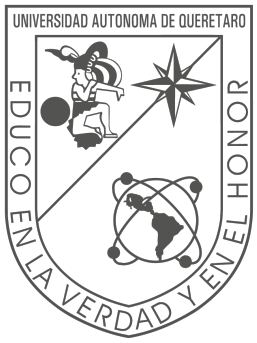
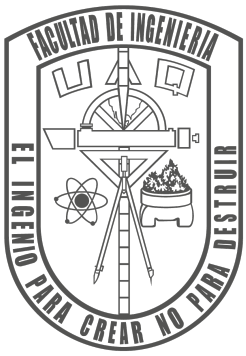
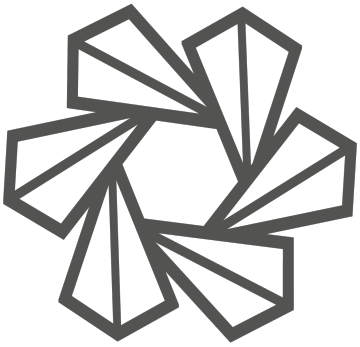
Universidad Autónoma de Querétaro

Facultad de Ingeniería
División de Investigación y Posgrado







Examen 2

Clasificación de hongos

Maestría en Ciencias en Inteligencia Artificial
Optativa de especialidad IV - Machine Learning

Aldo Cervantes Marquez
Expediente: 262775
Profesor: Dr. Marco Antonio Aceves Fernández

Santiago de Querétaro, Querétaro, México
Semestre 2023-1
23 de Junio de 2023

Índice

1. Objetivo

Esta práctica tiene como objetivo el de analizar una bases de datos de hongos y aplicar métodos de agrupamiento, con el fin de observar tendencias o comportamientos en los grupos involucrados para poder realizar una toma de decisiones y evaluar el método aplicado mediante las métricas correspondientes.

2. Introducción

La práctica consiste en obtener una base de datos tabulares de clasificación de hongos con base a sus características físicas, por lo que la base de datos está etiquetada, se aplicarán métodos de aprendizaje máquina basados en aprendizaje supervisado con el fin de poder predecir la categoría de un conjunto de datos que corresponden a los hongos. En este caso se realizará el método de Knn y arboles de decisión (ID3).

3. Marco Teórico

Para la realización de la práctica fueron necesarios los siguientes conceptos, los cuales fueron obtenidos principalmente de [1].

3.1. Algoritmo Knn

El algoritmo de *Knn* es un algoritmo de clasificación categórica supervisado que permite realizar una predicción de las clases [2]. Para realizar este algoritmo es necesario tener una base de datos etiquetada. Esta base de datos será dividida según sea conveniente en un periodo de entrenamiento y prueba, donde el entrenamiento será modelado en el conjunto de datos y estos serán la referencia para los datos de prueba [3]. Posteriormente se evaluará un punto de prueba y se obtendrán las distancias con respecto a todos los puntos de entrenamiento, agrupando los n más cercanos [4]. Para posteriormente elegir por medio de un voto mayoritario o plural el que más frecuencia de clase tenga, como se muestra en la Figura 1.

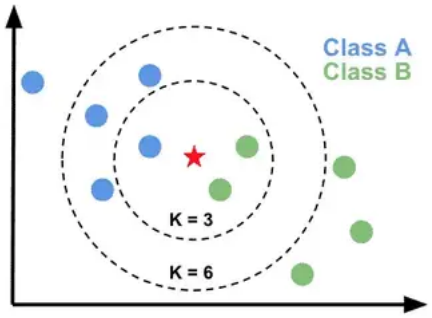


Figura 1: Knn visualización (obtenido de [3]).

3.1.1. Métodos de distancia entre puntos

Distancia euclidiana

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1}$$

Distancia Manhattan

$$d(x,y) = \sum_{i=1}^{n}|x_i - y_i| \tag{2}$$

Distancia Chebyshev

$$d(x,y) = \max_{i=1,2,3,\cdots,n}|x_i - y_i| \tag{3}$$

Distancia coseno

$$d(x,y) = \arccos\left(\frac{x \cdot y}{\|x\|\,\|y\|}\right) \tag{4}$$

Distancia euclidiana normalizada

$$d(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)} \tag{5}$$

Donde S^{-1} es una matriz con $\frac{1}{\sigma}$ en la diagonal

Distancia Mahalanobis

$$d(x,y) = \sqrt{(x-y)^T V^{-1}(x-y)} \tag{6}$$

Donde V es la matriz de varianzas y covarianzas.

3.2. Algoritmo de arboles de decisión ID3

Los algoritmos basados en arboles de decisión se utilizan tanto para clasificación como para regresión, teniendo como ventaja de que son fáciles de interpretar y que se manejan matemáticamente con nodos y ramas, donde cada nodo es un atributo y cada rama una decisión (regla).

En este caso el algoritmo de árbol de decisión ID3 es utilizado y se hace realizando los siguientes pasos:

1. Se inicia el árbol con el nodo raíz, del conjunto que contiene los datos completos.
2. Encontrar el mejor atributo en el conjunto de datos utilizando la medida de selección de atributos (Ganancia de información)
3. Dividir el conjunto original de subconjuntos que contengan valores posibles para los mejores atributos.
4. Generar el nodo del árbol de decisión, que contiene el mejor atributo.
5. Crear recursivamente nuevos árboles de decisión utilizando los subconjuntos del conjunto de datos creado en el paso 3.

6. Continuar con el proceso hasta llegar a una etapa en la que no pueda clasificar más los nodos y llame al nodo final como nodo hoja

Matemáticamente hablando se obtienen los siguientes pasos:

- Calcular entropía, esto se hace mediante la formula:

$$E(S) = \sum -P(I)\log_2(P(I)) \tag{7}$$

  Donde se definen las probabilidades por cada decisión (clases de salida S) y se suman.

- Definir nodo raíz. Hay que buscar el atributo que mejor clasifica los datos de entrenamiento, el que tenga mayor valor de ganancia de información.

$$G(S, A_i) = E(S) - \sum [p(S|A_i)E(S|A_i)] \tag{8}$$

  En donde se observa la entropía de la salida y la probabilidad dada la salida y la observación categórica del atributo A_i, esto se hace para cada observación del atributo

- Definir las observaciones en función de los atributos que más aporten al problema (el que tenga la mayor ganancia $G(S, A_i)$).
  - Realizar nuevas ganancias a partir de cada nodo e ir bajando, lo que implica que S será el atributo del nivel anterior.

3.3. Imputación de datos por moda de clases

La manipulación de datos faltantes en atributos se puede manejar con este método el cual consiste en aprovechar el etiquetado de los datos, empezando por dividir los datos en cada categoría y encontrar la moda de dicho atributo en cada categoría y sustituir los valores faltantes con dicho valor. Esto es más recomendable aplicarlo en variables categóricas.

$$V(C_i) = M_o(C_i) \tag{9}$$

En donde se explica que el valor faltante en la categoria C_i será sustituido por la moda de los datos no faltantes en el atributo de C_i

3.4. Métricas de evaluación

Para poder medir el desempeño del modelo knn e $ID3$ se tiene que partir de una matriz de confusión, la cual nos permite tener una observabilidad del comportamiento del modelo y se puede describir como la Figura 2

A partir de esta matriz podemos obtener las métricas:

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 2: Matriz de confusión para datos categóricos clasificados.

- **Exactitud**: Explica el desempeño total del modelo en cuanto a sus aciertos obtenidos.

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

- **Precisión**: explica la precisión de las predicciones positivas.

$$Pr = \frac{TP}{TP + FP} \tag{11}$$

- **Sensibilidad (recall)**: Indica la cobertura de las muestras positivas.

$$Sens = \frac{TP}{TP + FN} \tag{12}$$

- **F1 score**: Calcula la media armónica de la precisión y sensibilidad (recall) de forma que se enfatiza el valor más bajo entre ambas.

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{13}$$

3.5. Stratifield k-fold (k-fold estratificado) STF

Partiendo del concepto de k-fold, en donde la base de datos se divide en k segmentos, y se prueba cada segmento como prueba y las demás como entrenamiento. Se observa un problema con la diferencia de los estados aleatorios al momento de dividir los segmentos de datos. Obteniendo inconsistencia en la exactitud del modelo debido a los desbalances porcentuales de las clases (véase Figura 3).

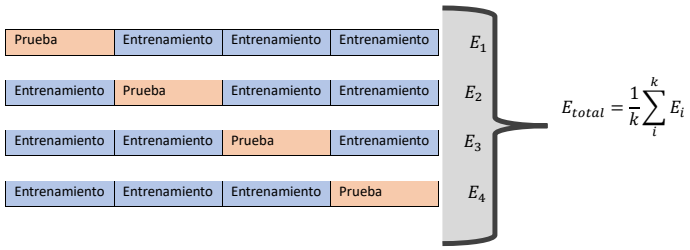


Figura 3: Estructura de algoritmos basados en k-folds.

Por lo que con k-fold estratificado se propone realizar un balance porcentual de las clases [5]. Por ejemplo: Supongamos que se tiene una base de datos con 2 observaciones (positivo y negativo) en el atributo de salida y 100 instancias, 80 son negativas y 20 positivas. Si se utiliza k-folds, por probabilidad se tendrían segmentos con la relación 8:2, existiendo la posiblidad de que hayan segmentos de entrenamiento y prueba que únicamente tengan una clase, teniendo mala exactitud (incorrecto).

Por otro lado se tiene el método de k-folds estratificado, el cual permite realizar el balance porcentual de las clases, por ejemplo si se tiene un k-fold estratificado $k = 5$ en total cada segmento representa el 20 % del conjunto. Entonces se tomaría en cada segmento el 20 % de cada observación de la clase. En este caso cada segmento contendrá (20 % de 80) $\rightarrow$ 16 muestras de la clase negativa y (20 % de 20) $\rightarrow$ 4 muestras de la clase positiva.

En conclusión k-folds hace una segmentación aleatoria y stratified k-folds hace una segmentación basada en la distribución igualitaria porcentual de las clases.

3.6. Codificación de datos categóricos

Se introducen los siguientes conceptos de codificación de los datos categóricos [6].

3.6.1. Codificación etiquetada (ordinal)

Se le asigna valores numéricos a cada categoría (véase Figura 4).

	player	point
0	Stephen Curry	4
1	Anthony Edwards	6
2	Anthony Edwards	1
3	Duncan Robinson	2
4	Duncan Robinson	3
5	Stephen Curry	5
6	Stephen Curry	2
7	Stephen Curry	3

	player	point
0	1	4
1	3	6
2	3	1
3	2	2
4	2	3
5	1	5
6	1	2
7	1	3

Figura 4: Codificación ordinal.

3.6.2. Codificación por frecuencia normalizada

Se cuenta la cantidad de valores diferentes y se calcula su porcentaje de aparición (véase Figura 5).

	player	point
0	Stephen Curry	4
1	Anthony Edwards	6
2	Anthony Edwards	1
3	Duncan Robinson	2
4	Duncan Robinson	3
5	Stephen Curry	5
6	Stephen Curry	2
7	Stephen Curry	3

	player	point
0	0.50	4
1	0.25	6
2	0.25	1
3	0.25	2
4	0.25	3
5	0.50	5
6	0.50	2
7	0.50	3

Figura 5: Codificación ordinal.

$$Ci_{freq} = \frac{fi_{cat}}{n_{instancias}} \tag{14}$$

Donde:

- Ci_{freq} es el valor codificado de la categoría i
- f_{cat} es la frecuencia (numero de veces que aparece) la categoria i
- $n_{instancias}$ es el número de instancias (renglones).

3.6.3. Codificación binaria

La cantidad de observaciones se numera de una manera binaria agregando atributos, en menor medida que One-hot, donde se incrementa un atributo por cada observación (véase Figura 6).

	player	point
0	Stephen Curry	4
1	Anthony Edwards	6
2	Anthony Edwards	1
3	Duncan Robinson	2
4	Duncan Robinson	3
5	Stephen Curry	5
6	Stephen Curry	2
7	Stephen Curry	3

	player_0	player_1	point
0	0	1	4
1	1	0	6
2	1	0	1
3	1	1	2
4	1	1	3
5	0	1	5
6	0	1	2
7	0	1	3

Figura 6: Codificación binaria.

4. Materiales y Métodos

4.1. Materiales

4.1.1. Base de datos

La base de datos fue obtenida de un repositorio de bases de datos de UCI [7], la cual consta de una base de datos para poder conocer si un hongo es venenosos o comestible, todo esto con base a sus características físicas. Esta base de datos cuenta con 8124 instancias y 23 atributos, entre el que

destaca el de clase ('class'), el cual indica si el hongo es venenoso ('p') o es comestible ('e'). Además de que todas las demás instancias son categóricas a excepción de la cantidad de anillos del hongo. De una manera más ilustrativa se observa en la Figura 7 la anatomía del hongo.

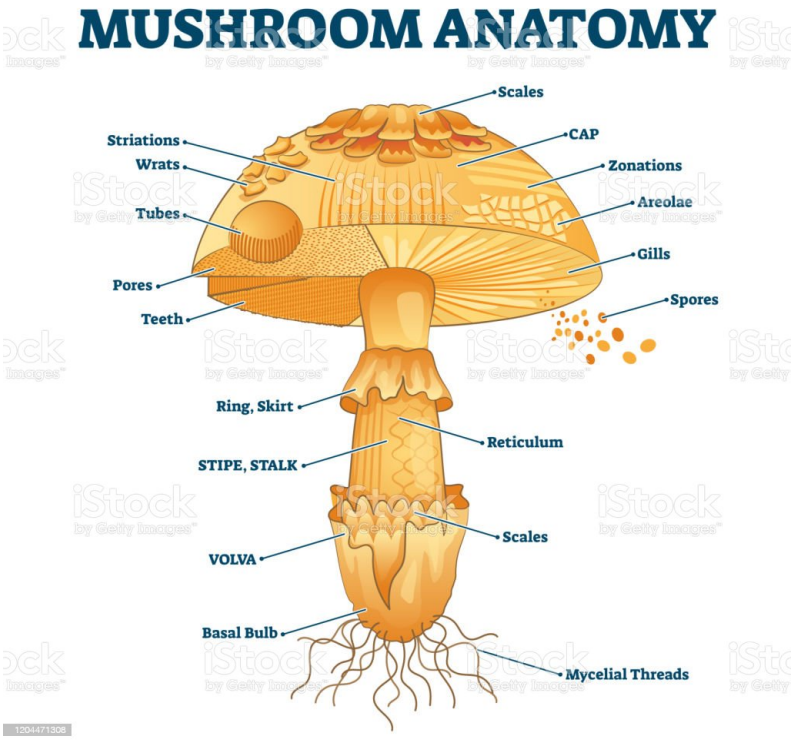


Figura 7: Anatomía de hongo (obtenido de [8]).

4.1.2. Librerías y entorno de desarrollo

El análisis de los datos se llevará a cabo en el lenguaje de programación Python dentro del entorno de desarrollo de Jupyter Notebook. Ocupando las librerías matplotlib, seaborn, numpy, Pandas, scipy, category_encoders y scikit-learn.

4.2. Metodología

La metodología consiste en la recopilación de las bases de datos, aplicar métodos de preprocesamiento de datos, realizar los algoritmos de clasificación y aplicar pruebas de segregación, k-folds y k-folds estratificado (véase Figura 8).

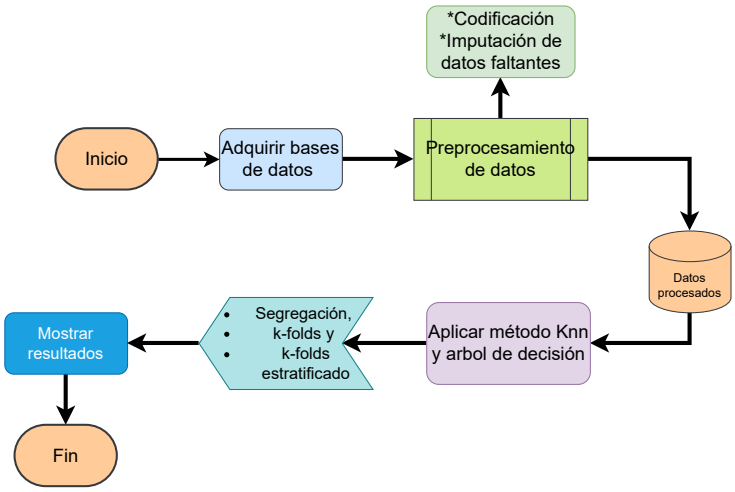


Figura 8: Metodología de la práctica.

5. Pseudocódigo

Algoritmo 1 Pseudocódigo regresión.

```
Inicio

Class codificar(datos,tipo):
        Imputar
        D_cod ← datos,tipo
        Regresar D_cod

Class knn(entrenamiento,prueba):
      Distancias
      Regresar Clase

Class árbol_ID3(Datos_entrenamiento,datos_prueba):
      Ajuste
      Probar
      Regresar Clase

Class métrica (Resultados, clases):
      Regresar (ex,pres,F1,sens)

Class segregación(%E,%P):
      Regresar (d_ent,d_pru)

Class kfolds(k)
     Regresar seg_d(k)

Class str_kfolds(datos):
      D_str_kfold ← kfolds(k),datos
      Regresar  D_str_kfold

Data ← codificar(data,[label,freq,bin])
res_tree← arbol_ID3(data)
m_tree ← métrica(res_tree,data)
Para i en Data:
     res_knn1[i] ←segregación(knn(Data[i])) ## tipo de codificación
     res_knn2[i] ← str_kfolds(kfolds(k=n),Data[i])
     m_knn1[i] ← métrica(res_knn1[i],Data[i])
     m_knn2[i] ← métrica(res_knn2[i],Data[i])
Fin
```

6. Resultados

6.1. Preprocesamiento y analisis de los datos

Se realizó la imputación por moda de clase en el atributo llamado *stalk-root*, obteniendo las siguientes modas

- Moda para clase venenoso: b
- Moda para clase comestible: b

Se realizaron las codificaciones anteriormente mencionadas, obteniendo las siguientes dimensiones.

- Codificación ordinal: (8124×23)
- Codificación por frecuencia: (8124×23)
- Codificación binaria: (8124×65)

6.2. Algoritmo ID3 segregación 80-20

Se realizó el algoritmo ID3, obteniendo la siguiente estructura (véase Figura 9):

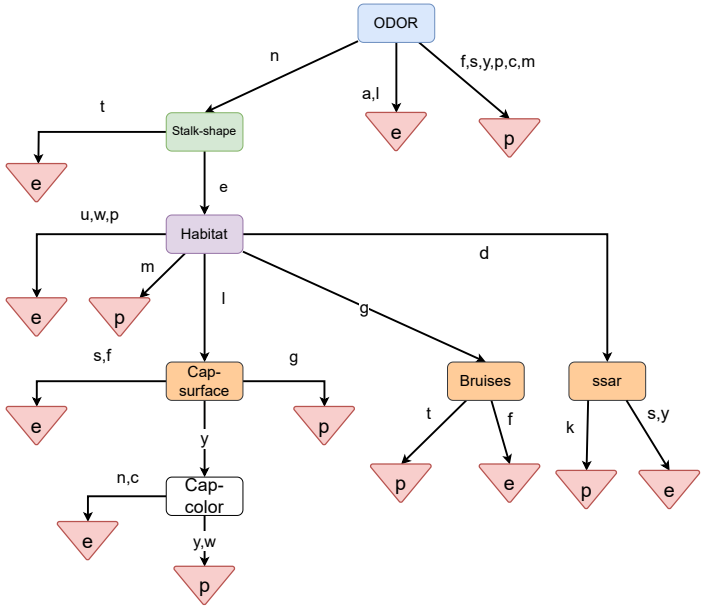


Figura 9: Estructura del árbol ID3.

Se observa que se tuvo una reducción de los atributos a 7, por lo que estas características definen lo más importante del árbol. Obteniendo los siguientes resultados con una segregación 80-20 (véase Figura 1). Cabe mencionar que se utilizó la base de datos cruda, es decir sin codificar.

Se tuvo un desempeño perfecto, por lo que se puede decir que con este modelo se puede predecir de una manera certera esta base de datos (véase Tabla 1).

Tabla 1: Resultado de desempeño del árbol de decisión con segregación 80-20.

Clase	Precision	Recall	F1-score	Cuenta
e	1	1	1	507
p	1	1	1	1118
		Exactitud	1	1625

6.3. Resultados algoritmo Knn

En este algoritmo se realizaron pruebas con el modelo knn con segregación, en diferentes casos de n y diferente cantidad de *splits* de k-fold estratificado (STF) (véase Tabla 2).

Tabla 2: Resultado de desempeño de knn con diferentes hipérparametros.

n	k	Codificación	Métricas	Segregación 80-20	K-folds	STF
5	7	Ordinal	exactitud	1	1	0.87
			Precisión	1	1	0.88
			Recall	1	1	0.87
			F1-Score	1	1	0.87
5	7	Binaria	exactitud	1	1	0.92
			Precisión	1	1	0.92
			Recall	1	1	0.92
			F1-Score	1	1	0.92
5	7	Frecuencia	exactitud	1	1	0.89
			Precisión	1	1	0.89
			Recall	1	1	0.89
			F1-Score	1	1	0.89
9	11	Ordinal	exactitud	1	1	0.87
			Precisión	1	1	0.87
			Recall	1	1	0.87
			F1-Score	1	1	0.87
9	11	Binaria	exactitud	1	1	.92
			Precisión	1	1	.92
			Recall	1	1	.92
			F1-Score	1	1	.92
9	11	Frecuencia	exactitud	1	1	.89
			Precisión	1	1	.89
			Recall	1	1	.89
			F1-Score	1	1	.89

Se observa que en las pruebas *convencionales* se tiene un desempeño perfecto, sin embargo, en las prueba estratificada, se bajo el rendimiento en un 10 a 15 % aproximadamente, esto se debe a los requerimientos de la prueba, la cual iguala el porcentaje de aparición de las clases al momento de dividir el conjunto de datos. Tambien se observó que el rendimiento cambió con el tipo de codificación,

el número de vecinos cercanos n y la cantidad de folds k, no afecto en gran medida el desempeño general del modelo.

7. Conclusiones

La presente práctica mostró la aplicación de clasificación de una base de datos etiquetada mediante modelos de aprendizaje automático, usando el algoritmo knn y de árbol de decisión ID3, haciendo uso de preprocesamiento de datos, específicamente el uso de imputación por moda de clases. La base de datos esta bien estructurada y tiene mucha información referente al tema, además de que al pertenecer a una universidad de alto prestigio, se tiene una mayor fiabilidad de los datos obtenidos.

En los resultados obtenidos, fue posible observar que con las pruebas básicas para testear los modelos, se obtuvo fácilmente el 100 % en las métricas. Además de observar una estabilidad en las métricas en el intervalo de $1 > n < 15$, por lo que se encontraba en el rango de n adecuado. Por parte del árbol de decisión, se observó que la entropía de los datos era baja y por lo tanto muchos atributos no fueron incluidos, teniendo un desempeño de 100 % también. Por otro lado, el valor k de STF tampoco se observó que afectará mucho al rendimiento entre pruebas, esto debido a que al balancear porcentualmente las clases en cada segmento, se pudo homogeneizar el resultado de una mejor manera.

Sin embargo al momento de realizar el algoritmo de knn con la prueba STF, se observo una disminución en el rendimiento del modelo, mostrando que los datos que se estaban seleccionando para entrenar no eran lo suficientemente relevantes o no cumplían con un balance adecuado de clases, evitando sobre-entrenamiento. A pesar de ello, se tuvieron buenas métricas de desempeño del modelo, observándose que en la codificación binaria se tiene el mejor desempeño, esto debido a la naturaleza de los datos, los cuales son categóricos nominales (el orden no importa).

Es posible mejorar el rendimiento si se realiza una categoria personalizada para cada atributo segun sea el caso.

Se utilizó el algoritmo knn debido a la simpleza para poder entrenar el modelo a pesar de trabajar con muchas instancias, y se trabajó el algoritmo de arbol de decisión ID3 porque se observó una reducción amplia de las instancias, aunque implique re entrenarlo en cada prueba. Obteniendo resultados bastante aceptables y buenos, comprobando que el tipo de prueba puede decirnos más sobre el modelo que se está implementando.

Referencias

[1] M. Fernández, *Inteligencia Artificial para Programadores con Prisa*. Amazon Digital Services LLC - KDP Print US, 2021.

[2] "Descubra el algoritmo knn : un algoritmo de aprendizaje supervisado." https://datascientest.com/es/que-es-el-algoritmo-knn. (Accessed on 06/01/2023).

[3] J. Chouinard, "k-nearest neighbors (knn) in python." https://www.jcchouinard.com/k-nearest-neighbors/, May 2022. (Accessed on 05/29/2023).

[4] M. REDA, "Knn & precision and recall." https://www.kaggle.com/code/mahmoudreda55/knn-precision-and-recall, 2021. (Accessed on 06/01/2023).

[5] GeeksforGeeks, "Stratified k fold cross validation." https://www.geeksforgeeks.org/stratified-k-fold-cross-validation/, Enero 2023. (Accessed on 06/22/2023).

[6] A. Cavin, "6 ways to encode features for machine learning algorithms." https://towardsdatascience.com/6-ways-to-encode-features-for-machine-learning-algorithms-21593f6 (Accessed on 06/22/2023).

[7] UCI, "Mushroom - uci machine learning repository." https://archive.ics.uci.edu/dataset/73/mushroom. (Accessed on 06/22/2023).

[8] Istock, "Ilustración de anatomía de setas etiquetada sóloque de ilustración vectorial de diagrama de biología y más vectores libres de derechos de hongo." https://www.istockphoto.com/es/vector/anatom%C3%ADa-de-setas-etiquetada-s%C3%B3loque-de-ilustraci%C3%B3n-vectorial-de-diagrama-de-gm1204471308-346590959. (Accessed on 06/22/2023).

Clasificacion de hongos

June 22, 2023

[31]:
```
import csv
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

0.1 Base de datos de hongos

respuestas en bd hongos res

[32]:
```
data_file = 'agaricus-lepiota.data'
data=[]
with open(data_file, 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        # Procesar cada fila del archivo .data
        data.append(row)
        #print(row)

df=pd.DataFrame(data)
column_names=['class','cap-shape','cap-surface','cap-color','bruises','odor','gill-attachment',
              ␣
 ↪'gill-color','stalk-shape','stalk-root','ssar','ssbr','scar','scbr','veil-type','veil-color',
              'ring-number','ring-type','spore-p-color','population','habitat']
df.columns=column_names
# Leer el archivo .names
names_file = 'agaricus-lepiota.names'
with open(names_file, 'r') as file:
    lines = file.readlines()
    for line in lines:
        # Procesar cada línea del archivo .names
        print(line)
```

```
1. Title: Mushroom Database


2. Sources:
```

(a) Mushroom records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf

(b) Donor: Jeff Schlimmer (Jeffrey.Schlimmer@a.gp.cs.cmu.edu)

(c) Date: 27 April 1987

3. Past Usage:

1. Schlimmer,J.S. (1987). Concept Acquisition Through Representational Adjustment (Technical Report 87-19). Doctoral disseration, Department of Information and Computer Science, University of California, Irvine.
--- STAGGER: asymptoted to 95% classification accuracy after reviewing 1000 instances.
2. Iba,W., Wogulis,J., & Langley,P. (1988). Trading off Simplicity and Coverage in Incremental Concept Learning. In Proceedings of the 5th International Conference on Machine Learning, 73-79. Ann Arbor, Michigan: Morgan Kaufmann.
-- approximately the same results with their HILLARY algorithm
3. In the following references a set of rules (given below) were learned for this data set which may serve as a point of comparison for other researchers.

Duch W, Adamczak R, Grabczewski K (1996) Extraction of logical rules from training data using backpropagation networks, in: Proc. of the The 1st Online Workshop on Soft Computing, 19-30.Aug.1996, pp. 25-30,

available on-line at: http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/

Duch W, Adamczak R, Grabczewski K, Ishikawa M, Ueda H, Extraction of crisp logical rules using constrained backpropagation networks - comparison of two new approaches, in: Proc. of the European Symposium on Artificial Neural Networks (ESANN'97), Bruge, Belgium 16-18.4.1997, pp. xx-xx

Wlodzislaw Duch, Department of Computer Methods, Nicholas Copernicus University, 87-100 Torun, Grudziadzka 5, Poland
e-mail: duch@phys.uni.torun.pl
WWW http://www.phys.uni.torun.pl/kmk/

Date: Mon, 17 Feb 1997 13:47:40 +0100
From: Wlodzislaw Duch <duch@phys.uni.torun.pl>
Organization: Dept. of Computer Methods, UMK

I have attached a file containing logical rules for mushrooms.
It should be helpful for other people since only in the last year I have seen about 10 papers analyzing this dataset and obtaining quite complex rules. We will try to contribute other results later.

With best regards, Wlodek Duch

```
Logical rules for the mushroom data sets.

Logical rules given below seem to be the simplest possible for the
mushroom dataset and therefore should be treated as benchmark results.

Disjunctive rules for poisonous mushrooms, from most general
to most specific:

P_1) odor=NOT(almond.OR.anise.OR.none)
     120 poisonous cases missed, 98.52% accuracy

P_2) spore-print-color=green
     48 cases missed, 99.41% accuracy

P_3) odor=none.AND.stalk-surface-below-ring=scaly.AND.
          (stalk-color-above-ring=NOT.brown)
     8 cases missed, 99.90% accuracy

P_4) habitat=leaves.AND.cap-color=white
         100% accuracy

Rule P_4) may also be
```

```
P_4') population=clustered.AND.cap_color=white

These rule involve 6 attributes (out of 22). Rules for edible
mushrooms are obtained as negation of the rules given above, for
example the rule:

odor=(almond.OR.anise.OR.none).AND.spore-print-color=NOT.green

gives 48 errors, or 99.41% accuracy on the whole dataset.

Several slightly more complex variations on these rules exist,
involving other attributes, such as gill_size, gill_spacing,
stalk_surface_above_ring, but the rules given above are the simplest
we have found.
```

```
4. Relevant Information:
   This data set includes descriptions of hypothetical samples
   corresponding to 23 species of gilled mushrooms in the Agaricus and
   Lepiota Family (pp. 500-525).  Each species is identified as
   definitely edible, definitely poisonous, or of unknown edibility and
   not recommended.  This latter class was combined with the poisonous
   one.  The Guide clearly states that there is no simple rule for
   determining the edibility of a mushroom; no rule like ``leaflets
```

```
    three, let it be'' for Poisonous Oak and Ivy.


5. Number of Instances: 8124


6. Number of Attributes: 22 (all nominally valued)


7. Attribute Information: (classes: edible=e, poisonous=p)
     1. cap-shape:                bell=b,conical=c,convex=x,flat=f,
                                  knobbed=k,sunken=s
     2. cap-surface:              fibrous=f,grooves=g,scaly=y,smooth=s
     3. cap-color:                brown=n,buff=b,cinnamon=c,gray=g,green=r,
                                  pink=p,purple=u,red=e,white=w,yellow=y
     4. bruises?:                 bruises=t,no=f
     5. odor:                     almond=a,anise=l,creosote=c,fishy=y,foul=f,
                                  musty=m,none=n,pungent=p,spicy=s
     6. gill-attachment:          attached=a,descending=d,free=f,notched=n
     7. gill-spacing:             close=c,crowded=w,distant=d
     8. gill-size:                broad=b,narrow=n
     9. gill-color:               black=k,brown=n,buff=b,chocolate=h,gray=g,
                                  green=r,orange=o,pink=p,purple=u,red=e,
                                  white=w,yellow=y
    10. stalk-shape:              enlarging=e,tapering=t
    11. stalk-root:               bulbous=b,club=c,cup=u,equal=e,
                                  rhizomorphs=z,rooted=r,missing=?
```

```
    12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s

    13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s

    14. stalk-color-above-ring:   brown=n,buff=b,cinnamon=c,gray=g,orange=o,
                                  pink=p,red=e,white=w,yellow=y

    15. stalk-color-below-ring:   brown=n,buff=b,cinnamon=c,gray=g,orange=o,
                                  pink=p,red=e,white=w,yellow=y

    16. veil-type:                partial=p,universal=u

    17. veil-color:               brown=n,orange=o,white=w,yellow=y

    18. ring-number:              none=n,one=o,two=t

    19. ring-type:                cobwebby=c,evanescent=e,flaring=f,large=l,
                                  none=n,pendant=p,sheathing=s,zone=z

    20. spore-print-color:        black=k,brown=n,buff=b,chocolate=h,green=r,
                                  orange=o,purple=u,white=w,yellow=y

    21. population:               abundant=a,clustered=c,numerous=n,
                                  scattered=s,several=v,solitary=y

    22. habitat:                  grasses=g,leaves=l,meadows=m,paths=p,
                                  urban=u,waste=w,woods=d


8. Missing Attribute Values: 2480 of them (denoted by "?"), all for
   attribute #11.


9. Class Distribution:

    --    edible: 4208 (51.8%)

    -- poisonous: 3916 (48.2%)
```

```
--     total: 8124 instances
```

[33]:
```
class_column = df['class']
df = df.drop('class', axis=1)
df['class'] = class_column
df
```

[33]:

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	\
0	x	s	n	t	p	f	
1	x	s	y	t	a	f	
2	b	s	w	t	l	f	
3	x	y	w	t	p	f	
4	x	s	g	f	n	f	
...	...	...	...	...	...	...	
8119	k	s	n	f	n	a	
8120	x	s	n	f	n	a	
8121	f	s	n	f	n	a	
8122	k	y	n	f	y	f	
8123	x	s	n	f	n	a	

	gill-spacing	gill-size	gill-color	stalk-shape	...	scar	scbr	veil-type	\
0	c	n	k	e	...	w	w	p	
1	c	b	k	e	...	w	w	p	
2	c	b	n	e	...	w	w	p	
3	c	n	n	e	...	w	w	p	
4	w	b	k	t	...	w	w	p	
...	...	...	...	...	...	...	...	...	
8119	c	b	y	e	...	o	o	p	
8120	c	b	y	e	...	o	o	p	
8121	c	b	n	e	...	o	o	p	
8122	c	n	b	t	...	w	w	p	
8123	c	b	y	e	...	o	o	p	

	veil-color	ring-number	ring-type	spore-p-color	population	habitat	class
0	w	o	p	k	s	u	p
1	w	o	p	n	n	g	e
2	w	o	p	n	n	m	e
3	w	o	p	k	s	u	p
4	w	o	e	n	a	g	e
...	...	...	...	...	...	...	...
8119	o	o	p	b	c	l	e
8120	n	o	p	b	v	l	e
8121	o	o	p	b	c	l	e
8122	w	o	e	w	v	l	p
8123	o	o	p	o	c	l	e

```
[8124 rows x 23 columns]
```

0.2 Imputación del atributo 'stalk-root'

Se realizará una imputación por modas de clases. métodos Básicamente reemplaza el valor más frecuente de cada clase de salida (venenoso y comestible).

[34]:
```
moda_venenoso=df['stalk-root'][(df['class']=='p') & (df['stalk-root']!='?')].
↪mode()
print('Moda de stalk-root venenoso:', str(moda_venenoso[0]))
moda_comestible=df['stalk-root'][(df['class']=='e') & (df['stalk-root']!='?')].
↪mode()
print('Moda de stalk-root comestible;',str(moda_comestible[0]))
```

```
Moda de stalk-root venenoso: b
Moda de stalk-root comestible; b
```

[35]:
```
#df['stalk-root'][df['class']=='p'].replace('?',moda_venenoso[0],inplace=True)
#df['stalk-root'][df['class']=='e'].replace('?',moda_comestible[0],inplace=True)
#df['stalk-root'][df['class']].replace('?','b',inplace=True)
condicion = df['class'] == 'p'
df['stalk-root'] = np.where(condicion, df['stalk-root'].replace('?',␣
↪moda_venenoso[0]), df['stalk-root'])

condicion = df['class'] == 'e'
df['stalk-root'] = np.where(condicion, df['stalk-root'].replace('?',␣
↪moda_comestible[0]), df['stalk-root'])
```

[36]:
```
df['stalk-root'].value_counts()
```

[36]:
```
b    6256
e    1120
c     556
r     192
Name: stalk-root, dtype: int64
```

Se asigna un número único a cada valor único en la columna 0 venenoso ('p', poisonus) 1 no venenoso ('e', edible)

Los métodos de codificación que se hicieron fueron obtenidos de codificación

[37]:
```
# División de xy
d_temp_class=pd.factorize(df['class'])[0]
d_temp_class=pd.DataFrame(d_temp_class,columns=['class'])
d_temp_x=df.drop('class',axis=1)
```

1 Codificación ordinal

Se asigma un número único a cada valor único de la característica

[38]:
```
from category_encoders import OrdinalEncoder
encoder=OrdinalEncoder(cols=list(d_temp_x.columns))
encoder.fit(d_temp_x)
denc_ord=encoder.transform(d_temp_x)
denc_ord=pd.concat([denc_ord,d_temp_class],axis=1)
denc_ord.shape
```

[38]:
```
(8124, 23)
```

1.1 Codificación por frecuencia

Básicamente se cuenta la cantidad de valores diferentes y se calcula su porcentaje de aparición

$$Ci_{freq} = \frac{fi_{cat}}{n_{instancias}}$$

Donde: * Ci_{freq} es el valor codificado de la categoria i * f_{cat} es la frecuencia (numero de veces que aparece) la categoria i * $n_{instancias}$ es el número de instancias (renglones).

[39]:
```
from category_encoders import CountEncoder
encoder = CountEncoder(cols=list(d_temp_x.columns), normalize=True)
d_count_x = encoder.fit_transform(d_temp_x)
denc_count=pd.concat([d_count_x,d_temp_class],axis=1) ### Segunda codificacion
denc_count.shape
```

[39]:
```
(8124, 23)
```

1.2 Codificación binaria

Permite aumentar poco la dimensión del modelo. En comparación a One-hot

[40]:
```
from category_encoders import BinaryEncoder

encoder_bin=BinaryEncoder(cols=list(d_temp_x.columns))
d_bin_x=encoder_bin.fit_transform(d_temp_x)
denc_bin=pd.concat([d_bin_x,d_temp_class],axis=1) ### Segunda codificacion
denc_bin.shape
```

[40]:
```
(8124, 65)
```

[41]:
```
#tree=fit(d_temp_x,d_temp_class,list(d_temp_x.columns))  ### drop class en x
```

[42]:
```
### Método de segregación 80/20
tr=int(len(df)*0.8)
np.random.seed(167)
```

```
## Sin codificar
train_raw=df.sample(tr).reset_index(drop=True) # Prueba con
test_raw=df[~df.isin(train_raw)].dropna().reset_index(drop=True)
## Codificación ordinal
train_ordinal=denc_ord.sample(tr).reset_index(drop=True)
test_ordinal=denc_ord[~denc_ord.isin(train_ordinal)].dropna().
 ↪reset_index(drop=True)
## Codificación por frecuencias
train_freq=denc_count.sample(tr).reset_index(drop=True)
test_freq=denc_count[~denc_count.isin(train_freq)].dropna().
 ↪reset_index(drop=True)
## Codificación binaria
train_bin=denc_bin.sample(tr).reset_index(drop=True)
test_bin=denc_bin[~denc_bin.isin(train_bin)].dropna().reset_index(drop=True)
```

[43]:
```
#from sklearn.model_selection import cross_val_score
#knn1=KNeighborsClassifier()
"""
k_values=range(1,50)
cross_val_scores=[]
X=denc_ord.drop('class',axis=1)
y=denc_ord['class']
for k in k_values:
    knn=KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5)
    cross_val_scores.append(np.mean(scores))

best_k=k_values[np.argmax(cross_val_scores)]

for k,score in zip(k_values,cross_val_scores):
        print(f"K = {k}, Puntaje promedio: {score:.4f}")

print(f"\nMejor valor de k: {best_k}")
plt.scatter(k_values,cross_val_scores)
"""
```

[43]:
```
'\nk_values=range(1,50)\ncross_val_scores=[]\nX=denc_ord.drop(\'class\',axis=1)\
ny=denc_ord[\'class\']\nfor k in k_values:\n
knn=KNeighborsClassifier(n_neighbors=k)\n    scores = cross_val_score(knn, X, y,
cv=5) \n    cross_val_scores.append(np.mean(scores))\n\nbest_k=k_values[np.argma
x(cross_val_scores)]\n\nfor k,score in zip(k_values,cross_val_scores):\n
print(f"K = {k}, Puntaje promedio: {score:.4f}")\n\nprint(f"\nMejor valor de k:
{best_k}")\nplt.scatter(k_values,cross_val_scores)\n'
```

1.3 Stratified kfolds

Se obtuvo de g4g

[44]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score
skf=StratifiedKFold(n_splits=5)
```

1.4 Codificación ordinal

1.4.1 Segregación

[103]:
```
knn_ords=KNeighborsClassifier(n_neighbors=5)
knn_ords.fit(train_ordinal.drop('class',axis=1),train_ordinal['class'])
y_pred_seg=knn_ords.predict(test_ordinal.drop('class',axis=1))
report=classification_report(test_ordinal['class'],y_pred_seg)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1118
         1.0       1.00      1.00      1.00       507

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

1.5 K-fold

[104]:
```
from sklearn.model_selection import KFold


kfold=KFold(n_splits=7,shuffle=True,random_state=13)
knn_kfold_ord=KNeighborsClassifier(n_neighbors=5)
#scores=cross_val_score(knn_kfold_ord,knn_ord_stf,denc_ord.
↪drop('class',axis=1),denc_ord['class'])
#y_pred_kfold_ord=knn_kfold_ord.fit().
#print(scores)
all_y_true = []
all_y_pred = []
for train_index,test_index in kfold.split(denc_ord.
↪drop('class',axis=1),denc_ord['class']):
    x_train,x_test=denc_ord.drop('class',axis=1).iloc[train_index],denc_ord.
↪drop('class',axis=1).iloc[test_index]
    y_train,y_test=denc_ord.iloc[train_index]['class'],denc_ord.
↪iloc[test_index]['class']
```

```
    knn_kfold_ord.fit(x_train,y_train)
    y_pred=knn_kfold_ord.predict(x_test)
    report = classification_report(y_test, y_pred)
    #print(report) ## Por si se desea observar el reporte por fold, se␣
 ↪descomenta esto####
    all_y_true.extend(y_test)
    all_y_pred.extend(y_pred)

all_y_true = np.array(all_y_true)
all_y_pred = np.array(all_y_pred)

# Genera el informe de clasificación general
report = classification_report(all_y_true, all_y_pred)
print("Informe de clasificación general:")
print(report)
```

```
Informe de clasificación general:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3916
           1       1.00      1.00      1.00      4208

    accuracy                           1.00      8124
   macro avg       1.00      1.00      1.00      8124
weighted avg       1.00      1.00      1.00      8124
```

1.5.1 STF

[105]:
```
skf_ord=StratifiedKFold(n_splits=7)
knn_ord_stf=KNeighborsClassifier(n_neighbors=5)
y_pred_ord_stf=cross_val_predict(knn_ord_stf,denc_ord.
 ↪drop('class',axis=1),denc_ord['class'])
rep_ord_stf=classification_report(denc_ord['class'],y_pred_ord_stf)
print(rep_ord_stf)
confusion = confusion_matrix(y_pred_ord_stf, denc_ord['class'])
sns.heatmap(confusion,annot=True, fmt='d', cmap='Blues')
plt.xlabel('Clase Predicha')
plt.ylabel('Clase Real')
plt.title('Matriz de Confusión de STF codificación ordinal')
```

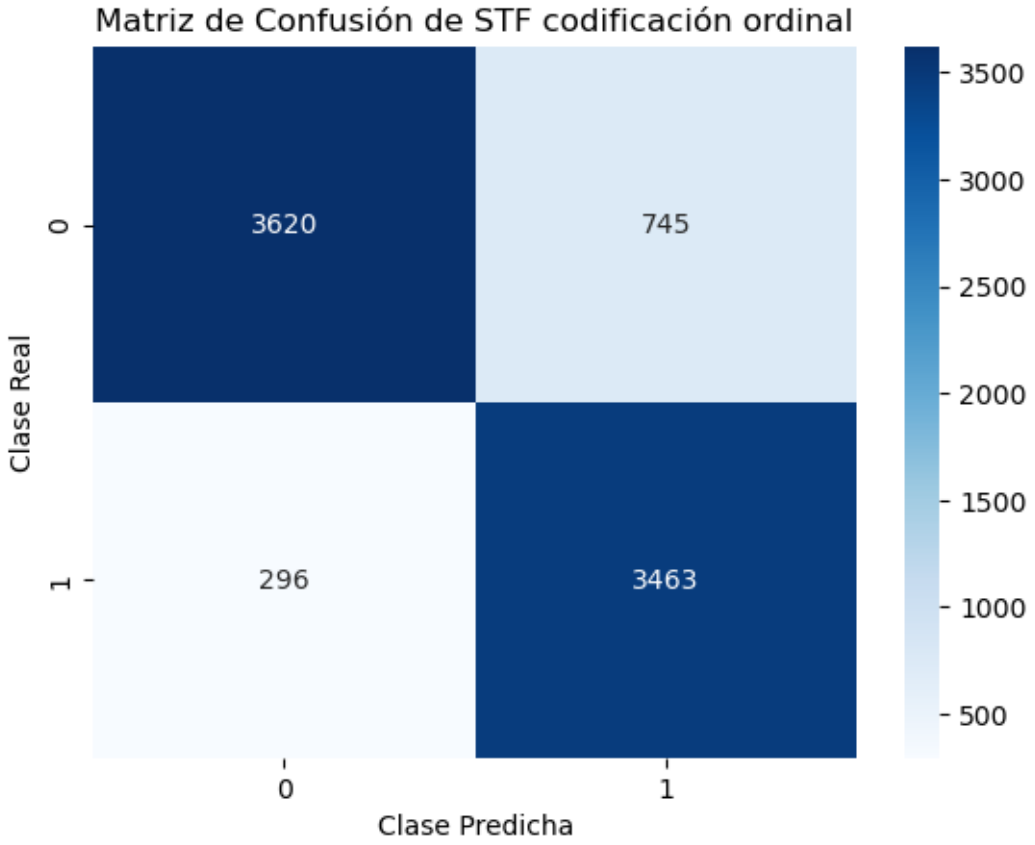
```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      3916
           1       0.92      0.82      0.87      4208

    accuracy                           0.87      8124
```

```
          macro avg       0.88      0.87      0.87      8124
       weighted avg       0.88      0.87      0.87      8124
```

[105]: `Text(0.5, 1.0, 'Matriz de Confusión de STF codificación ordinal')`



1.6 Codificación frecuencias normalizadas

1.6.1 Segregación

[106]:
```
knn_frec=KNeighborsClassifier(n_neighbors=5)
knn_frec.fit(train_freq.drop('class',axis=1),train_freq['class'])
y_pred_seg=knn_frec.predict(test_freq.drop('class',axis=1))
report=classification_report(test_freq['class'],y_pred_seg)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1118
         1.0       1.00      1.00      1.00       507
```

```
    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

1.6.2 K-folds

[107]:
```
kfold=KFold(n_splits=7,shuffle=True,random_state=13)
knn_kfold_count=KNeighborsClassifier(n_neighbors=5)
#scores=cross_val_score(knn_kfold_ord,knn_ord_stf,denc_ord.
 ↪drop('class',axis=1),denc_ord['class'])
#y_pred_kfold_ord=knn_kfold_ord.fit().
#print(scores)
all_y_true = []
all_y_pred = []
for train_index,test_index in kfold.split(denc_count.
 ↪drop('class',axis=1),denc_count['class']):
    x_train,x_test=denc_count.drop('class',axis=1).iloc[train_index],denc_count.
 ↪drop('class',axis=1).iloc[test_index]
    y_train,y_test=denc_count.iloc[train_index]['class'],denc_count.
 ↪iloc[test_index]['class']

    knn_kfold_count.fit(x_train,y_train)
    y_pred=knn_kfold_count.predict(x_test)
    report = classification_report(y_test, y_pred)
    #print(report) ## Por si se desea observar el reporte por fold, se␣
 ↪descomenta esto####
    all_y_true.extend(y_test)
    all_y_pred.extend(y_pred)

all_y_true = np.array(all_y_true)
all_y_pred = np.array(all_y_pred)

# Genera el informe de clasificación general
report = classification_report(all_y_true, all_y_pred)
print("Informe de clasificación general:")
print(report)
```

```
Informe de clasificación general:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3916
           1       1.00      1.00      1.00      4208

    accuracy                           1.00      8124
   macro avg       1.00      1.00      1.00      8124
weighted avg       1.00      1.00      1.00      8124
```

1.6.3 STF

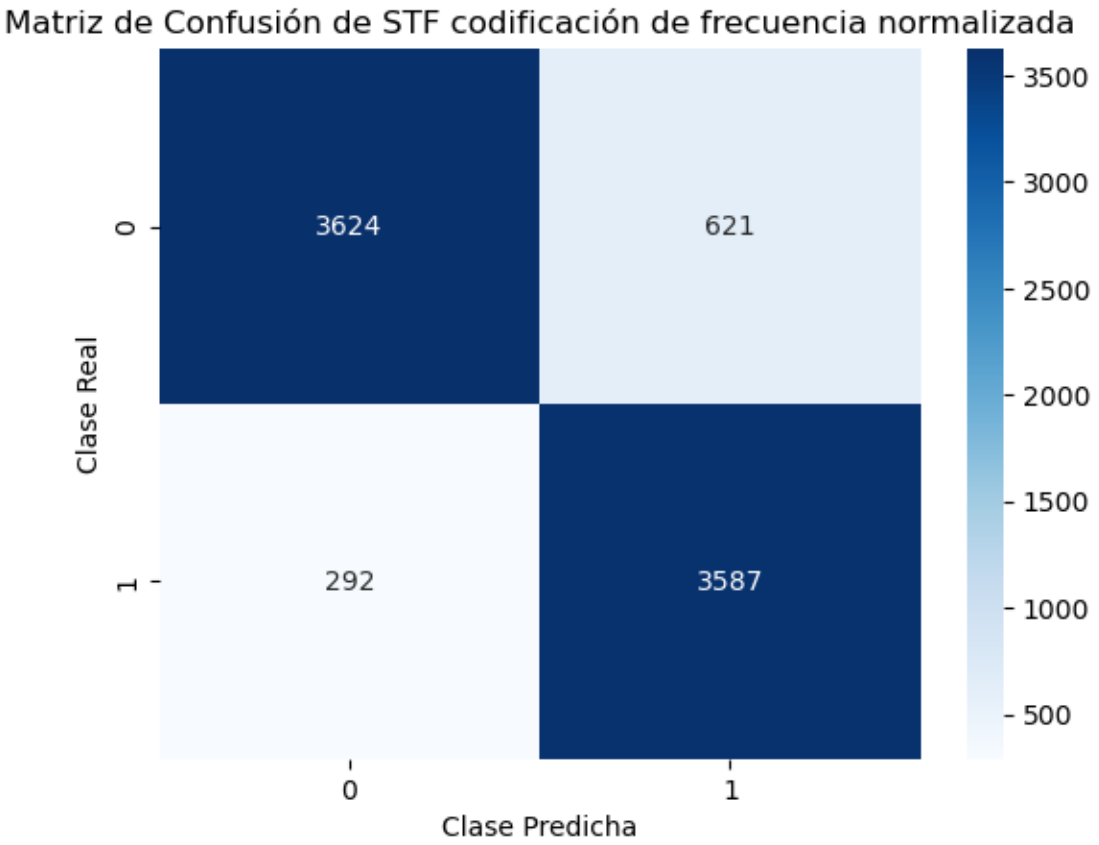
[108]:
```
skf_count=StratifiedKFold(n_splits=7)
knn_count_stf=KNeighborsClassifier(n_neighbors=5)
y_pred_count_stf=cross_val_predict(knn_count_stf,denc_count.
 ↪drop('class',axis=1),denc_count['class'])
rep_count_stf=classification_report(denc_count['class'],y_pred_count_stf)
print(rep_count_stf)
confusion = confusion_matrix(y_pred_count_stf, denc_count['class'])
sns.heatmap(confusion,annot=True, fmt='d', cmap='Blues')
plt.xlabel('Clase Predicha')
plt.ylabel('Clase Real')
plt.title('Matriz de Confusión de STF codificación de frecuencia normalizada')
```

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      3916
           1       0.92      0.85      0.89      4208

    accuracy                           0.89      8124
   macro avg       0.89      0.89      0.89      8124
weighted avg       0.89      0.89      0.89      8124
```

[108]:
```
Text(0.5, 1.0, 'Matriz de Confusión de STF codificación de frecuencia
normalizada')
```



1.7 Codificación binaria

1.7.1 segregación

[112]:
```
knn_bin=KNeighborsClassifier(n_neighbors=9)
knn_bin.fit(train_bin.drop('class',axis=1),train_bin['class'])
y_pred_seg=knn_bin.predict(test_bin.drop('class',axis=1))
report=classification_report(test_bin['class'],y_pred_seg)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1118
         1.0       1.00      1.00      1.00       507

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

1.7.2 Kfolds

[113]:
```
kfold=KFold(n_splits=11,shuffle=True,random_state=13)
knn_kfold_bin=KNeighborsClassifier(n_neighbors=9)
#scores=cross_val_score(knn_kfold_ord,knn_ord_stf,denc_ord.
 ↪drop('class',axis=1),denc_ord['class'])
#y_pred_kfold_ord=knn_kfold_ord.fit().
#print(scores)
all_y_true = []
all_y_pred = []
for train_index,test_index in kfold.split(denc_bin.
 ↪drop('class',axis=1),denc_bin['class']):
    x_train,x_test=denc_bin.drop('class',axis=1).iloc[train_index],denc_bin.
 ↪drop('class',axis=1).iloc[test_index]
    y_train,y_test=denc_bin.iloc[train_index]['class'],denc_bin.
 ↪iloc[test_index]['class']

    knn_kfold_bin.fit(x_train,y_train)
    y_pred=knn_kfold_bin.predict(x_test)
    report = classification_report(y_test, y_pred)
    #print(report) ## Por si se desea observar el reporte por fold, se␣
 ↪descomenta esto####
    all_y_true.extend(y_test)
    all_y_pred.extend(y_pred)

all_y_true = np.array(all_y_true)
all_y_pred = np.array(all_y_pred)

# Genera el informe de clasificación general
report = classification_report(all_y_true, all_y_pred)
print("Informe de clasificación general:")
print(report)
```

```
Informe de clasificación general:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3916
           1       1.00      1.00      1.00      4208

    accuracy                           1.00      8124
   macro avg       1.00      1.00      1.00      8124
weighted avg       1.00      1.00      1.00      8124
```

[114]:
```
skf_bin=StratifiedKFold(n_splits=11)
knn_bin_stf=KNeighborsClassifier(n_neighbors=9)
y_pred_bin_stf=cross_val_predict(knn_bin_stf,denc_bin.
 ↪drop('class',axis=1),denc_bin['class'])
```

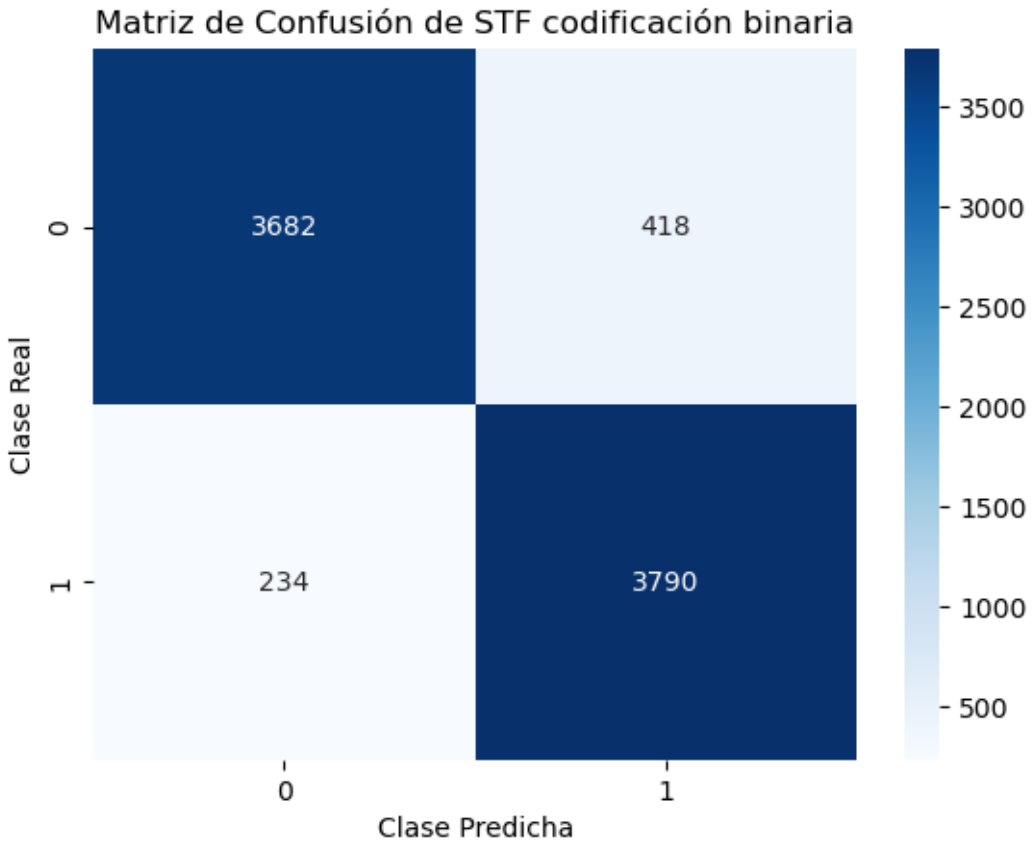
```
rep_bin_stf=classification_report(denc_bin['class'],y_pred_bin_stf)
print(rep_bin_stf)
confusion = confusion_matrix(y_pred_bin_stf, denc_bin['class'])
sns.heatmap(confusion,annot=True, fmt='d', cmap='Blues')
plt.xlabel('Clase Predicha')
plt.ylabel('Clase Real')
plt.title('Matriz de Confusión de STF codificación binaria')
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92      3916
           1       0.94      0.90      0.92      4208

    accuracy                           0.92      8124
   macro avg       0.92      0.92      0.92      8124
weighted avg       0.92      0.92      0.92      8124
```

[114]: Text(0.5, 1.0, 'Matriz de Confusión de STF codificación binaria')



2 Arboles de decisión (ID3)

[54]:
```
def entropy(y):
    # Calcular la entropía del conjunto de datos
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / len(y)
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

def information_gain(df, target_col, feature_col):
    # Calcular la ganancia de información para una característica específica
    entropy_before = entropy(df[target_col])
    _, counts = np.unique(df[feature_col], return_counts=True)
    probabilities = counts / len(df)

    entropy_after = 0
    for value, probability in zip(df[feature_col], probabilities):
        subset_df = df[df[feature_col] == value]
        subset_y = subset_df[target_col]
        entropy_after += probability * entropy(subset_y)

    information_gain = entropy_before - entropy_after
    return information_gain

def best_feature(df, target_col):
    # Encontrar la mejor característica para dividir el conjunto de datos
    num_features = len(df.columns) - 1
    best_feature = None
    best_information_gain = -1

    for feature in df.columns[:-1]:
        information_gain_value = information_gain(df, target_col, feature)
        if information_gain_value > best_information_gain:
            best_information_gain = information_gain_value
            best_feature = feature

    return best_feature

def fit(df, target_col):
    # Construir el árbol de decisión mediante el algoritmo ID3 recursivo
    tree = {}

    if len(df[target_col].unique()) == 1:
        # Si todas las muestras tienen la misma clase, establecerla como la␣
 ↪clase de la hoja
        tree['label'] = df[target_col].iloc[0]
        return tree
```

```
    if len(df.columns) == 1:
        # Si no quedan características para dividir, establecer la clase␣
↪mayoritaria como la clase de la hoja
        majority_class = df[target_col].value_counts().idxmax()
        tree['label'] = majority_class
        return tree

    best_feature_col = best_feature(df, target_col)

    tree['feature'] = best_feature_col
    tree['children'] = {}

    unique_values = df[best_feature_col].unique()
    for value in unique_values:
        subset_df = df[df[best_feature_col] == value]
        if len(subset_df) == 0:
            majority_class = df[target_col].value_counts().idxmax()
            tree['children'][value] = {'label': majority_class}
        else:
            subset_df = subset_df.drop(columns=best_feature_col)
            tree['children'][value] = fit(subset_df, target_col)

    return tree

def predict_instance(tree, instance):
    # Predecir la clase de una instancia utilizando el árbol de decisión
    current_node = tree
    while 'label' not in current_node:
        feature = current_node['feature']
        feature_value = instance[feature]
        if feature_value not in current_node['children']:
            return None
        current_node = current_node['children'][feature_value]
    return current_node['label']

def predict_dataframe(tree, df):
    # Predecir la clase de un DataFrame de atributos utilizando el árbol de␣
↪decisión
    predictions = []
    for _, instance in df.iterrows():
        prediction = predict_instance(tree, instance)
        predictions.append(prediction)
    return predictions
```

[55]:
```
#import pandas as pd
def calculate_node_entropies(tree, df, target_col):
```

```
    node_entropies = {}

    def traverse_tree(node, path):
        if 'label' in node:
            return

        feature = node['feature']
        children = node['children']

        for value, child in children.items():
            if 'label' in child:
                continue

            new_path = path + [(feature, value)]
            traverse_tree(child, new_path)

        if feature not in node_entropies:
            node_entropies[feature] = {}

        node_entropies[feature][tuple(path)] = entropy(df[target_col])

    traverse_tree(tree, [])

    # Create a DataFrame from the node_entropies dictionary
    df_entropies = pd.DataFrame(node_entropies).T
    df_entropies.index.name = 'Node'
    df_entropies.columns.names = ['Feature', 'Path']

    return df_entropies
```

[56]:
```
tree=fit(train_raw,'class')
res_raw_tree=predict_dataframe(tree,test_raw.drop('class',axis=1))
res_raw_tree=pd.DataFrame(res_raw_tree,columns=['class'])
```

[57]:
```
tree
```

[57]:
```
{'feature': 'odor',
 'children': {'n': {'feature': 'stalk-shape',
   'children': {'t': {'label': 'e'},
    'e': {'feature': 'habitat',
     'children': {'l': {'feature': 'cap-surface',
       'children': {'s': {'label': 'e'},
        'y': {'feature': 'cap-color',
         'children': {'n': {'label': 'e'},
          'y': {'label': 'p'},
          'w': {'label': 'p'},
          'c': {'label': 'e'}}},
```

```
            'f': {'label': 'e'},
            'g': {'label': 'p'}}},
          'w': {'label': 'e'},
          'p': {'label': 'e'},
          'm': {'label': 'p'},
          'g': {'feature': 'bruises',
           'children': {'f': {'label': 'e'}, 't': {'label': 'p'}}},
          'u': {'label': 'e'},
          'd': {'feature': 'ssar',
           'children': {'k': {'label': 'p'},
            's': {'label': 'e'},
            'y': {'label': 'e'}}}}}},
     'a': {'label': 'e'},
     'f': {'label': 'p'},
     'l': {'label': 'e'},
     's': {'label': 'p'},
     'y': {'label': 'p'},
     'p': {'label': 'p'},
     'c': {'label': 'p'},
     'm': {'label': 'p'}}}
```

[58]:
```
confusion = confusion_matrix(res_raw_tree, test_raw['class'])
report_tree=classification_report(test_raw['class'],res_raw_tree)
print(report_tree)
```

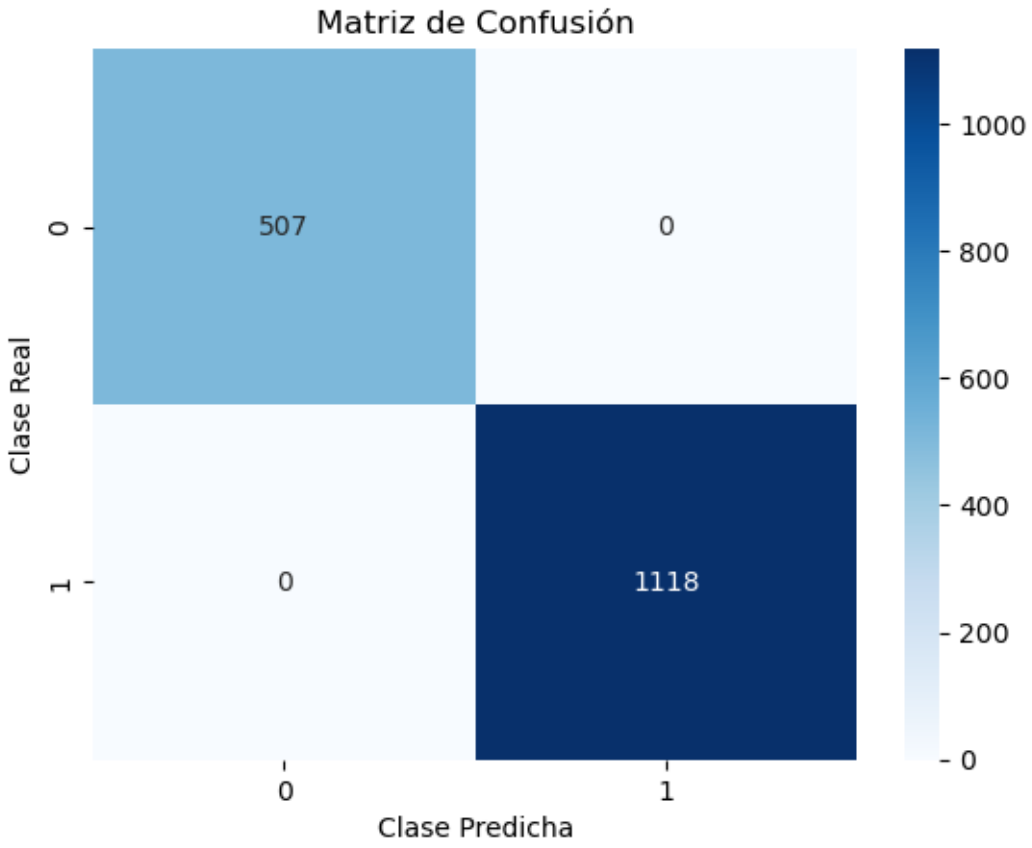
```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       507
           p       1.00      1.00      1.00      1118

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

[59]:
```
sns.heatmap(confusion,annot=True, fmt='d', cmap='Blues')
plt.xlabel('Clase Predicha')
plt.ylabel('Clase Real')
plt.title('Matriz de Confusión')
```

[59]:
```
Text(0.5, 1.0, 'Matriz de Confusión')
```



[60]:
```
confusion
```

[60]:
```
array([[ 507,    0],
       [   0, 1118]], dtype=int64)
```

[64]:
```
df['habitat'].value_counts()
```

[64]:
```
d    3148
g    2148
p    1144
l     832
u     368
m     292
w     192
Name: habitat, dtype: int64
```

[67]:
```
df['bruises'].value_counts()
```

[67]:
```
f    4748
t    3376
```

```
Name: bruises, dtype: int64
```

[ ]: