Taller Preprocesamiento - Minería de Datos - 2025 - I

Facultad de Ingeniería - Departamento de Ingeniería de Sistemas e Industrial

Estudiantes: Francisco José Salamanca Rivera, fsalamancar@unal.edu.co,
Daniel Mauricio Bonilla Muñoz, dabonilla@unal.edu.co,
David Camilo Gómez Medina, dcgomezme@unal.edu.co
Profesora: Elizabeth León Guzmán, eleonguz@unal.edu.co

ACTIVIDADES

1. Dado un conjunto de datos de una dimensión $X = \{-5.0,\ 23.0,\ 17.6,\ 7.23\ 1.11\}$, normalizar usando:

a) Min–max normalización en el intervalo $[0, 1]$.

b) Min–max normalización en el intervalo $[-1, 1]$.

c) Desviación estándar.

d) Escala decimal en el intervalo $[-1, 1]$.

e) Comparar los resultados y discutir ventajas y desventajas.

Solución

```python
import numpy as np
from sklearn.preprocessing import MinMaxScaler

v = np.array([-5.0, 23.0, 17.6, 7.23, 1.11])

# a) Normalizacion min-max con el intervalo [0,1]
v_n = (v - v.min()) / (v.max() - v.min())

print('Datos normalizados [0,1]:\n', v_n)
print()

# b) min-max intervalo [-1, 1]
v_n_sk = MinMaxScaler(feature_range=(-1, 1))
v_n_sk = v_n_sk.fit_transform(v.reshape(-1, 1))
v_n_sk = v_n_sk.T

print('Datos normalizados [-1, 1]:\n', v_n_sk)
print()

# c) Desviacion estandar
v_n_std = (v - v.mean()) / v.std()

print('Datos normalizados con desviacion estandar:\n', v_n_std)
print()

# d) Escala decimal en el intervalo [-1,1]
# Valor absoluto maximo
m_v = np.max(np.abs(v))
```

```
29
30 # j = numero de digitos
31 j = int(np.ceil(np.log10(m_v)))
32
33 v_n_dec = v / (10 ** j)
34
35 print('Datos normalizados con escala decimal:\n', v_n_dec)
36 print()
```

Listing 1: Solución Punto 1.

Salida del código:

```
Datos normalizados [0,1]: [0. 1. 0.80714286 0.43678571 0.21821429]
Datos normalizados [-1, 1]: [[-1. 1. 0.61428571 -0.12642857 -0.56357143]]
Datos normalizados con desviacion estandar: [-1.33779582 1.37893488
0.85499396 -0.15116666 -0.74496637]
Datos normalizados con escala decimal: [-0.05 0.23 0.176 0.0723 0.0111]
```

Ventajas y Desventajas

1. Min-Max [0,1]

Ventajas

- Conserva la forma de la distribución original (solo escala y traslada).
- Todos los valores quedan en un rango fijo [0,1], útil para redes neuronales con activaciones en ese intervalo.
- Fácil de interpretar: 0 = mínimo, 1 = máximo.

Desventajas

- Muy sensible a *outliers*: un valor extremo "estira" el rango y comprime el resto.

2. Min-Max [-1,1]

Ventajas

- Igual que [0,1], pero en un rango centrado en cero, lo cual favorece algunos algoritmos (p.ej. redes con activación *tanh*, modelos basados en distancias).
- Todos los valores quedan en un rango fijo [0,1], útil para redes neuronales con activaciones en ese intervalo.
- Fácil de interpretar: 0 = mínimo, 1 = máximo.

Desventajas

- Comparte la misma sensibilidad a *outliers* que el min–max [0,1].

3. Z-score (normalización por desviación estándar)

Ventajas

- Centra los datos en media 0 y varianza 1; los valores quedan en unidades de σ, lo que ayuda a comparar la "lejanía" de la media..
- Menos afectado por outliers extremos que el min–max, pues esos solo influyen en σ y μ.
- Los algoritmos basados en supuestos gaussianos (p.ej. regresión lineal, PCA, SVM) suelen mejorar su rendimiento.

Desventajas

- No acota los valores a un rango fijo: pueden surgir valores muy grandes o pequeños si hay *outliers*.
- La interpretación (unidades de desviación estándar) puede ser menos intuitiva.

4. Escala decimal)

Ventajas

- Normalización sencilla de calcular.
- Se garantiza que $\texttt{max}((|v'|)) < 1$, donde v' es la normalización decimal. Esto, sin necesidad de conocer distribución ni estadísticos.

Desventajas

- No tiene en cuenta la dispersión interna de los datos ni la forma de la distribución.
- Rango muy amplio de posibles escalas (p.ej. pasar de 0.07 a 0.001 si aumenta el máximo), poco control sobre cómo quedan los valores intermedios.
- Menos frecuente en algoritmos de ML modernos, que suelen preferir min–max o z-$score$.

¿Cuál elegir?

- Si se necesita un rango fijo y el dataset no tiene *outliers* fuertes, se puede utilizar **Min-Max**.
- Si el modelo se beneficia de datos centrados en 0, se puede hacer uso de **Min-Max [-1,1]**.
- Para algoritmos que asumen distribución gaussiana o requieren características con varianza unitaria, se puede utilizar $z - score$.
- Para casos muy rápidos y sencillos donde solo importa que $|v'| < 1$, se puede utilizar escala decimal.

2. Dado un conjunto de datos de 4 dimensiones con valores perdidos:

$I1$	$I2$	$I3$	$I4$
0	1	1	2
2	1	?	1
1	?	?	0
?	2	1	?
2	2	1	0

a) Dado que el dominio para todos los atributos es $[0, 1, 2]$, ¿cuál debe ser el número de ejemplos "artificiales" si los valores perdidos son interpretados como "no importa el valor" y ellos son reemplazados con todos los posibles valores para su dominio?

b) ¿Cuál otro método utilizaría para reemplazar los valores perdidos?

Solución

Solución a) Para la columna **I1**, el número de ejemplos artificiales es **3**, y son los siguientes:

- Ejemplo 1 de la columna I1: [0, 2, 1, 0, 2]
- Ejemplo 2 de la columna I1: [0, 2, 1, 1, 2]
- Ejemplo 3 de la columna I1: [0, 2, 1, 2, 2]

Para la columna **I2**, el número de ejemplos artificiales es **3**, y son los siguientes:

- Ejemplo 1 de la columna I2: [1, 1, 0, 2, 2]
- Ejemplo 2 de la columna I2: [1, 1, 1, 2, 2]
- Ejemplo 3 de la columna I2: [1, 1, 2, 2, 2]

Para la columna **I3**, el número de ejemplos artificiales es **9**, y son los siguientes:

- Ejemplo 1 de la columna I3: [1, 0, 0, 1, 1]
- Ejemplo 2 de la columna I3: [1, 0, 1, 1, 1]
- Ejemplo 3 de la columna I3: [1, 0, 2, 1, 1]
- Ejemplo 4 de la columna I3: [1, 1, 0, 1, 1]
- Ejemplo 5 de la columna I3: [1, 1, 1, 1, 1]
- Ejemplo 6 de la columna I3: [1, 1, 2, 1, 1]
- Ejemplo 7 de la columna I3: [1, 2, 0, 1, 1]
- Ejemplo 8 de la columna I3: [1, 2, 1, 1, 1]
- Ejemplo 9 de la columna I3: [1, 2, 2, 1, 1]

Para la columna **I4**, el número de ejemplos artificiales es **3**, y son los siguientes:

- Ejemplo 1 de la columna I4: [2, 1, 0, 0, 0]
- Ejemplo 2 de la columna I4: [2, 1, 0, 1, 0]
- Ejemplo 3 de la columna I4: [2, 1, 0, 2, 0]

En total, el número de ejemplos artificiales para todas las columnas es **18**.
Solución b) Otro método para reemplazar valores perdidos es asignarles una **constante global**, como un valor fijo previamente definido. Sin embargo, esta estrategia depende del contexto y de la aplicación que el usuario le dará a los datos. Otra técnica común es reemplazar los valores perdidos con la **media** (si el atributo es numérico) o la **moda** (si es categórico) del atributo correspondiente. Además, existen métodos más avanzados, como utilizar modelos de **predicción** (por ejemplo, regresión o árboles de decisión) que estiman los valores faltantes a partir de la información de otros atributos.

3. El número de hijos de diferentes pacientes es dado por el siguiente vector:
$C = \{3, 1, 0, 2, 7, 3, 6, 4, -2, 0, 0, 10, 15, 6\}$.

a) Encontrar "outliers" usando parámetros estadísticos estándar: media y varianza.

1) Si el umbral cambia de ± 3 desviaciones estándar a ± 2 desviaciones estándar, ¿cuál "*outlier* adicional" se encuentra?

Solución

```python
import numpy as np

# Datos
C = np.array([3, 1, 0, 2, 7, 3, 6, 4, -2, 0, 0, 10, 15, 6])

# Cálculo de media y desviacion estandar
mean = C.mean()
std = C.std(ddof=1)
print(f'Media: {mean:.4f}')
print(f'Desviacion estandar: {std:.4f}\n')

# Umbral +- 3 desv estandar
lower_3 = mean - 3 * std
upper_3 = mean + 3 * std
outliers_3 = C[(C < lower_3) | (C > upper_3)]
print('Outliers con umbral +- 3 desv est:', outliers_3)

# Umbral +- 2 desv estandar
lower_2 = mean - 2 * std
upper_2 = mean + 2 * std
outliers_2 = C[(C < lower_2) | (C > upper_2)]
print('Outliers con umbral +- 2 desv est:', outliers_2)

# Outlier adicional al cambiar de +- 3 desv est a +- 2 desv est
```

```
additional = np.setdiff1d(outliers_2, outliers_3)
print('\n Outlier adicional encontrado al usar +- 2 desv est:', additional)
```

Salida del código:

```
Media: 3.9286
Desviacion estandar: 4.5820

Outliers con umbral +- 3 desv est: []
Outliers con umbral +- 2 desv est: [15]

Outlier adicional encontrado al usar +- 2 desv est: [15]
```

4. Dado un conjunto de tres dimensiones,
$X = [\{1,2,0\},\{3,1,4\},\{2,1,5\},\{0,1,6\},\{2,4,3\},\{4,4,2\},\{5,2,1\},\{7,7,7\},\{0,0,0\},\{3,3,3\}]$

a) Describir el procedimiento e interpretar los resultados de detección de *outliers* basado en la media y varianza.

Solución

Siguiendo el planteamiento del ejercicio anterior, se decide utilizar 2 veces la desviación estándar ($\mathbf{2}\sigma$) para el análisis de los *outliers*. Esto, con el fin de poder conseguir por lo menos, un dato atípico.

```
import numpy as np

# Datos proporcionados
X = [
    [1, 2, 0],
    [3, 1, 4],
    [2, 1, 5],
    [0, 1, 6],
    [2, 4, 3],
    [4, 4, 2],
    [5, 2, 1],
    [7, 7, 7],
    [0, 0, 0],
    [3, 3, 3]
]

# Se convierte a array de numpy para facilitar calculos
data = np.array(X)

# Calculo de la media y la desv estandar
means = np.mean(data, axis = 0)
stds = np.std(data, axis = 0)

# Limite sup e inf desv estandar
```

```
low_b = means - 2 * stds
upp_b = means + 2 *stds

# Identificar outliers
outliers = []
for i, point in enumerate(data):
  outlier_dims = []
  for dim in range(0,3):
    if not (low_b[dim] <= point[dim] <= upp_b[dim]):
      outlier_dims.append(dim + 1)
  if outlier_dims:
    outliers.append({"punto": point.tolist(),
                     "indice": i,
                     "dimensiones": outlier_dims,
                     "motivo": f"Fuera de rango en dimensión(es)
                     {outlier_dims}"})
# Mostrar resultados
print("=== Estadísticas por dimensión ===")
for dim in range(3):
    print(f"Dimensión {dim + 1}:")
    print(f"  Media: {means[dim]:.2f}")
    print(f"  Desviación: {stds[dim]:.2f}")
    print(f"  Rango aceptable: [{low_b[dim]:.2f}, {upp_b[dim]:.2f}]\n")

print("\n=== Resultados de detección ===")
if outliers:
    for outlier in outliers:
        print(f"Punto {outlier['punto']} (índice {outlier['indice']}):")
        print(f"  - {outlier['motivo']}")
else:
    print("No se encontraron outliers")
```

Salida del código:

```
=== Estadísticas por dimensión ===
Dimensión 1:
Media: 2.70
Desviación: 2.10
Rango aceptable: [-1.50, 6.90]

Dimensión 2:
Media: 2.50
Desviación: 1.96
Rango aceptable: [-1.42, 6.42]

Dimensión 3:
Media: 3.10
Desviación: 2.30
```

```
Rango aceptable: [-1.50, 7.70]

=== Resultados de detección ===
Punto [7, 7, 7] (índice 7):
- Fuera de rango en la dimensión(es) [1, 2]
```
Esto, refleja que, los datos atípicos son el 7, para las dimensiones 1 y 2.

5. En Weka cargar el conjunto de datos iris

a) Eliminar manualmente valores (15 %) en sus atributos, para simular valores perdidos. Luego aplicar varios métodos que estan en weka para remplazar esos valores perdidos. Discutir las diferencias entre el valor real y el que valor que lo remplaza, y las diferencias entre los métodos.

b) Normalizar usando varios métodos.

c) Discretizar usando varios métodos.

Solución

Solución a)

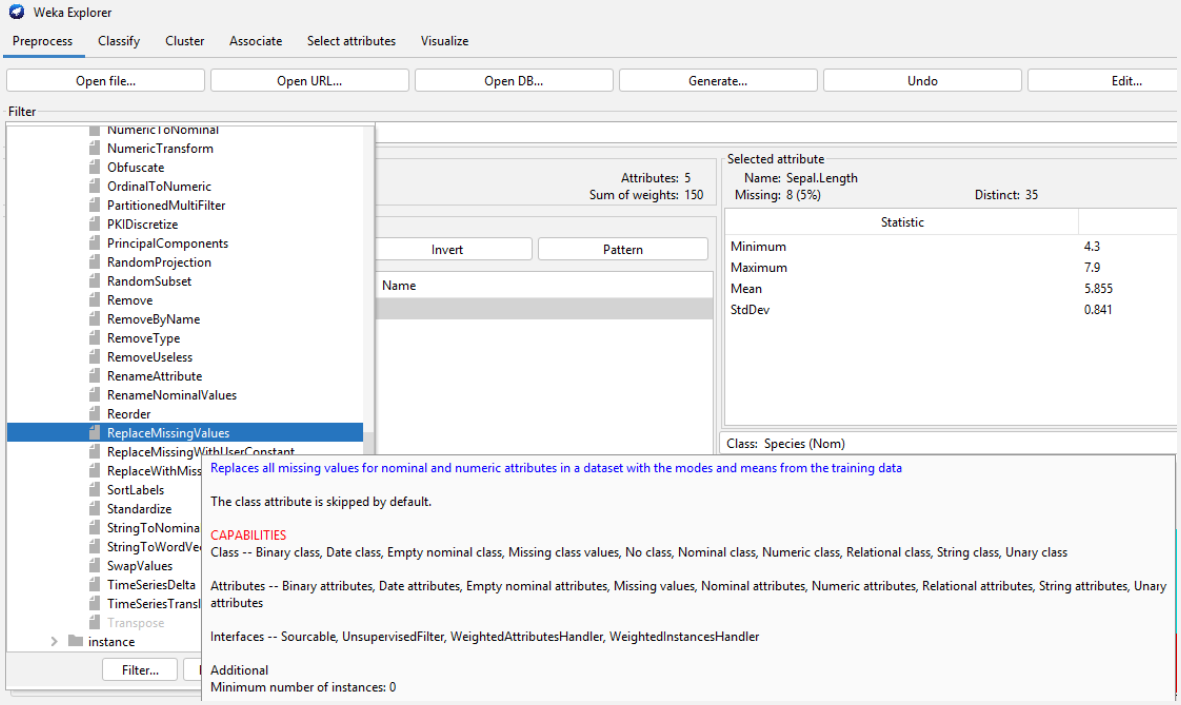

Figura 1: Captura de pantalla usando el método ReplaceMissingValues.

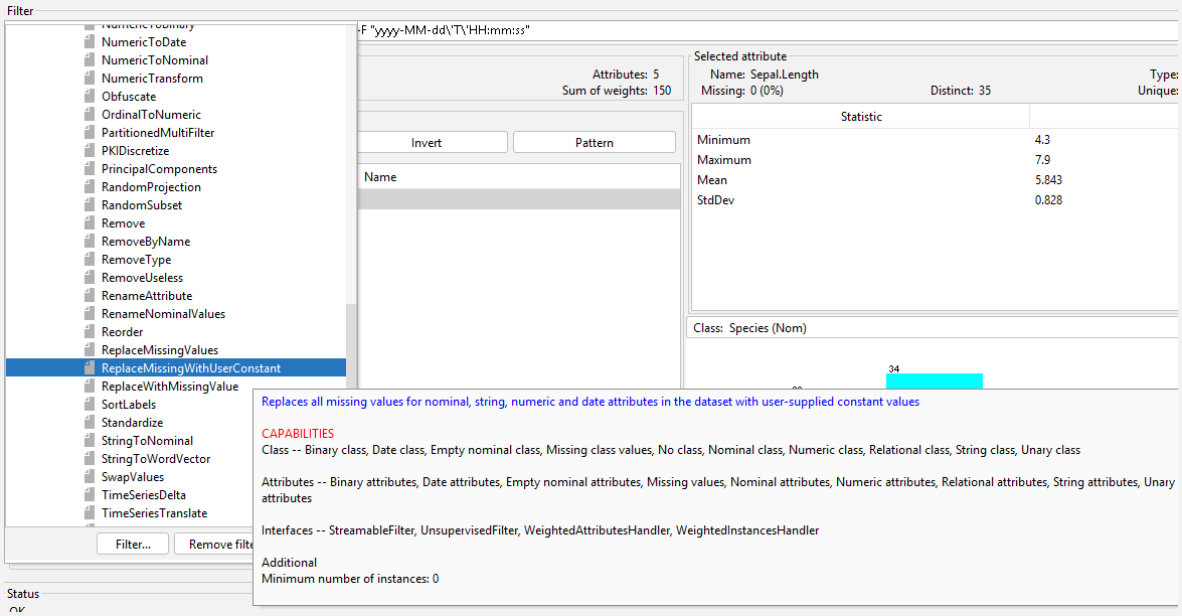

Figura 2: Captura de pantalla usando el método ReplaceMissingWithUserConstant.

Al comparar los valores reales con los valores que se reemplazan mediante los métodos de Weka, se observa que rara vez coinciden de forma exacta.

El método ReplaceMissingValues sustituye los valores perdidos por la media (en atributos numéricos) o la moda (en atributos nominales), lo que permite mantener cierta coherencia con la distribución general del conjunto de datos, aunque puede reducir la variabilidad y alejarse del valor específico que se eliminó.

Por otro lado, el método ReplaceMissingWithUserConstant reemplaza todos los valores perdidos con un valor fijo definido por el usuario, lo que resulta en una solución simple pero menos representativa, ya que no tiene en cuenta la distribución real de los datos. La principal diferencia entre ambos métodos es que ReplaceMissingValues adapta los valores de forma dinámica según el contenido del dataset, mientras que ReplaceMissingWithUserConstant aplica un único valor igual para todos los casos, lo cual puede introducir un sesgo mayor si no se elige cuidadosamente el valor constante.

Solución b)

Método Normalize

El método Normalize se utiliza para escalar los valores numéricos de los atributos dentro de un rango entre 0 y 1. Este proceso es importante porque permite que todos los atributos tengan la misma escala, evitando que aquellos con valores más grandes dominen sobre los más pequeños durante el análisis.

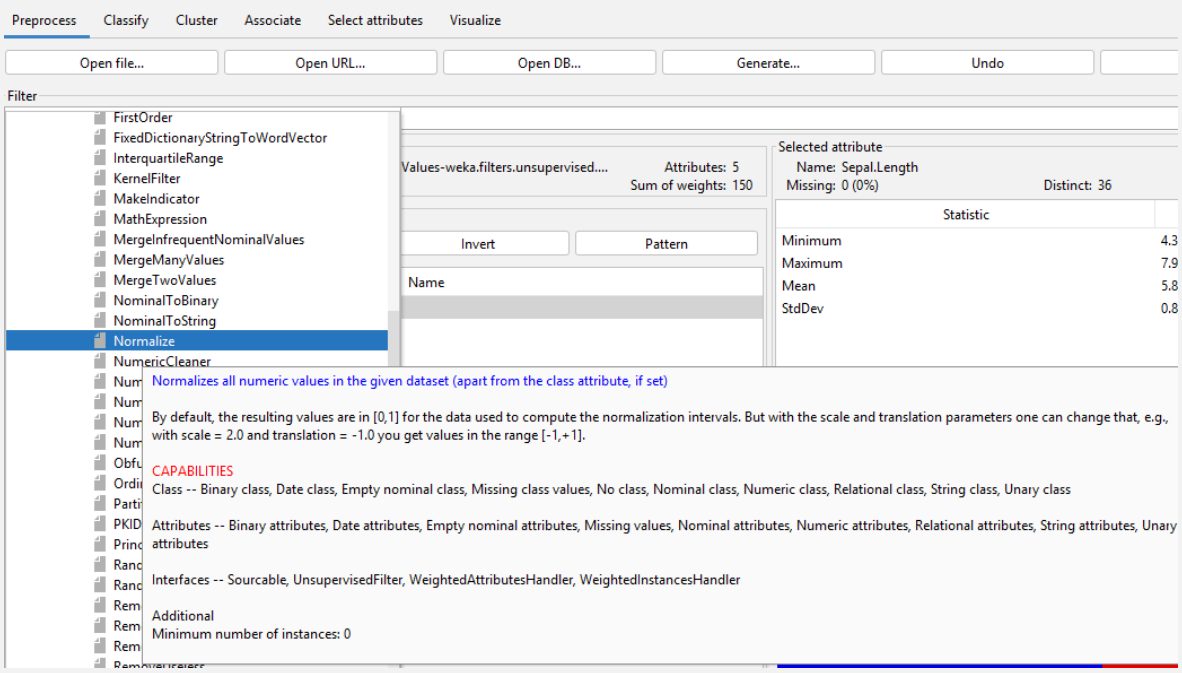

Figura 3: Captura de pantalla usando el método normalize.

Para nuestro conjunto de datos *Iris*, los nuevos valores de la media y la desviación estándar de las variables, después de aplicar los métodos, son los siguientes:

- **Sepal length**: media = 0.429, desviación estándar = 0.23
- **Sepal width**: media = 0.439, desviación estándar = 0.181
- **Petal length**: media = 0.468, desviación estándar = 0.299
- **Petal width**: media = 0.458, desviación estándar = 0.318

Método Standardize

El método Standardize transforma los valores numéricos de los atributos para que tengan una media de 0 y una desviación estándar de 1. Este proceso, se conoce como estandarización o normalización Z-score.

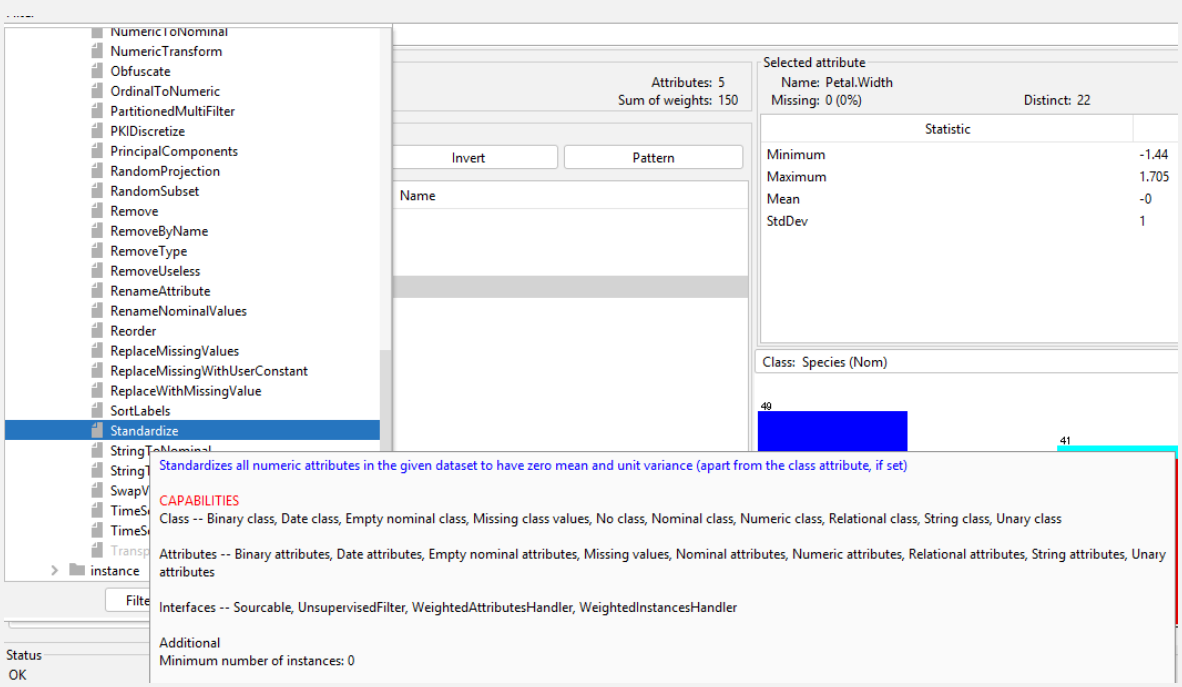

Figura 4: Captura de pantalla usando el método standardize.

Después de aplicar el método los valores de la media y desviación estandar para los cuatro atributos es de 0 y 1 respectivamente.

Solución c)

Método Discretize

El método Discretize en Weka convierte atributos numéricos en valores categóricos o intervalos, dividiendo el rango de datos en un número específico de grupos llamados bins. Cada bin representa un rango de valores y es tratado como una categoría

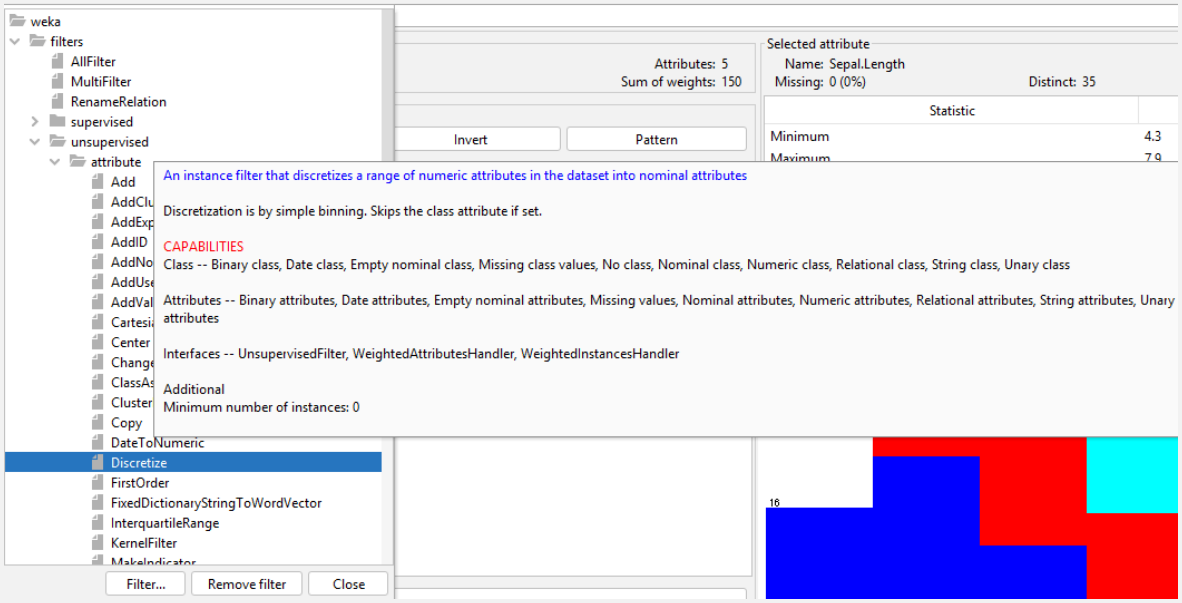

Figura 5: Captura de pantalla usando el método Discretize.

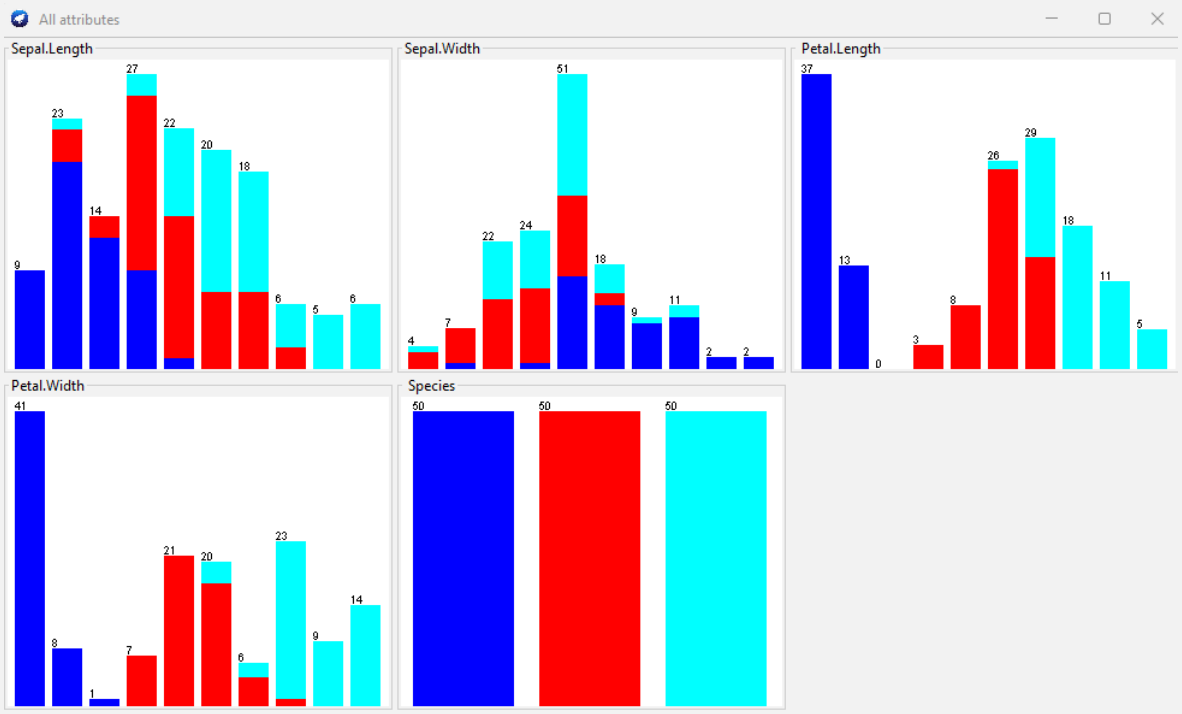

Figura 6: Gráfico de barras con las nuevas categorias después de aplicar el método discretize.

Por defecto, el método Discretize utiliza 10 bins para crear las nuevas categorías, en la siguiente figura se puede observar un gráfico de barras para las nuevas categorías en cada atributo.

Método NumericToNomial

El método NumericToNominal en Weka convierte atributos numéricos en categorías o etiquetas. Esto es útil cuando los números no tienen un valor continuo, sino que representan diferentes grupos o clases

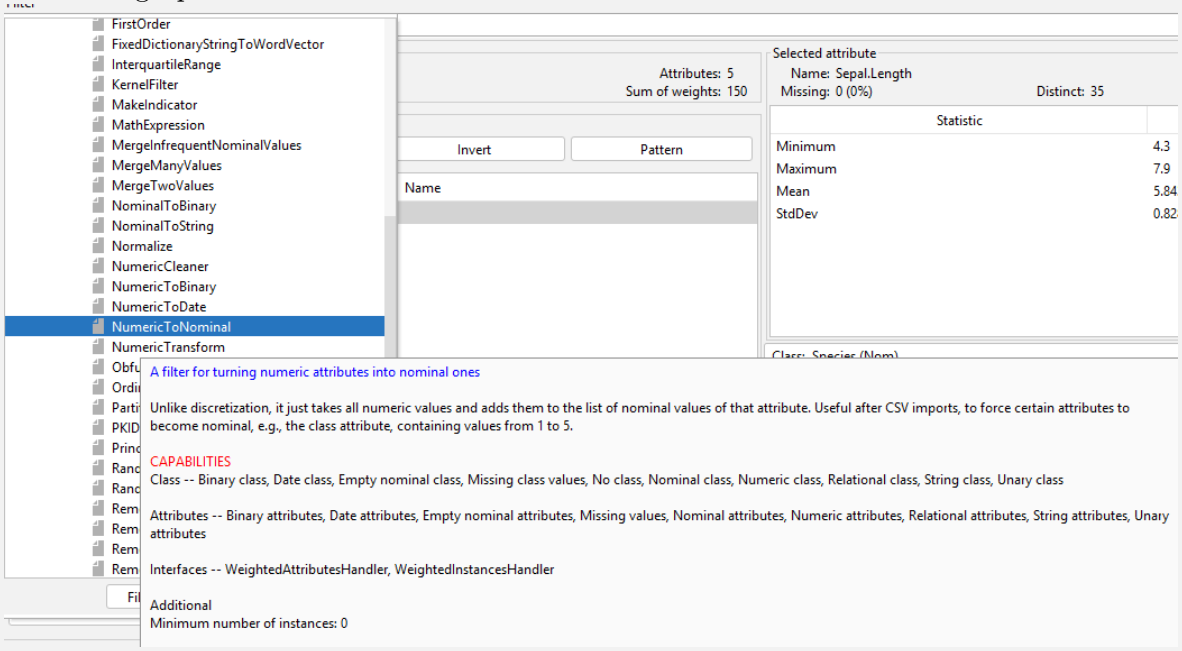

Figura 7: Captura de pantalla usando el método NumericToNominal.

6. Dado el conjunto de datos: Realizar reducción de valores basado en la técnica de BIN con el mejor corte para lo siguiente (mostrar pasos):

$I1$	$I2$	$I3$
1	5.9	3.4
2	2.1	6.2
1	1.6	2.8
2	6.8	5.8
1	3.1	3.1
1	8.3	4.1
2	2.4	5.0

a) Dimensión I2 usando la media como representantes de 2 BINS

b) Dimensión I3 usando el limite más cercano como representante de 2 BINS

```
import numpy as np
punto6= np.array([["I1","I2","I3"],
                  [1,5.9,3.4],
                  [2,2.1,6.2],
                  [1,1.6,2.8],
                  [2,6.8,5.8],
                  [1,3.1,3.1],
                  [1,8.3,4.1],
                  [2,2.4,5.0]])



#a. Dimensión I2 usando la media como representantes de 2 BINS (Separar dos bins
    mediante la media)

punto6a=punto6[1:,1]
punto6a=np.sort(punto6a)
punto6a = np.array(punto6a, dtype=float)
mean=np.mean(punto6a)

#Separar bins segun las medias
bin1=punto6a[punto6a <= mean]
bin2=punto6a[punto6a > mean]

mean_bin1=np.mean(bin1)
mean_bin2=np.mean(bin2)

#reemplaza aquellos menores a la media por el valor de bin1, y el resto con bin2
I2=np.where(punto6a <= mean, mean_bin1, mean_bin2)

print(I2)

#otra manera usando numpy
bins=np.digitize(punto6a,bins=[mean])
print(bins)


#b.Dimension I3 usando el limite más cercano como representante de 2 BINS

```

```python
#separar en 2 bins, pero el valor se reemplazara por el limite mas cercano, ya sea el superor o el inferior del bin

punto6b=punto6[1:,1]
punto6b=np.sort(punto6b)
punto6b=np.array(punto6b, dtype=float)
meanb=np.mean(punto6b)

bin1b=punto6b[punto6b <= meanb]
bin2b=punto6b[punto6b > meanb ]

#bin1
lim_sup1=np.max(bin1b)
lim_inf1=np.min(bin1b)

#bin2
lim_sup2=np.max(bin2b)
lim_inf2=np.min(bin2b)

#Funcion reemplazo
def reemplazo_limite(valor):
    if valor <= mean:
        return lim_inf1 if abs(valor - lim_inf1) <= abs(valor - lim_sup1) else lim_sup1
    else:
        return lim_inf2 if abs(valor - lim_inf2) <= abs(valor - lim_sup2) else lim_sup2

I3 = np.array([reemplazo_limite(v) for v in punto6b])
print(I3)
```

Listing 2: Punto 6.

Salida del código:

- `print(I2)`: Imprime los valores de la dimensión I2 una vez reemplazados por la media del bin correspondiente. Los valores menores o iguales a la media general se reemplazan por la media de su grupo (bin1) y los mayores por la media del otro grupo (bin2).

  I2 = [2.3 , 2.3 , 2.3 , 2.3 , 7. , 7. , 7.]

- `print(I3)`: Imprime los valores de la dimensión I3 reemplazados por el límite más cercano (inferior o superior) del bin correspondiente, en lugar de por la media.

  I3 = [1.6, 1.6, 3.1, 3.1 , 5.9, 5.9, 8.3]

7. Dado el conjunto de datos con tres dimensiones de entrada y una dimension representando la clase:

$I1$	$I2$	$I3$	C
2.5	1.6	5.9	0
7.2	4.3	2.1	1
3.4	5.8	1.6	1
5.6	3.6	6.8	0
4.8	7.2	3.1	1
8.1	4.9	8.3	0
6.3	4.8	2.4	1

Hacer el ranking de las dimensiones realizando comparación de medias y varianzas.

```
import numpy as np
punto7= np.array([["I1","I2","I3", "C"],
                  [2.5,1.6,5.9,0],
                  [7.2,4.3,2.1,1],
                  [3.4,5.8,1.6,1],
                  [5.6,3.6,6.8,0],
                  [4.8,7.2,3.1,1],
                  [8.1,4.9,8.3,0],
                  [6.3,4.8,2.4,1]])


I1= punto7[1:,0]
I1=np.array(I1, dtype=float)
I2= punto7[1:,1]
I2=np.array(I2, dtype=float)
I3= punto7[1:,2]
I3=np.array(I3, dtype=float)
I4= punto7[1:,3]


print(I1)
print(I2)
print(I3)


#Varianzas
var=[]
I1_var= np.var(I1)
var.append(I1_var)
I2_var= np.var(I2)
var.append(I2_var)
I3_var= np.var(I3)
var.append(I3_var)

print(var)
sorted_var=np.sort(var)
print(sorted_var)

print(var)


print(f"El ranking de las clases segun varianzas es I2, I1, I3")

#medias
med=[]
I1_var= np.median(I1)
med.append(I1_var)
I2_var= np.median(I2)
med.append(I2_var)
I3_var= np.median(I3)
med.append(I3_var)

print(med)
sorted_med=np.sort(med)
print(sorted_med)

```

```
57 print(f"El ranking de las clases segun medias es I1, I2, I3")
```

Listing 3: Punto 7.

Salida del código:

- `print(var)`: [3.449795918367347, 2.6314285714285712, 5.9983673469387755].
- `print(var)`: El ranking de las clases segun las varianzas es I2,I1,I3.
- `print(med)`: [np.float64(5.6), np.float64(4.8), np.float64(3.1)]
- `print(b)`: El ranking de las clases segun las varianzas es I1,I2,I3.

8. Dado el conjunto de datos X, donde X1 y X2 son dimensiones numericas, X3 y X4 son dimensiones con datos categoricos.

$X1$	$X2$	$X3$	$X4$
2.7	3.4	1	A
3.1	6.2	2	A
4.5	2.8	1	B
5.3	5.8	2	B
6.6	3.1	1	A
5.0	4.1	2	B

a) Aplicar el método se selección de características basado en la entropía para reducir una dimensión (mostrar pasos).

b) Implementar un programa para realizar el "ranking" de dimensiones usando entropía.

Solución

a) Similaridades:

	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
$R1$		0.25	0.25	0	0.5	0
$R2$			0	0.25	0.25	0.25
$R3$				0.25	0.25	0.25
$R4$					0	0.5
$R5$						0
$R6$						

Calculamos la entropía de todo el conjunto de datos con la siguiente formula:
$E = -\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(S_{ij}\cdot\log(S_{ij}) + (1-S_{ij})\cdot\log(1-S_{ij}))$
$E = 5.88$
Descartamos la siguiente variable X1, y calculamos nuevamente la entropía:

	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
$R1$		0.33	0.33	0	0.66	0
$R2$			0	0.33	0.33	0.33
$R3$				0.33	0.33	0.33
$R4$					0	0.66
$R5$						0
$R6$						

$E_{f1} = 6.36$
$E - E_{f1} = 5.88 - 6.36 = -0.48$
Descartamos X2:

	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
$R1$		0.33	0.33	0	0.66	0
$R2$			0	0.33	0.33	0.33
$R3$				0.33	0.33	0.33
$R4$					0	0.66
$R5$						0
$R6$						

$E_{f2} = 6.36$
$E - E_{f2} = 5.88 - 6.36 = -0.48$
Descartamos X3:

	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
$R1$		0.33	0	0	0.33	0
$R2$			0	0	0.33	0
$R3$				0.33	0	0.33
$R4$					0	0.33
$R5$						0
$R6$						

$E_{f3} = 3.81$ $E - E_{f3} = 5.88 - 3.81 = 2.06$
Descartamos X4:

	$R1$	$R2$	$R3$	$R4$	$R5$	$R6$
$R1$		0	0.33	0	0.33	0
$R2$			0	0.33	0	0.33
$R3$				0	0.33	0
$R4$					0	0.33
$R5$						0
$R6$						

$E_{f4} = 3.81$ $E - E_{f4} = 5.88 - 3.81 = 2.06$
Ranking

a) Descartando x_1 $(\Delta E = -0.48)$

b) Descartando x_2 $(\Delta E = -0.48)$

c) Descartando x_4 $(\Delta E = 2.06)$

d) Descartando x_3 $(\Delta E = 2.06)$

Se puede reducir la dimensionalidad del conjunto de datos eliminando la variable X1 o X2 porque, al hacerlo, la entropía total del sistema no disminuye significativamente. De hecho, la entropía aumenta ligeramente , lo que indica que estas variables no están aportando información relevante al conjunto de datos.

b)

```
    import numpy as np

def hamming_similarity(matrix):
    # Obtener el número de filas
    n_rows = matrix.shape[0]

    # Inicializamos una matriz para almacenar las similitudes de Hamming
    similarity_matrix = np.zeros((n_rows, n_rows))

    # Comparamos todas las filas
    for i in range(n_rows):
        for j in range(i, n_rows):  # Solo comparar una vez para evitar duplicados
            # Calculamos la cantidad de coincidencias en las filas
            hamming_distance = np.sum(matrix[i] != matrix[j])
            # La similitud de Hamming es el complemento de la distancia de Hamming
            similarity = 1 - (hamming_distance / len(matrix[i]))
            similarity_matrix[i, j] = similarity
            similarity_matrix[j, i] = similarity  # Simetría de la matriz

    return similarity_matrix

def binary_entropy_matrix(S):
    """Calcula la entropía total de una matriz S con la fórmula dada."""
    E = 0.0
    n = S.shape[0]
    for i in range(n - 1):
        for j in range(i + 1, n):
            sij = S[i][j]

            if sij in [0, 1]:
                term = 0  # Evita log(0) que es indefinido
            else:
                term = sij * np.log(sij) + (1 - sij) * np.log(1 - sij)

            E += term
    return -E

def remove_column_and_recalculate(S, col_index):
    """Elimina una columna y su fila correspondiente de la matriz de similitud."""
```

```python
    S_reduced = np.delete(S, col_index, axis=1)
     # elimina columna
    S_reduced = hamming_similarity(S_reduced)
    return S_reduced

def rank_features_by_entropy(S):
    s= hamming_similarity(S)
    total_entropy = binary_entropy_matrix(s)
    print(f"Entropía total del conjunto original: {total_entropy:.4f}\n")
    rankings = []


    for i in range(S.shape[1]):
        S_reduced = remove_column_and_recalculate(S, i)
        entropy_reduced = binary_entropy_matrix(S_reduced)
        delta_entropy = total_entropy - entropy_reduced
        rankings.append((f"x{i+1}", delta_entropy))


    # Ordenamos de menor a mayor contribución (menos importante primero)
    rankings.sort(key=lambda x: x[1])

    print("Ranking de variables (de menor a mayor aporte de información):\n")
    for var, delta in rankings:
        print(f"{var}: E = {delta:.4f}")


X= np.array([
    [2.7, 3.4, 1, 'A'],
    [3.1, 6.2, 2,'A'],
    [4.5, 3.8, 1,'B'],
    [5.3,5.8,2,'B'],
    [6.6,3.1,1,'A'],
    [5,4.1,2,'B'],
])



rank_features_by_entropy(X)
```

```
Entropía total del conjunto original: 5.8850

Ranking de variables (de menor a mayor aporte de información):

x1: ΔE = -0.4802
x2: ΔE = -0.4802
x3: ΔE = 2.0659
x4: ΔE = 2.0659
```

Figura 8: Salida rank$_f eatures_b y_e ntropy(X)$.

9. Al conjunto de datos Adult del repositorio de Machine Learning:

a) Convertir todos los atributos numéricos a categóricos utilizando dos estrategias diferentes.

a) Transformar el conjunto de datos de manera que todos los atributos sean numéricos.

Solución

a) La primer estrategia a implementar, es el uso del paquete `sklearn`. Se hizo uso del módulo `preprocessing` y de la clase `KBinsDiscretizer`. El código se detalla a continuación:

```
pip install ucimlrepo

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from ucimlrepo import fetch_ucirepo

# fetch dataset
adult = fetch_ucirepo(id=2)

# data (as pandas dataframes)
X = adult.data.features
y = adult.data.targets

# metadata
print(adult.metadata)

# variable information
print(adult.variables)

from sklearn.preprocessing import KBinsDiscretizer

numeric_cols = ['age', 'fnlwgt', 'education-num', 'capital-gain',
'capital-loss', 'hours-per-week']

# Crea copia del DataFrame original para no modificarlo
X_cat_strat1 = X.copy()

# Aplica discretización
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')
X_cat_strat1[numeric_cols] = discretizer.fit_transform(X[numeric_cols])

# Renombrar categorías (opcional)
for col in numeric_cols:
    X_cat_strat1[col] = X_cat_strat1[col].map({0: 'low', 1: 'medium',
    2: 'high'})
```

Además, se realizó el mismo método de *Binning* pero sin el apoyo de la clase

`KBinsDiscretizer`. Esto, con la finalidad de comprender mejor la metodología.

```
# columna 'age'
bins = [0, 30, 60, 100]
labels = ['young', 'adult', 'senior']
X_cat_strat2 = X.copy()
X_cat_strat2['age'] = pd.cut(X['age'], bins=bins, labels=labels)

# columna 'education num'
bins = [0, 8, 12, X['education-num'].max()]
labels = ['elementary', 'middle', 'high']
X_cat_strat2['education-num'] = pd.cut(X['education-num'], bins=bins,
labels=labels)

#columna 'capital gain'
bins = [-1, 10000, 50000, X['capital-gain'].max()]
labels = ['low', 'medium', 'high']
X_cat_strat2['capital-gain'] = pd.cut(X['capital-gain'], bins=bins,
labels=labels)

#columna 'capital loss'
bins = [-1, 1000, 3000, X['capital-loss'].max()]
labels = ['low', 'medium', 'high']
X_cat_strat2['capital-loss'] = pd.cut(X['capital-loss'], bins=bins,
labels=labels)

#columna 'hours per week'
bins = [0, 30, 40, X['hours-per-week'].max()]
labels = ['part-time', 'full-time', 'overtime']
X_cat_strat2['hours-per-week'] = pd.cut(X['hours-per-week'], bins=bins,
labels=labels)
```

La segunda metodología para abordar el problema, fue la Binaria. A cotinuación, se detalla el código:

```
X_cat_strat3 = X.copy()

# 'age': binario basado en un umbral
X_cat_strat3['age'] = np.where(X['age'] < 40, 'young', 'adul')

# 'capital-gain': binario basado en la media
mean_gain = X['capital-gain'].mean()
X_cat_strat3['capital-gain'] = np.where(X['capital-gain'] <= mean_gain,
'low', 'high')

# Aplica a todas las columnas numéricas
for col in numeric_cols:
```

```
    if col not in ['age', 'capital-gain']:  # Evita columnas ya procesadas
        threshold = X[col].median()  # Usa mediana como umbral
        X_cat_strat3[col] = np.where(X[col] <= threshold, 'low', 'high')

X_cat_strat3[numeric_cols] = X_cat_strat3[numeric_cols].astype('category')
```

b) La transformación de todos los atributos a numéricos, se hizo por medio de la metodología *One Hot Encoding*.

```
from sklearn.preprocessing import OneHotEncoder

# Codificar variables categóricas originales
encoder = OneHotEncoder(drop='first', sparse_output=False)
X_encoded = pd.DataFrame(
    encoder.fit_transform(X.select_dtypes(include='object')),
    columns=encoder.get_feature_names_out()

# Unir columnas numéricas originales y categóricas codificadas
X_numeric = pd.concat([X.select_dtypes(exclude='object'), X_encoded], axis=1)
```

Resultados:

```
# Para el inciso a)
print("Estrategia 1 (cuantiles):")
print(X_cat_strat1.dtypes) #Todas las columnas deben ser 'object' o 'category'

print("\nEstrategia 2 (Binaria):")
print(X_cat_strat3[numeric_cols].dtypes)

# Para el inciso b)
print("\nDataset totalmente numérico:")
print(X_numeric.dtypes)     # Todas las columnas deben ser 'float64' o 'int64'
```

```
Estrategia 1 (cuantiles):
age               object
workclass         object
fnlwgt            object
education         object
education-num     object
marital-status    object
occupation        object
relationship      object
race              object
sex               object
capital-gain      object
capital-loss      object
hours-per-week    object
native-country    object
dtype: object

Estrategia 2 (Binaria):
age               category
fnlwgt            category
education-num     category
capital-gain      category
capital-loss      category
hours-per-week    category
dtype: object

Dataset totalmente numérico:
age                                int64
fnlwgt                             int64
education-num                      int64
capital-gain                       int64
capital-loss                       int64
                                   ...
native-country_Trinadad&Tobago     float64
native-country_United-States       float64
native-country_Vietnam             float64
native-country_Yugoslavia          float64
native-country_nan                 float64
Length: 103, dtype: object
```

Figura 9: Verificación de los tipos de datos en las estrategias 1 y 2 del inciso a) y del inciso b).

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	medium	State-gov	low	Bachelors	high	Never-married	Adm-clerical	Not-in-family	White	Male	low	low	medium	United-States
1	high	Self-emp-not-inc	low	Bachelors	high	Married-civ-spouse	Exec-managerial	Husband	White	Male	low	low	low	United-States
2	medium	Private	high	HS-grad	medium	Divorced	Handlers-cleaners	Not-in-family	White	Male	low	low	medium	United-States
3	high	Private	high	11th	low	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	low	low	medium	United-States
4	low	Private	high	Bachelors	high	Married-civ-spouse	Prof-specialty	Wife	Black	Female	low	low	medium	Cuba

Figura 10: Salida primera metodología inciso a) - KBinsDiscretizer.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	adult	State-gov	77516	Bachelors	high	Never-married	Adm-clerical	Not-in-family	White	Male	low	low	full-time	United-States
1	adult	Self-emp-not-inc	83311	Bachelors	high	Married-civ-spouse	Exec-managerial	Husband	White	Male	low	low	part-time	United-States
2	adult	Private	215646	HS-grad	middle	Divorced	Handlers-cleaners	Not-in-family	White	Male	low	low	full-time	United-States
3	adult	Private	234721	11th	elementary	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	low	low	full-time	United-States
4	young	Private	338409	Bachelors	high	Married-civ-spouse	Prof-specialty	Wife	Black	Female	low	low	full-time	Cuba

Figura 11: Salida primera metodología inciso a) - Manual.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	young	State-gov	low	Bachelors	high	Never-married	Adm-clerical	Not-in-family	White	Male	high	low	low	United-States
1	adul	Self-emp-not-inc	low	Bachelors	high	Married-civ-spouse	Exec-managerial	Husband	White	Male	low	low	low	United-States
2	young	Private	high	HS-grad	low	Divorced	Handlers-cleaners	Not-in-family	White	Male	low	low	low	United-States
3	adul	Private	high	11th	low	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	low	low	low	United-States
4	young	Private	high	Bachelors	high	Married-civ-spouse	Prof-specialty	Wife	Black	Female	low	low	low	Cuba

Figura 12: Salida segunda metodología inciso a) - Binaria.

10. Escoger un conjunto de datos del repositorio de Machine learning, que tenga varias dimensiones y que sean numéricas, y aplicar PCA. Describir el nuevo conjunto de datos.

Hacer el ranking de las dimensiones realizando comparación de medias y varianzas.

```
import pandas as pd
import numpy as np
import random as rd
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine


wine = load_wine()
wine_df = pd.DataFrame(wine.data, columns=wine.feature_names)

wine_df.head()

scaled_wine = preprocessing.scale(wine_df)
pca = PCA()
pca.fit(scaled_wine)
pca_data = pca.transform(scaled_wine)

#Calcular el porcentaje de variacion de cada PC
per_var = np.round(pca.explained_variance_ratio_* 100, decimals=1)
labels = ['PC' + str(x) for x in range(1, len(per_var)+1)] #Etiquetas


#Crear grafico
plt.bar(x=range(1,len(per_var)+1), height=per_var, tick_label=labels)
plt.ylabel('Porcentaje de varianza explicada')
plt.xlabel('Componentes principales')
plt.title('Porcentaje de varianza explicada por cada PC')
plt.show()

#Se observa que la variacion se condensa principalmente en los 3 primeros PC
```

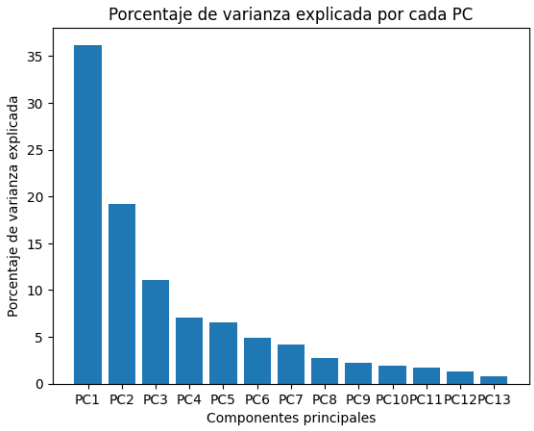

Figura 13: Porcentaje de varianza explicada por cada componente principal

```

import matplotlib.pyplot as plt

# Crear DataFrame
pca_df = pd.DataFrame(pca_data, columns=labels)
pca_df['target'] = wine.target

# Colores por clase
colors = ['red', 'green', 'blue']
labels_unique = wine.target_names

plt.figure(figsize=(8, 6))

for i, label in enumerate(np.unique(wine.target)):
    subset = pca_df[pca_df['target'] == label]
    plt.scatter(subset['PC1'], subset['PC2'], color=colors[i], label=labels_unique
    [i], s=80)

    # Anotaciones opcionales
    for idx in subset.index:
        plt.annotate(wine.target[idx], (pca_df.PC1[idx], pca_df.PC2[idx]),
    fontsize=8, alpha=0.5)

plt.title('PCA de Wine Dataset')
plt.xlabel('PC1 - {:.2f}%'.format(per_var[0]))
plt.ylabel('PC2 - {:.2f}%'.format(per_var[1]))
plt.legend(title='Clase')
plt.show()
```

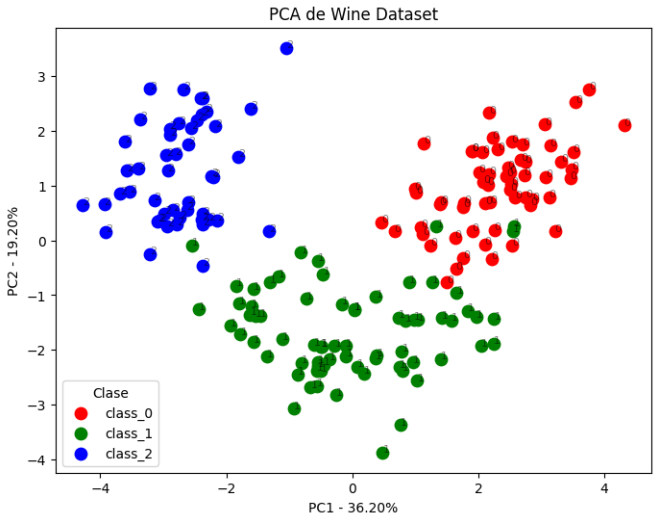

Figura 14: PCA del dataset Wine

```python
loading_scores = pd.Series(pca.components_[0], index=wine_df.columns)
sorted_loading_scores = loading_scores.abs().sort_values(ascending=False)


#Top de las variables que mas contribuyeron a la separacion de las variables
top_10_variables = sorted_loading_scores.head(10)
print(top_10_variables)
```

```
flavanoids                      0.422934
total_phenols                   0.394661
od280/od315_of_diluted_wines    0.376167
proanthocyanins                 0.313429
nonflavanoid_phenols            0.298533
hue                             0.296715
proline                         0.286752
malic_acid                      0.245188
alcalinity_of_ash               0.239320
alcohol                         0.144329
dtype: float64
```

Figura 15: Top variables que mas contribuyeron a la separación de variables

Solución:

- : Se escogió la base de datos **Wine**, proveniente del repositorio de *machine learning* de la Universidad de California en Irvine (UCI). Esta base de datos contiene el análisis químico de vinos cultivados en la región italiana de **Piamonte**, con el objetivo de clasificar los vinos según el tipo de uva (*cultivar*) del cual provienen.

  La base consta de vinos pertenecientes a **tres cultivares diferentes**, y cada observación incluye **13 características químicas** medidas en una muestra del vino. Estas características incluyen, entre otras:

  - Porcentaje de alcohol
  - Ácido málico

- Cenizas
- Alcalinidad de las cenizas
- Magnesio
- Fenoles totales
- Flavonoides
- Intensidad del color

El objetivo principal es utilizar estas variables químicas para construir modelos que permitan **clasificar correctamente** a qué tipo de uva corresponde cada vino. En este punto, se imprimen los valores de la dimensión I2 una vez reemplazados por la media del bin correspondiente. Los valores menores o iguales a la media general se reemplazan por la media de su grupo (bin1) y los mayores por la media del otro grupo (bin2).

- `PCA`: Se observó que los dos primeros componentes principales explican en conjunto un **56 %** de la variabilidad total de los datos, lo cual permite visualizar una separación clara entre las clases de vino. Esta reducción de dimensionalidad facilita la agrupación y análisis visual de los vinos según su cultivar.
- `Importancia de variables`: Dentro de las variables más relevantes identificadas mediante el análisis de componentes principales, las tres con mayor contribución al primer componente fueron: **flavonoids** con un valor de **0.42**, **total phenols** con **0.39** y **od280/od315** con **0.37**. Estas variables son claves para la separación entre clases de vino.