Optativa IV. Machine Learning

**Alumno: Gerardo Hernández Nava**
**Correo electrónico: ghernandez100@alumnos.uaq.mx**

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

## Examen Final

---

### I. Introducción.

La base de datos de mushroom es una base de datos proporcionada por la Universidad de Irvine, California (*Mushroom Data Set*, 1987). 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.

### II. Objetivo

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

## III. Marco teórico.

### A. Análisis de componentes principales

El análisis de componentes principales es una técnica estadística con la cual se busca la reducción de dimensionalidad y la minimización de la pérdida de la información. Donde cada uno de los atributos que se mantienen toman el nombre de componentes principales.

### B. 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 totalmente o localmente conectado (IBM, n.d.).

### C. Algoritmo de K vecinos más cercanos

El algoritmo de KNN es un algoritmo que pertenece a los algoritmos de aprendizaje supervisado. 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 màs 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 (IBM, 2021).

### D. 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 (Martinez, 2020).

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 (Martinez, 2020).

## IV. Metodología

1. Se importaron las librerías necesarias para poder implementar el algoritmo, como pandas, math, numpy, matplot, seaborn, tqdm, operator, entre otras.
2. Se comienzan a declarar las funciones en las cuales se va a basar nuestro algoritmo. Las funciones fueron las siguientes:
    a. Funciones para PCA
        i. *covarianza,* como lo dice su nombre es para obtener la covarianza, recibe como parámetros dos vectores, donde cada uno corresponde a un atributo de nuestra base de datos. Se calcula la covarianza entre ellos, mediante la siguiente fórmula, y se regresa el valor de covarianza entre ellos.

$$cov_{xy} = \frac{\Sigma_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

        ii. *matriz_covarianza* hace uso de la función descrita anteriormente, mandando poco a poco los atributos para que de esta forma quede como resultado una matriz cuadrada que nos indica la covarianza que existe entre cada uno de los atributos.
    b. Funciones para KNN.
        i. *distancia_euclidiana,* es utilizada para conocer la distancia euclidiana de un punto dado hacia todos sus vecinos y hacer un posterior análisis con la función *pertenencia*.
        ii. *pertenencia,* indica a qué clase se asocia el punto analizado, esta decisión se toma con la clase a cual pertenece la mayoría de los vecinos más cercanos según la distancia euclidiana.
        iii. *entrenamiento_KNN,* primero se calcula el número de clases existentes en el dataset, el número de

instancias y se crea una matriz donde se guardaran las exactitudes dado un número n de vecinos. La matriz se va llenando mediante dos ciclos for anidados, el primero de ellos recorre el número de vecinos que se desean tomar en cuenta y el segundo cada unos de los puntos. Ya que se terminó de llenar la matriz se calcula máxima exactitud para poder determinar el mejor número de vecinos a tomar en cuenta. Está función regresa todas las exactitudes calculadas, la mejor y el número de vecinos que se tomaron en cuenta.

iv. Para finalizar con las funciones se realizó una función para clasificar los datos de prueba. Donde se calculan las distancias de cada uno de los puntos y se le asigna una etiqueta según los vecinos determinados como los mejores en el entrenamiento.

c. Funciones para ID3
   i. *contar_clases*, como lo dice su nombre esta función cuenta el número de instancias que corresponden a cada clase. Como entrada la función recibe la columna de atributo de decisión de la base de datos a analizar, o si se quiere ver de otra forma 'y', y las observaciones que se tienen en el atributo de decisión o aquellas que se desean contar. Y como salida, la función retorna una arreglo con el número de observaciones contadas.
   ii. *calcular_entropía*, está función arroja como resultado la entropía de decisión del sistema que se esté analizando
   iii. *ganancia_atributo*, se calcula la ganancia de cada atributo respecto al sistema tomando en cuenta la entropía de decisión.
   iv. *nodo_raiz*, dicha función selecciona el atributo de la base de datos que tenga mayor ganancia respecto a los demás, para que de esta forma sea el nodo raíz del árbol que se está formando o sea el siguiente. Retorna

el nombre del nodo (atributo) y la entropía por la que fue seleccionado.

v. *nodos_intermedios*, en esta función se generan sub-bases de datos, esto es que la función recibe como parámetros, la base de datos de atributos con los cuales se debe realizar la clasificación, la columna de atributo de decisión, el atributo bajo el cual se va a realizar el análisis y las observaciones que contiene dicho atributo; para después encontrar todos aquellos índices en los cuales se puede encontrar la primera observación del atributo y una vez que se tienen los índices se extraen de la base de datos original y se haría la primer sub-base de datos y también se extrae de la columna del atributo de decisión. Esto se repite hasta terminar con todas las observaciones.

vi. *fin_rama*, ayuda a determinar si la rama que se generó en el árbol es final o mejor dicho termina en una hoja, se dice que es una hoja cuando todas las instancias que llegan hasta ese punto pertenecen a la misma clase de forma que se puede tomar una decisión. Retorna un cero o uno, cero cuando no la rama no es final y el árbol debe continuar y uno cuando la rama termina en una hoja,

vii. *arbol_id3*, es una función de recursividad la cual ayudará a ir formando la estructura del árbol id3 mandando a llamar todas las funciones antes descritas y así misma cada que la rama del árbol no termine en una hoja, es decir, se haya tomado una decisión. Retorna la estructura del árbol en un diccionario.

d. Funciones para el perceptrón multicapa

*i.* *capa_neuronal,* es un 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.

   ii. *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.

   iii. *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.

   iv. *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.

e. Funciones para el tratamiento de la base de datos.

   *i.* *split_80_20,* como lo dice su nombre ayuda a dividir el conjunto de datos en dos, la primera parte corresponde a un 80% de los datos de que se tienen y serán utilizados para el entrenamiento del modelo, el restante 20% se asigna para la prueba. Pero antes de dividir el conjunto de datos se crea una vector con los índices de los datos y se aleatoriza, posteriormente se divide este arreglo y para finalizar se extraen los datos de X y Y según los indicen que se tengan en cada arreglo. La función recibe los datos X y sus etiquetas Y.

   *ii.* *split_80_20_df,* hace lo mismo que la función anterior pero está enfocada para dividir dataframes.

   *iii.* *normalizar,* como lo dice su nombre normaliza la base de datos haciendo que la información quede con valores entre ceros y uno.

f. Funciones para calcular las cifras de mérito.

   *i.* *exactitud*

   *ii.* *precision*

iii. *precision_mlp*
iv. *sensitividad*
v. *especificidad*
vi. *puntaje_F1*
vii. *MSE*

3. Una vez teniendo todas la funciones definidas se importó la base de datos con la cual se trabajó mediante pandas y la función read_csv.
4. Se analizó la base de datos, para saber sus dimensiones, atributos, instancias, observaciones por atributos y si habían valores faltantes.
5. Como se encontraron datos faltantes se procede a calcular el número de datos faltantes para saber si el atributo puede permanecer en la base de datos realizando una imputación o debe ser eliminada.
6. Como los datos son los suficientes para poder realizar una imputación, se procede a realizar dos tipos de imputación, el primero de ellos es mediante el algoritmo de KNN y la segunda mediante la moda por clase.
7. Una vez realizada la imputación se procedió a realizar un análisis de componentes principales para reducir la dimensionalidad y trabajar con los atributos que proporcionen mayor información.
8. Se dividen los datos en conjuntos de prueba y de entrenamiento.
9. Se procede a declarar las funciones de activación y de coste para el perceptrón multicapa, así como la tasa de aprendizaje, las épocas de entrenamiento y las neuronas por capa.
10. Las clases de la base de datos se traduce a categorical para poder tener dos neuronas de salida y se compila la red neuronal.
11. Se entrena la red neuronal y se evalúa.
12. Se procede a trabajar con el algoritmo de KNN, llamando a llamar la función *entrenamiento_KNN* y una vez que se tiene el mejor número de vecinos se clasifica la base de datos con la función *clasificar_KNN*.
13. Para finalizar con este algoritmo se evalúa con todas las cifras de mérito con las que se cuenta.
14. Por último se intenta generar un árbol de decisión para la base de datos presente.

15. Como primer paso para este algoritmo se obtienen todas las clases de la base de datos.
16. Posteriormente se declara una variable tipo diccionario vicio donde se estructurará el árbol decisión y se manda a llamar la función *arbol_id3.*

## V. Resultados

Tras un análisis se puede determinar que la base de datos cuenta con un total de 22 atributos más uno de decisión. Cada atributo cuenta con un número diferente de observaciones.

Tabla 1. Primeras cinco renglones de la base de datos, donde se pueden observar algunos de los atributos, observaciones y clases que contiene.

| | 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 |

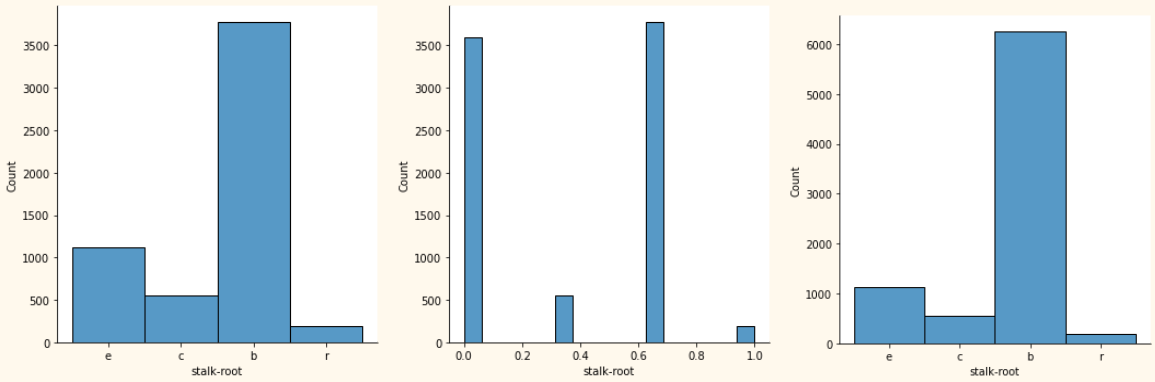
Ya que la base de datos es amplia y no se pueden ver todas las columnas ni sus observaciones se imprimieron cada una de las columnas con sus determinadas observaciones, de esa forma también se podía corroborar que no existieran datos faltantes por codificación.

```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
```

```
{'bruises': array(['t', 'f'], dtype=object),
 'cap-color': array(['n', 'y', 'w', 'g', 'e', 'p', 'b', 'u', 'c', 'r'], dtype=object),
 'cap-shape': array(['x', 'b', 's', 'f', 'k', 'c'], dtype=object),
 'cap-surface': array(['s', 'y', 'f', 'g'], dtype=object),
 'class': array(['p', 'e'], dtype=object),
 'gill-attachment': array(['f', 'a'], dtype=object),
 'gill-color': array(['k', 'n', 'g', 'p', 'w', 'h', 'u', 'e', 'b', 'r', 'y', 'o'],
       dtype=object),
 'gill-size': array(['n', 'b'], dtype=object),
 'gill-spacing': array(['c', 'w'], dtype=object),
 'habitat': array(['u', 'g', 'm', 'd', 'p', 'w', 'l'], dtype=object),
 'odor': array(['p', 'a', 'l', 'n', 'f', 'c', 'y', 's', 'm'], dtype=object),
 'population': array(['s', 'n', 'a', 'v', 'y', 'c'], dtype=object),
 'ring-number': array(['o', 't', 'n'], dtype=object),
 'ring-type': array(['p', 'e', 'l', 'f', 'n'], dtype=object),
 'spore-print-color': array(['k', 'n', 'u', 'h', 'w', 'r', 'o', 'y', 'b'], dtype=object),
 'stalk-color-above-ring': array(['w', 'g', 'p', 'n', 'b', 'e', 'o', 'c', 'y'], dtype=object),
 'stalk-color-below-ring': array(['w', 'p', 'g', 'b', 'n', 'e', 'y', 'o', 'c'], dtype=object),
 'stalk-root': array(['e', 'c', 'b', 'r', '?'], dtype=object),
 'stalk-shape': array(['e', 't'], dtype=object),
 'stalk-surface-above-ring': array(['s', 'f', 'k', 'y'], dtype=object),
 'stalk-surface-below-ring': array(['s', 'f', 'y', 'k'], dtype=object),
 'veil-color': array(['w', 'n', 'o', 'y'], dtype=object)}
```

Figura 1. En la figura de la izquierda se pueden observar el número de datos faltantes indicados con Nan encontrados en la base de datos. Mientras que en la figura de la derecha se pueden observar en el atributo stalk-root datos faltantes codificados.

Como la base de datos contenía datos faltantes fue necesario desplegar la distribución del atributo incompleto para poder determinar cómo había afectado la imputación al atributo.




a) b) c)

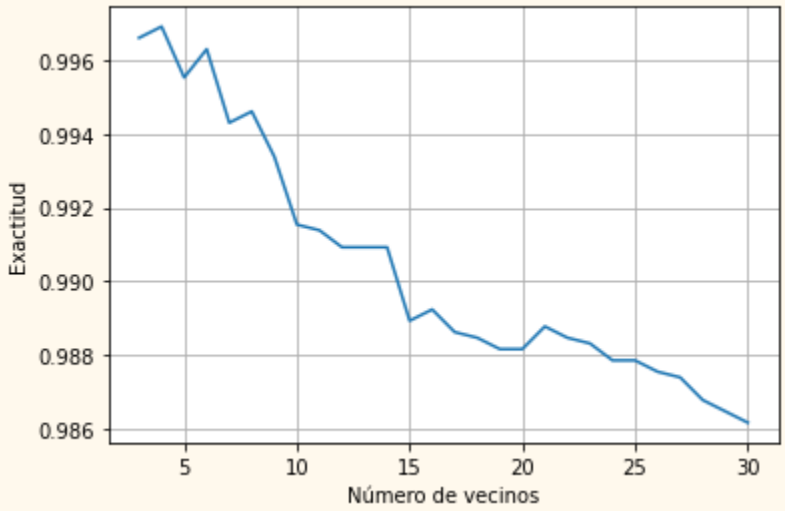
Figura 2. Distribución del atributo antes y después de la imputación. La figura a) corresponde a la distribución original del atributo, es decir, sin haber realizado la imputación. La figura b) corresponde a la distribución final del atributo mediante el algoritmo de KNN. La figura c) corresponde a la distribución una vez aplicada la imputación por moda de clase.

En la Figura 3 se puede observar el resultado de aplicar el algoritmo de PCA a nuestra base de datos, el cual nos ayudó a reducir la dimensionalidad de la base de datos en un 55%.

```
cap-shape: 27.33131629439478
cap-surface: 19.116313398009613
cap-color: 13.811856127134503
bruises: 8.944489167827191
odor: 6.278821265690815
gill-attachment: 4.960805125956759
gill-spacing: 3.73645496039574
gill-size: 2.8141949653051843
gill-color: 2.6157152649091286
stalk-shape: 1.8624035422528316
stalk-root: 1.7419561370568812
stalk-surface-above-ring: 1.392171726558767
stalk-surface-below-ring: 1.2151064285251587
stalk-color-above-ring: 1.1125928392614262
stalk-color-below-ring: 0.07656254940670525
veil-color: 0.21498303357773432
ring-number: 0.282237018154296
ring-type: 0.7879038696485183
spore-print-color: 0.47117519142587877
population: 0.6488169142540722
habitat: 0.5841241802540197
```

Figura 3. Atributos de la base de datos con su correspondiente aporte de información para la toma de decisión en su clasificación.

En la siguiente gráfica se podrá observar la superioridad del algoritmo de KNN cuando trabaja con cuatro vecinos que con un número superior o inferior.




Gráfica 1. Exactitud vs número de vecinos tomados en cuenta para tomar una decisión en la base de datos.

A continuación, se muestran los resultados obtenidos en el entrenamiento y prueba de los algoritmos con la base de datos mushroom.

Tabla 3. En esta tabla se pueden comparar los diferentes rendimientos de los algoritmos al intentar clasificar la base de datos de mushroom.

| Algoritmo | Cifra de mérito | Entrenamiento | Prueba |
|---|---|---|---|
| Perceptrón multicapa | Exactitud | – | – |
| | Precisión | 69.04 % | 66.74 % |
| | Sensitividad | – | – |
| | Especificidad | – | – |
| | Puntaje F1 | – | – |
| | Tiempo de ejecución | 289.162 segundos | 0.24 segundos |
| KNN | Exactitud | 99.69 % | 99.32 % |
| | Precisión | 99.55 % | 99.03 % |
| | Sensitividad | 99.85 % | 99.63 % |
| | Especificidad | 99.51 % | 98.99 % |

| | | | |
|---|---|---|---|
| | Puntaje F1 | 99.70 % | 99.33 % |
| | Tiempo de ejecución | 7264.729 segundos | 68.392 segundos |
| ID3 | Exactitud | – | – |
| | Precisión | – | – |
| | Sensitividad | – | – |
| | Especificidad | – | – |
| | Puntaje F1 | – | – |
| | Tiempo de ejecución | – | – |

## VI. Conclusiones

Por los resultados obtenidos se puede ver claramente que el algoritmo de KNN es el algoritmo que presenta un mejor rendimiento al clasificar la base de datos. Cabe mencionar que si el algoritmo de árbol de decisión fuera más robusto, como contener criterios de paro podría haberse probado en la base de datos y superar las cifras reportadas para el perceptrón multicapa.

## VII. Referencias


Martinez, J. (2020, September 19). *Árboles de Decisión con ejemplos en Python*. IArtificial.net. Retrieved June 12, 2022, from https://www.iartificial.net/arboles-de-decision-con-ejemplos-en-python/

Rahman, T. (2021, March 27). *Step by Step Decision Tree: ID3 Algorithm From Scratch in Python [No Fancy Library]*. Medium. Retrieved June 12, 2022, from https://medium.com/geekculture/step-by-step-decision-tree-id3-algorithm-from-scratch-in-python-no-fancy-library-4822bbfdd88f