilizar la métrica de exactitud.

Tabla 14. Resultados en la etapa de Prueba del algoritmo de k-means ++

Orden de los grupos:	Aciertos	Porcentaje [%]:
0: 'Yes' , 1: 'No'	1182	72.73846

En la Tabla 15, se presenta la comparación de los resultados obtenidos entre los algoritmos implementados para este trabajo, en donde es posible observar que los algoritmos comparados: KNN vs árbol de decisión, obtienen mejores resultados en el KNN cuando no se utiliza PCA. Asimismo, se puede decir que el árbol de decisión presentó mejores resultados tanto en el porcentaje de exactitud, sensitividad y F1 score, cuando se utilizó KNN con kfold5. Mientras, que el árbol de decisión presentó mejores resultados en el porcentaje de exactitud, precisión, sensitividad y F1 score, cuando se utilizó KNN con kfold10, pero no así cuando se utiliza KNN sin PCA, dado que aquí dicho algoritmo obtiene el 1 en todas las métricas.

Tabla 15. Comparación de las métricas de evaluación del rendimiento del conjunto de prueba.

	Clasificación KNN con PCA	Clasificación KNN sin PCA	Clasificación KNN con PCA con kfold5	Clasificación KNN con PCA con kfold10	Árbol de Decisión Con PCA
Exactitud	0.99815	1.0	0.99433	0.98953	0.99507
Precisión	0.99411	1.0	0.98709	0.96972	0.98446
Sensitividad	1.0	1.0	0.98447	0.96368	1.0
F1 Score	0.99705	1.0	0.98575	0.96604	0.99217

Ahora bien, en la Tabla 16 se puede observar un mejor rendimiento por parte del algoritmo KNN comparando la métrica de exactitud contra el método de Kmeans++ y el método de Regresión Logística.

Tabla 16. Comparación de la métrica de exactitud con respecto al conjunto de prueba.

	Clasificación KNN con PCA	Clasificación KNN sin PCA	Clasificación KNN con PCA con kfold5	Clasificación KNN con PCA con kfold10	Kmeans++ con PCA	Alpha = 0.3 Épocas = 100 Tolerancia=1e-3
Exactitud	0.99815	1.0	0.99433	0.98953	0.72738	0.67630

Por último, en la Tabla 17 se muestran los valores de precisión obtenidos por parte del árbol de decisión y MLP, en donde MLP obtuvo el valor más bajo, mientras que por el contrario la arquitectura del árbol de decisión obtuvo un valor en la métrica de precisión muy cercano al 1.

Tabla 17. Comparación de la métrica de precisión del conjunto de prueba.

	MLP con PCA	Árbol de Decisión con PCA
Precisión	0.31200	0.98446

Conclusiones

En este programa generalizado inicialmente se generaron funciones que permitieran conocer el comportamiento de los datos mediante la implementación de algoritmos y modelos de clasificación, tales como KNN, K-means++, regresión logística, perceptrón multicapa y árboles de decisión, así como funciones que permiten obtener métricas estadísticas de los datos.

Aunque un algoritmo puede resultar mejor que otro dependiendo de las necesidades y el tipo de datos, hay ciertas propiedades de cada algoritmo que generalmente se presentan y conocen. Además, particularmente para la base de datos de Mushrooms, el haber implementado el análisis de componentes principales (PCA) resultó favorable, ya que para los resultados obtenidos en el conjunto de prueba para los algoritmos de KNN y Árbol de Decisión son muy cercanos al 1.0. Sin embargo, se presentaron mejores resultados en el algoritmo de KNN sin PCA dado que, los valores en las métricas de evaluación son de 1.0.

Por otro lado, gracias a esta práctica fue posible reconocer y entender las ventajas y desventajas de métodos como el de regresión logística, el cual es rápido de entrenar, fácil de entender, pero

sin embargo el comportamiento del algoritmo puede ser no optimo si se cuenta con valores atípicos. Asimismo, el algoritmo KNN entrena rápido, solo basta escoger un valor de k adecuado, pues el método se basa en la similitud de características, sin embargo, puede resultar costoso y lento computacionalmente hablando, al igual que los árboles de decisión, en donde incluso pueden surgir árboles tan complejos que sean difíciles de interpretar, aunque no fue el caso en este trabajo, dado que, se puede realizar modificaciones en su profundidad. Los algoritmos con resultados menos favorables fueron MLP, debido a las miles de combinaciones posibles para encontrar un mejor resultado, solo se exploraron unas cuantas, una de las desventajas del MLP; así como, la regresión logística, en donde también se deben probar y ajustar distintos parámetros, y no se encontraron los cuales dieran mejores valores en las métricas de evaluación; y por último, en K-means ++ , con el cual, a pesar de haber escogido el mejor k, los resultados no superaron al método de KNN o Árboles de decisión.

Referencias

[1] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[2] L. Laura-Ochoa, “Evaluation of Classification Algorithms using Cross Validation,” Ind. Innov. Infrastruct. Sustain. Cities Communities, pp. 24–26, 2019, doi: 10.18687/LACCEI2019.1.1.471.].

[3] M. Antonio and A. Fernández, Inteligencia artificial para programadores con prisa by Marco Antonio Aceves Fernández - Books on Google Play. Universo de Letras. [Online]. Available: https://play.google.com/store/books/details/Inteligencia_artificial_para_programadores_con_pri?id=ieFYEAAAQBAJ&hl=en_US&gl=US

[4] “8 algoritmos de agrupación en clústeres en el aprendizaje automático que todos los científicos de datos deben conocer.” [Online]. Available: https://www.freecodecamp.org/espanol/news/8-algoritmos-de-agrupacion-en-clusteres-en-el-aprendizaje-automatico-que-todos-los-cientificos-de-datos-deben-conocer/. [Accessed: 28-Mar-2022].

[5] “ML | K-means++ Algorithm - GeeksforGeeks.” [Online]. Available: https://www.geeksforgeeks.org/ml-k-means-algorithm/. [Accessed: 28-Mar-2022].

[6] “Algoritmos de Agrupamiento - 🤖 Aprende IA.” [Online]. Available: https://aprendeia.com/algoritmos-de-clustering-agrupamiento-aprendizaje-no-supervisado/. [Accessed: 28-Mar-2022].

[7] Husted, Alex. https://github.com/jalexander03/dsc-3-final-project-online-ds-ft-041519