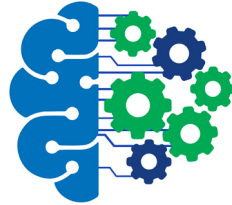


ESCUELA POLITÉCNICA SUPERIOR

UNIVERSIDAD DE CÓRDOBA



# INTRODUCCIÓN AL APRENDIZAJE AUTOMÁTICO

*Memoria de prácticas*

BASE DE DATOS: FANTASMAS

Eduardo Arroyo Ramírez

i12arrae@uco.es

26/5/2020

# Índice

<b>I</b>	<b>Práctica 1</b>	<b>5</b>
1.	Actividad 1-1	5
1.1.	Consideraciones generales . . . . .	5
1.2.	Base de datos <b>breast-cancer</b> . . . . .	6
1.3.	Base de datos <b>dermatology</b> . . . . .	6
1.4.	Base de datos <b>wine</b> . . . . .	7
2.	Actividad 1-2	8
2.1.	filters/unsupervised/instance/Resample . . . . .	8
2.2.	filters/unsupervised/attributes/NominalToBinary . . . . .	10
2.3.	filters/unsupervised/attributes/RemoveUseless . . . . .	12
3.	Actividad 1-3	14
3.1.	filters/supervised/instance/Resample . . . . .	14
3.2.	filters/supervised/instance/ClassBalancer . . . . .	17
3.3.	filters/supervised/instance/SpreadSubsample . . . . .	18
<b>II</b>	<b>Práctica 2</b>	<b>20</b>
4.	Actividad 2-1	20
4.1.	Introducción . . . . .	20
4.2.	Preparación . . . . .	21
4.3.	Datos perdidos . . . . .	22
4.4.	Unificación de medidas . . . . .	23
4.5.	Selección de características . . . . .	24
4.5.1.	Análisis de atributos irrelevantes . . . . .	24
4.5.2.	Análisis de atributos redundantes . . . . .	25
4.6.	Normalización . . . . .	27
4.7.	Datos extremos . . . . .	28
<b>III</b>	<b>Práctica 3</b>	<b>30</b>
5.	Actividad 3-1	30
6.	Actividad 3-2	32
7.	Actividad 3-3	34
<b>IV</b>	<b>Práctica 4</b>	<b>36</b>
8.	Actividad 4-1	36
9.	Actividad 4-2	39
	<b>Bibliografía</b>	<b>43</b>

## Índice de figuras

1.	Ejemplo de archivo <code>.arff</code> . . . . .	5
2.	Captura de <code>breast-cancer.arff</code> . . . . .	6
3.	Archivo <code>breast-cancer.arff</code> cargado en Weka. . . . .	6
4.	Captura de <code>dermatology.arff</code> . . . . .	7
5.	Archivo <code>dermatology.arff</code> cargado en Weka. . . . .	7
6.	Captura del archivo <code>wine.arff</code> . . . . .	7
7.	Archivo <code>wine.arff</code> cargado en Weka. . . . .	7
8.	Configuración del filtro <code>unsupervised/Resmaple</code> . . . . .	9
9.	Frecuencias de clases con diferentes <code>undersamplings</code> . . . . .	9
10.	Frecuencias de clases con diferentes <code>oversamplings</code> . . . . .	10
11.	Configuración del filtro <code>unsupervised/NominalToBinary</code> . . . . .	11
12.	Información del atributo <code>menopause</code> . . . . .	11
13.	Información del atributo <code>node_caps</code> . . . . .	11
14.	Campos tras aplicar <code>NominalToBinary</code> . . . . .	12
15.	Configuración del filtro <code>unsupervised/RemoveUseless</code> . . . . .	12
16.	Atributo <code>atr1</code> . . . . .	13
17.	Atributo <code>atr2</code> . . . . .	13
18.	Atributo <code>atr3</code> . . . . .	13
19.	Atributo <code>atr4</code> . . . . .	13
20.	Atributo <code>atr5</code> . . . . .	13
21.	Atributo <code>atr6</code> . . . . .	13
22.	Resultado de aplicar el filtro <code>RemoveUseless</code> . . . . .	13
23.	Configuración del filtro <code>supervised/Resmaple</code> . . . . .	15
24.	Frecuencias de clases al hacer resample con <code>biasToUniformClass=0</code> . . . . .	16
25.	Frecuencias de clases al hacer resamples con <code>biasToUniformClass=1</code> . . . . .	16
26.	Diálogo de configuración del filtro supervisado <code>ClassBalancer</code> . . . . .	17
27.	Datos de la clase antes de aplicar el filtro. . . . .	17
28.	Datos de la clase después de aplicar el filtro. . . . .	17
29.	Datos balanceados en dos intervalos. . . . .	18
30.	Datos balanceados en cinco intervalos. . . . .	18
31.	Configuración del filtro supervisado <code>SpreadSubsample</code> . . . . .	18
32.	Distribución inicial de la clase en la base de datos <code>breast-cancer</code> . . . . .	19
33.	Resultado del filtro con <code>distributionSpread=0</code> y <code>maxCount=50</code> . . . . .	19
34.	Resultado del filtro con <code>distributionSpread=1</code> y <code>maxCount=0</code> . . . . .	19
35.	Resultado del filtro con <code>distributionSpread=1.5</code> . . . . .	19
36.	Configuración del clasificador logístico . . . . .	21
37.	Captura de <code>alturaola.arff</code> . . . . .	21
38.	Captura de <code>alturaolatest.arff</code> . . . . .	21
39.	Resultados del clasificador logístico antes del tratamiento de datos. . . . .	22
40.	Resultados tras eliminar TIDS, VIS y MWD. . . . .	22
41.	Resultados tras reemplazar los datos perdidos . . . . .	23
42.	Resultados de <code>CorrelationAttributeEval</code> . . . . .	24
43.	Resultados tras eliminar atributos irrelevantes . . . . .	25
44.	Resultados tras quitar atributos redundantes . . . . .	26
45.	Resultados tras normalización de datos . . . . .	28

46.	Detección de valores extremos con Weka . . . . .	28
47.	Resultados tras eliminación de valores extremos . . . . .	29
48.	Influencia de los atributos en la clase . . . . .	31
49.	Matriz de correlaciones entre atributos . . . . .	31
50.	Configuración del clasificador IBk . . . . .	32
51.	Resultados del clasificador IBk con un 10-fold . . . . .	33
52.	Configuración del clasificador SimpleLogistic . . . . .	34
53.	Pesos de los atributos para cada clase . . . . .	35
54.	Resultados del clasificador SimpleLogistic con un 10-fold . . . . .	35
55.	Árbol de decisión obtenido mediante C4.5 . . . . .	37
56.	Resultados del clasificador . . . . .	38
57.	Topología de la red . . . . .	39
58.	Formulario de configuración para MultilayerPerceptron . . . . .	40
59.	Resultados de MultilayerPerceptron con configuración por defecto . . . . .	41

## Índice de cuadros

1.	Frecuencias de clases con diferentes undersamplings . . . . .	9
2.	Frecuencias de clases con diferentes oversamplings . . . . .	10
3.	Frecuencias de clases con diferentes resamples y <code>biasToUniformClass=0</code> . . . . .	15
4.	Frecuencias de clases con diferentes resamples y <code>biasToUniformClass=1</code> . . . . .	16
5.	Datos perdidos y medias de atributos en entrenamiento por clase. . . . .	23
6.	Correlaciones entre atributos . . . . .	25
7.	Valores máximos, mínimos y rangos por atributo. . . . .	27
8.	Pesos de los atributos para cada clase . . . . .	35
9.	Patrones correctamente clasificados en función del training time . . . . .	41
10.	Patrones correctamente clasificados en función del learning rate . . . . .	42

## Parte I

# Práctica 1

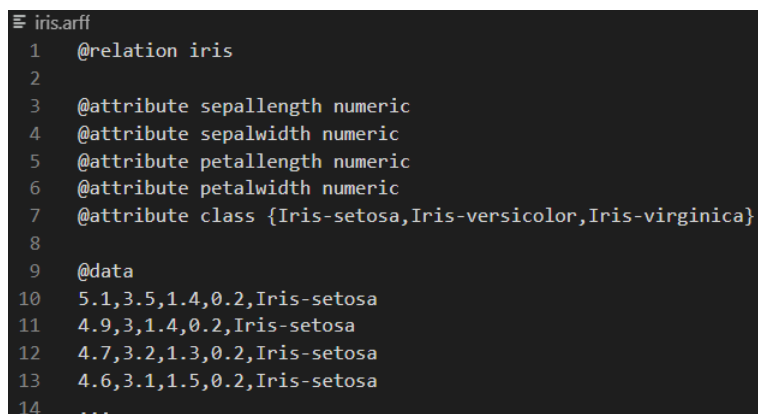
## 1. Actividad 1-1

*Elija 3 bases de datos de la UCI Machine Learning Repository de las que hay en Moodle y transformelas a .arff, indicando en cada una de ellas qué procedimiento ha seguido.*

### 1.1. Consideraciones generales

Para la realización de la tarea he tomado la decisión de editar manualmente los archivos utilizando Visual Studio Code y guardarlos con la extensión `.arff`. El paso a csv utilizando excel que se ha sugerido en clase supone varios cambios de formato en los que pueden aparecer diversos problemas como conflictos entre el separador de columnas CSV y el separador de decimales o miles, problemas con el carácter de salto de línea, codificación, etc.

Para conocer los identificadores y tipos de los atributos de cada base de datos he consultado el archivo `.names` del directorio de descarga de cada base de datos, que contiene el listado de campos con su nombre, su tipo y sus posibles valores, si procede. He consultado la especificación del formato `.arff` en la [página correspondiente del manual de weka](#)<sup>1</sup>. Como puede comprobarse en la figura 1, un archivo `.arff` consta de tres secciones:



```
iris.arff
1  @relation iris
2
3  @attribute sepalength numeric
4  @attribute sepalwidth numeric
5  @attribute petallength numeric
6  @attribute petalwidth numeric
7  @attribute class {Iris-setosa,Iris-versicolor,Iris-virginica}
8
9  @data
10 5.1,3.5,1.4,0.2,Iris-setosa
11 4.9,3,1.4,0.2,Iris-setosa
12 4.7,3.2,1.3,0.2,Iris-setosa
13 4.6,3.1,1.5,0.2,Iris-setosa
14 ...
```

Figura 1: Ejemplo de archivo `.arff`

1. Identificación de la base de datos. Se trata de una línea con el token `@relation` seguido por un espacio y el nombre de la base de datos. Por ejemplo: `@relation breast-cancer`.
2. Identificación de los atributos. Tantas líneas como atributos tenga la base de datos, cada una comienza con el token `@attribute` seguido del nombre del atributo y el tipo. Los tipos pueden ser:
  - Numéricos: `@attribute <nombre_atributo> numeric`
  - Cadenas de texto: `@attribute <nombre_atributo> string`
  - Listas de etiquetas: `@attribute <nombre_atributo> {valor_1, valor_2, ...}`
  - Fechas: `@attribute <nombre_atributo> date [formato_de_fecha]`. El formato de fecha es opcional, y por defecto acepta ISO-8601 y “yyyy-MM-dd’T’HH:mm:ss”.

---

<sup>1</sup><https://www.cs.waikato.ac.nz/ml/weka/arff.html>

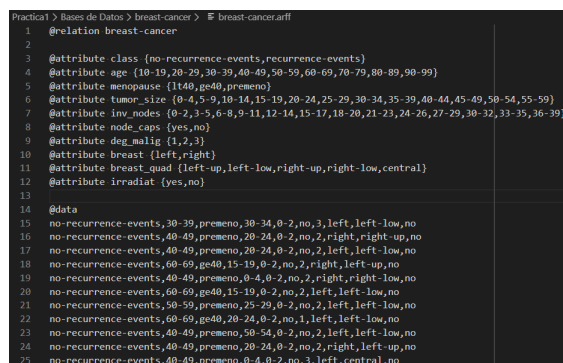
3. Bloque de datos. Esta sección se inicia con una línea que contiene únicamente palabra clave **@data**. A continuación se encontrarán los registros dispuestos en líneas y con sus atributos separados por comas, en el mismo orden en que se han especificado en la cabecera:

```
@data
v1a1, v1a2, ..., v1aN
v2a1, v2a2, ..., v2aN
...
vMa1, vMa2, ..., vMaN
```

## 1.2. Base de datos **breast-cancer**

Tras aplicar el tratamiento mencionado en el apartado 1.1 al archivo **breast-cancer.data** se ha obtenido el fichero **.arff** (fig. 2) y una vez cargado en Weka (fig. 3) y se han obtenido las siguientes conclusiones:

1. La base de datos tiene 10 atributos, de los cuales el primero es la clase, que toma los valores “no-recurrence-events” y “recurrence-events”. Convendría colocar la clase al final, que es su lugar por defecto.
2. Los atributos son nominales basados en etiquetas (por ejemplo **breast\_quad**, **menopause...**) o en rangos numéricos (**inv\_nodes**, **tumor\_size...**). El atributo **deg\_malign** se podría poner como numérico ya que parece representar el grado de malignidad en un rango de 1 a 3, por lo que hay una distancia distinta entre los elementos (por ejemplo 1-2 y 1-3).



```
@relation breast-cancer
@attribute class {no-recurrence-events,recurrence-events}
@attribute age {10-19,20-29,30-39,40-49,50-59,60-69,70-79,80-89,90-99}
@attribute menopause {1,40,ge40,premeno}
@attribute tumor_size {0-4,5-9,10-14,15-19,20-24,25-29,30-34,35-39,40-44,45-49,50-54,55-59}
@attribute inv_nodes {0-2,3-5,6-8,9-11,12-14,15-17,18-20,21-23,24-26,27-29,30-32,33-35,36-39}
@attribute node_caps {yes,no}
@attribute deg_malign {1,2,3}
@attribute breast {left,right}
@attribute breast_quad {left-up,left-low,right-up,right-low,central}
@attribute irradiat {yes,no}

@data
no-recurrence-events,30-39,premeno,30-34,0-2,no,3,left,left-low,no
no-recurrence-events,40-49,premeno,20-24,0-2,no,2,right,right-up,no
no-recurrence-events,40-49,premeno,20-24,0-2,no,2,left,left-low,no
no-recurrence-events,60-69,ge40,15-19,0-2,no,2,right,left-up,no
no-recurrence-events,40-49,premeno,0-4,0-2,no,2,right,right-low,no
no-recurrence-events,60-69,ge40,15-19,0-2,no,2,left,left-low,no
no-recurrence-events,50-59,premeno,25-29,0-2,no,2,left,left-low,no
no-recurrence-events,60-69,ge40,20-24,0-2,no,1,left,left-low,no
no-recurrence-events,40-49,premeno,50-54,0-2,no,2,left,left-low,no
no-recurrence-events,40-49,premeno,20-24,0-2,no,2,right,left-up,no
no-recurrence-events,40-49,premeno,0-4,0-2,no,3,left,central,no
```

Figura 2: Captura de **breast-cancer.arff**.

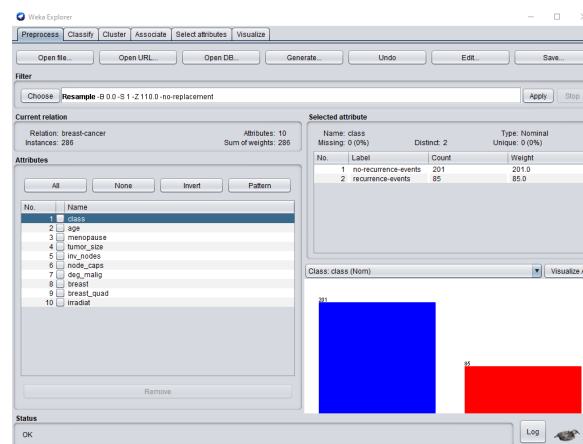


Figura 3: Archivo **breast-cancer.arff** cargado en Weka.

## 1.3. Base de datos **dermatology**

Tras aplicar el tratamiento mencionado en el apartado 1.1 al archivo **dermatology.data** se ha obtenido el fichero **.arff** (fig. 4) y una vez cargado en Weka (fig. 5) y se han obtenido las siguientes conclusiones:

1. La base de datos tiene 34 atributos independientes y una clase. En total hay 366 patrones.
2. La mayoría los atributos son de tipo numérico con valores [0-3]. En la descripción se indica que los valores indican un grado obtenido de un análisis. El caso del atributo **family-history** es una excepción, ya que es nominal con valores 0 y 1 y representa si alguna de las enfermedades ha sido observada en la familia. Otra excepción es la edad, que siendo numérica, no está restringida al rango anterior.

3. La clase toma valores de 1 a 6 y cada valor representa un diagnóstico diferente, por lo que es un dato nominal.

```
@relation dermatology

@attribute erythema numeric
@attribute scaling numeric
@attribute definite-borders numeric
@attribute itching numeric
@attribute koebner-phenomenon numeric
@attribute polygonal-papules numeric
@attribute follicular-papules numeric
@attribute oral-mucosal-involvement numeric
@attribute knee-and-elbow-involvement numeric
@attribute scalp-involvement numeric
@attribute family-history {0,1}
@attribute Age numeric
@attribute melanin-incontinence numeric
@attribute eosinophils-in-the-infiltrate numeric
@attribute PMU-infiltrate numeric
@attribute fibrosis-of-the-papillary-dermis numeric
@attribute exocytosis numeric
@attribute acanthosis numeric
@attribute hyperkeratosis numeric
@attribute parakeratosis numeric
@attribute clubbing-of-the-rete-ridges numeric
@attribute elongation-of-the-rete-ridges numeric
@attribute thinning-of-the-suprapapillary-epidermis numeric
@attribute spongiform-pustule numeric
@attribute munro-microabscess numeric
@attribute focal-hypergranulosis numeric
@attribute disappearance-of-the-granular-layer numeric
@attribute vacuolisation-and-damage-of-basal-layer numeric
@attribute spongiosis numeric
@attribute saw-tooth-appearance-of-retes numeric
@attribute follicular-horn-plug numeric
@attribute perifollicular-parakeratosis numeric
@attribute inflammatory-mononuclear-infiltrate numeric
@attribute band-like-infiltrate numeric
@attribute class {1,2,3,4,5,6}

@data
2,2,0,3,0,0,0,0,1,0,0,0,0,0,0,3,2,0,0,0,0,0,0,0,0,0,0,3,0,0,0,
1,0,55,2
```

Figura 4: Captura de dermatology.arff.

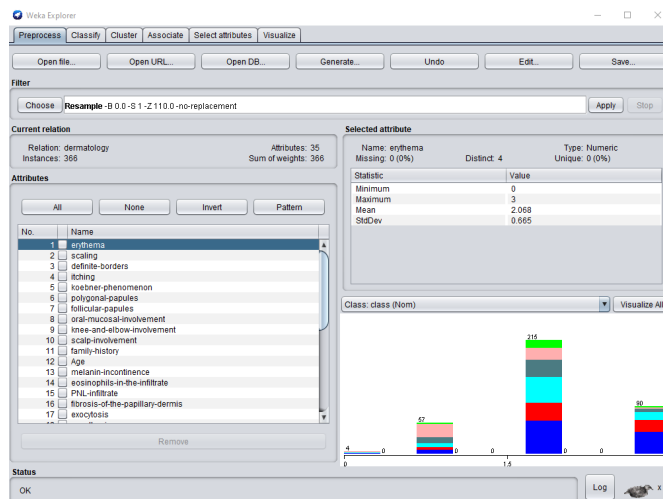


Figura 5: Archivo dermatology.arff cargado en Weka.

## 1.4. Base de datos wine

Tras aplicar el tratamiento mencionado en el apartado 1.1 al archivo wine.data se ha obtenido el fichero .arff (fig. 6) y una vez cargado en Weka (fig. 7) y se han obtenido las siguientes conclusiones:

1. La base de datos tiene 13 atributos independientes y una clase al principio.
2. Los atributos son numéricos continuos.
3. La clase toma valores 1, 2 o 3 con frecuencias 59, 71 y 48 respectivamente. Es un dato nominal

```
@relation wine

@attribute class {1,2,3}
@attribute alcohol numeric
@attribute malic-acid numeric
@attribute ash numeric
@attribute alkalinity-of-ash numeric
@attribute magnesium numeric
@attribute total-phenols numeric
@attribute flavanoids numeric
@attribute nonflavanoid-phenols numeric
@attribute proanthocyanins numeric
@attribute color-intensity numeric
@attribute hue numeric
@attribute od280-od315-of-diluted-wines numeric
@attribute proline numeric

@data
1,14,23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13,2,1.78,2.14,11,2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
```

Figura 6: Captura del archivo wine.arff.

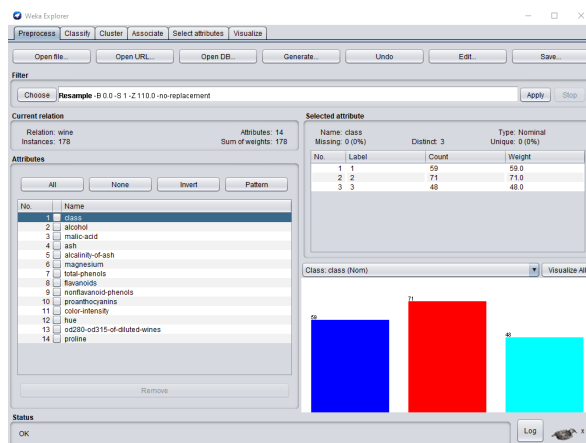


Figura 7: Archivo wine.arff cargado en Weka.



## 2. Actividad 1-2

*Elija 3 filtros No Supervisados de los que aparecen listados, explíquelos y describa cómo quedan los datos antes y después al aplicarlos sobre una o varias bases de datos.*

- Consulte el *UCI Machine Learning Repository* para una descripción de la base de datos y la transformación a `.arff`
- Si no puede aplicar un filtro elegido en ninguna base de datos describa por qué, y constrúyase una base de datos ficticia y pequeña donde si pueda aplicarlo.
- Use capturas de pantalla, salidas de Weka y todo lo que considere necesario para sus ejercicios.
- La puntuación variará en función de la argumentación y dificultad de los filtros elegidos.
  1. `filters/unsupervised/attribute/Normalize`
  2. `filters/unsupervised/attribute/ReplaceMissingValues`
  3. `filters/unsupervised/attributes/NominalToBinary`
  4. `filters/unsupervised/instance/RemoveDuplicates`
  5. `filters/unsupervised/instance/Resample`
  6. `filters/unsupervised/attribute/Remove`
  7. `filters/unsupervised/attributes/RemoveUseless`

### 2.1. `filters/unsupervised/instance/Resample`

Según la información que proporciona Weka, este filtro produce un conjunto de datos mediante un remuestreo de la base de datos original. Este remuestreo no supervisado se utiliza para aumentar el número de patrones (**oversampling**) mediante la creación de duplicados de patrones existentes o para reducirlo (**undersampling**) mediante la eliminación de patrones aleatorios. La selección de patrones para oversampling se puede hacer con o sin repetición. Si es sin repetición, un patrón duplicado no podrá ser seleccionado de nuevo para su duplicación.

Para ilustrar el comportamiento del filtro observaremos cómo cambian las frecuencias relativas en las clases tras realizar diversos undersamplings y oversamplings. La hipótesis es que las frecuencias relativas no se mantendrán constantes ya que el filtro elimina o duplica patrones al azar. En Weka, se ha cargado la base de datos `dermatology.arff`. En la base de datos hay inicialmente 366 patrones repartidos en 6 clases con las frecuencias que se pueden ver en la columna “Frec. Orig.” del cuadro 1. Posteriormente, se ha seleccionado el filtro Resample no supervisado: `unsupervised/instance/Resample`.

El filtro Resample no supervisado cuenta con los siguientes parámetros de configuración (ver fig. 8):

- `invertSelection False/True`: Invierte la selección (descartados por seleccionados).
- `noReplacement True/False`: Deshabilita el reemplazo de instancias, esto es, al seleccionar instancias para duplicar, permite o no que la misma instancia sea seleccionada más de una vez. Esto sólo tiene sentido al hacer oversampling (`sampleSizePercent>100`).
- `randomSeed (número 1)`: Semilla para la selección aleatoria.

- `sampleSizePercent` (número 100): Tamaño del conjunto resultante como % del original.

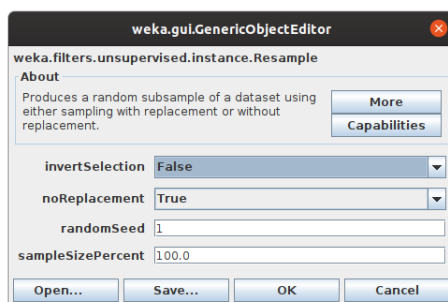


Figura 8: Configuración del filtro `unsupervised/Resmaple`.

Para este ejercicio se han establecido los siguientes valores de configuración:

- `randomSeed=50`
- `noReplacement=True`
- `invertSelection=False`
- `sampleSizePercent={75, 50, 25}`.

En el cuadro 1 y figura 9 se muestra la distribución de las clases al utilizar el filtro para realizar **undersampling** reduciendo el conjunto al 75 %, al 50 % y al 25 %. Cada resample se ha realizado sobre el conjunto original. El proceso ha sido: aplicar el filtro, tomar los datos, deshacer, siguiente.

Clase	Frec. Orig.		Frec. US75		Frec. US50		Frec. US25	
1	112	30,60 %	87	31,75 %	62	33,88 %	31	34,07 %
2	61	16,67 %	43	15,69 %	25	13,66 %	13	14,29 %
3	72	19,67 %	54	19,71 %	37	20,22 %	16	17,58 %
4	49	13,39 %	37	13,50 %	24	13,11 %	12	13,19 %
5	52	14,21 %	39	14,23 %	27	14,75 %	16	17,58 %
6	20	5,46 %	14	5,11 %	8	4,37 %	3	3,30 %
Total <sup>2</sup>	366	100 %	274	74,86 %	183	50,00 %	91	24,86 %

Cuadro 1: Frecuencias de clases con diferentes undersamplings

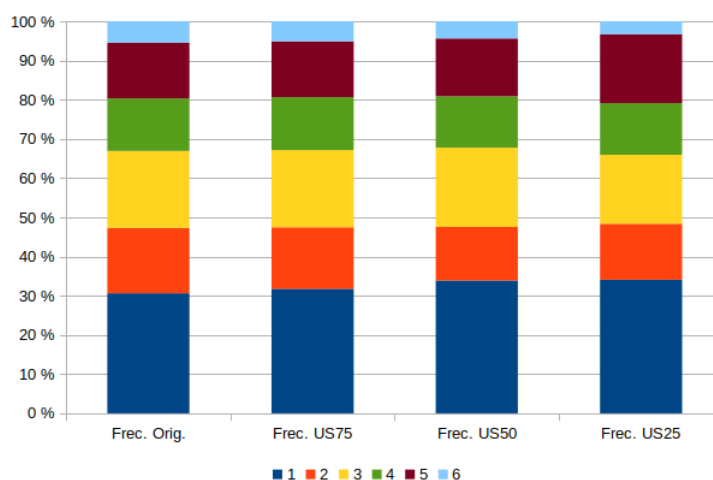


Figura 9: Frecuencias de clases con diferentes undersamplings

<sup>2</sup>Porcentajes respecto al tamaño número inicial de patrones

Siguiendo la misma dinámica que con el undersampling, se ha realizado un oversampling del conjunto original utilizando el filtro Resample no supervisado, observado que sólo tiene efecto cuando el parámetro `noReplacement` se pone a valor `False`. En el cuadro 2 y figura 10 se pueden ver las frecuencias de las diferentes clases al realizar oversamplings al 125 %, 150 %, 175 % y 200 %.

Clase	Frec. Orig.		Frec. OS125		Frec. OS150		Frec. OS175		Frec. OS200	
1	112	30,60 %	136	29,76 %	168	30,60 %	190	29,69 %	216	29,51 %
2	61	16,67 %	66	14,44 %	83	15,12 %	100	15,63 %	114	15,57 %
3	72	19,67 %	89	19,47 %	107	19,49 %	127	19,84 %	149	20,36 %
4	49	13,39 %	58	12,69 %	68	12,39 %	80	12,50 %	88	12,02 %
5	52	14,21 %	79	17,29 %	91	16,58 %	104	16,25 %	118	16,12 %
6	20	5,46 %	29	6,35 %	32	5,83 %	39	6,09 %	47	6,42 %
Total <sup>3</sup>	366	100,00 %	457	124,86 %	549	150,00 %	640	174,86 %	732	200,00 %

Cuadro 2: Frecuencias de clases con diferentes oversamplings

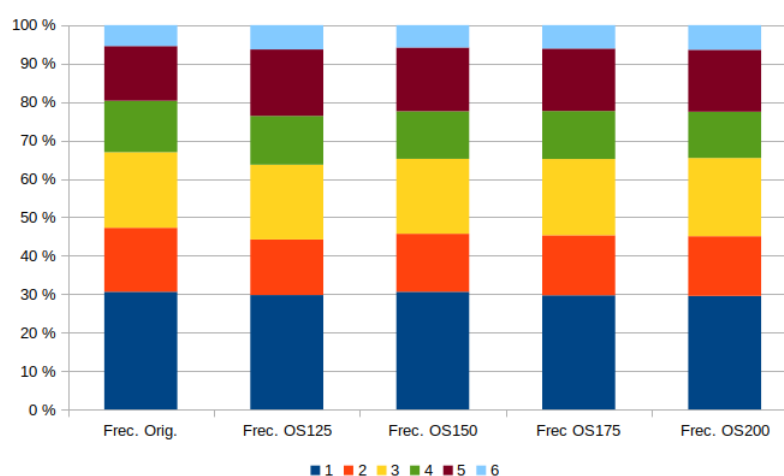


Figura 10: Frecuencias de clases con diferentes oversamplings

Al comparar la frecuencia de cada clase tras las distintas aplicaciones del filtro (fig. 9 y 10), se puede observar que varían para los distintos valores de `sampleSizePercent`, aunque no demasiado. Dado que se trata de un filtro no supervisado (no tiene en cuenta la clase a la hora de eliminar o crear patrones), este hecho puede deberse a que la base de datos es bastante homogénea en el sentido de que los patrones de cada clase se encuentran distribuidos regularmente a lo largo de la base de datos. En casos de undersampling extremo (ej.: reducción al 4 %) se observa cómo llegan a desaparecer todos los patrones de algunas clases.

En el apartado 3.1 se probará el filtro supervisado equivalente para comparar las frecuencias obtenidas. Sería lógico suponer que en el filtro supervisado, las frecuencias relativas de las clases variarían menos que en el supervisado según se va reduciendo la muestra.

## 2.2. filters/unsupervised/attributes/NominalToBinary

El filtro no supervisado `NominalToBinary` reemplaza cada atributo nominal que tenga más de dos valores distintos, por tantos atributos binarios como valores diferentes tenga el atributo original. Estos atributos binarios tendrán valor 1 si el patrón tenía el valor correspondiente al atributo binario en el atributo original y 0 si no. Este proceso se denomina binarización.

El filtro `NominalToBinary`, cuyo formulario de configuración puede verse en la figura 11 cuenta con los siguientes parámetros:

<sup>3</sup>Porcentajes respecto al tamaño muestral inicial

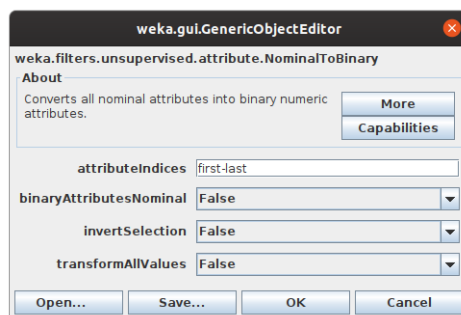


Figura 11: Configuración del filtro `unsupervised/NominalToBinary`.

- **attributeIndices** ([first-last]): indica el rango de atributos que se van a binarizar. Se especifican los índices o rangos de índices separados por comas. También son válidos los valores “first” y “last”.
- **binaryAttributesNominal** False/True: Permite establecer si los valores de los nuevos atributos binarios serán valores de tipo 0-1 o si o nominal f-t.
- **invertSelection** False/True: Si es False, se binarizan los atributos indicados en **attributeIndices**. Si es True, se binarizan sólo los que no están especificados en dicho parámetro.
- **transformAllValues** False/True: Si es True, se procesan también los atributos nominales con sólo dos valores.

Para ilustrar el funcionamiento de este filtro se ha cargado la base de datos `breast-cancer.arff` en Weka y se ha aplicado el filtro a los campos 3 y 6. El primero es nominal con tres valores y el segundo es nominal con dos valores (fig. 12 y 13). La configuración del filtro es **attributeIndices=3,6**, **binaryAttributesNominal=True**, **invertSelection=False** y **transformAllValues=False**, por lo tanto se espera que reemplace el atributo `menopause` por tres atributos, uno por cada uno de los tres valores que toma, y que el atributo `node_caps` se quede intacto ya que salvo que se ponga **transformAllValues=True** o **binaryAttributesNominal=False**, este filtro no altera los atributos nominales de dos valores.

Selected attribute			
Name: menopause		Type: Nominal	
Missing: 0 (0%)		Distinct: 3	
		Unique: 0 (0%)	
No.	Label	Count	
1	lt40	7	
2	ge40	129	
3	premeno	150	

Figura 12: Información del atributo `menopause`

Selected attribute		
Name: node_caps		
Type: Nominal		
Missing: 8 (3%)		
Distinct: 2		
Unique: 0 (0%)		
No.	Label	Count
1	yes	56
2	no	222

Figura 13: Información del atributo `node_caps`

Los resultados obtenidos (fig. 14) confirman la hipótesis:

- El filtro ha reemplazado el atributo `menopause` por tres campos binarios, uno por cada valor diferente que tomaba el atributo original: `menopause=lt40`, `menopause=ge40` y `menopause=premeno`.
- El filtro no ha alterado el atributo `node_caps` porque sólo toma dos valores diferentes.
- Las distribuciones de los nuevos campos son coherentes con la distribución del campo original, es decir: hay 7 patrones con valor 1 en `menopause=lt40`, hay 129 patrones con valor 1 en `menopause=ge40` y hay 150 patrones con valor 1 en `menopause=premeno`.

No.	Name
1	class
2	age
3	menopause=lt40
4	menopause=ge40
5	menopause=premeno
6	tumor_size
7	inv_nodes
8	node_caps
9	deg_malig
10	breast
11	breast_quad
12	irradiat

Figura 14: Campos tras aplicar NominalToBinary.

Se ha observado que si se establece a False el parámetro , altera los atributos nominales con dos valores, reemplazando los valores nominales por 0 y 1 pero sin reemplazar el atributo. Para que reemplace el atributo como hace con los nominales de más de dos valores, debe ponerse a True el parámetro `transformAllValues`.

### 2.3. filters/unsupervised/attributes/RemoveUseless

El filtro no supervisado RemoveUseless elimina los atributos de la base de datos que no varían o que lo hacen demasiado. Es decir, elimina todos los atributos con valor constante o cuya variación excede el porcentaje indicado en el parámetro `maximumVariancePercentageAllowed`. Para calcular la variación de un atributo se aplica la siguiente fórmula:

$$\text{Variación} = \frac{\text{Nº Valores Distintos}}{\text{Nº Valores Total}} \cdot 100$$

El formulario de configuración de este filtro puede verse en la figura 15.

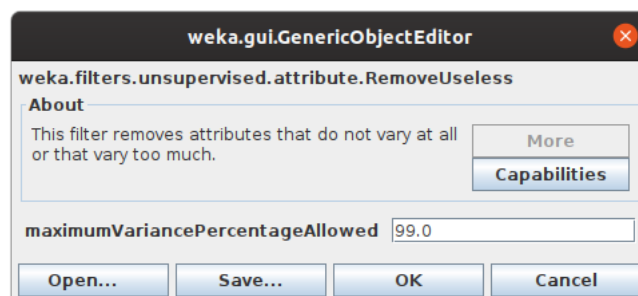


Figura 15: Configuración del filtro unsupervised/RemoveUseless

Para probar este filtro se va a componer una base de datos de ejemplo con 200 patrones y 6 atributos más una clase. Los atributos tienen las siguientes características:

1. **atr1** nominal binario con valores SI/NO. Todos los patrones tienen valor NO. Los datos que arroja Weka sobre este atributo se ven en la fig. 16.
2. **atr2** nominal binario con valores SI/NO. Todos los patrones tienen valor NO excepto uno con valor SI. Este atributo se ha puesto como control del primero, para comprobar que no es eliminado por el filtro. Los datos que arroja Weka sobre este atributo se ven en la fig. 17.
3. **atr3** numérico. Todos los patrones tienen valor 7.0. Es el mismo caso que **atr1** para numérico. Se espera que el filtro elimine este campo también. Los datos que arroja Weka sobre este atributo se ven en la fig. 18.

4. **atr4** nominal con valores **a1...a200**. Cada atributo toma un valor diferente. La tasa de variación es  $\frac{200}{200} \cdot 100 = 100$ . Los datos que arroja Weka sobre este atributo se ven en la fig. 18.
5. **atr5** nominal con valores **b1...b199**. Cada atributo toma un valor diferente salvo los dos últimos que son iguales. La tasa de variación es  $\frac{199}{200} \cdot 100 = 99,5$ . Los datos que arroja Weka sobre este atributo se ven en la fig. 20.
6. **atr6** nominal con valores **a1...a200**. Cada atributo toma un valor diferente salvo los tres últimos que son iguales. La tasa de variación es  $\frac{198}{200} \cdot 100 = 99$ . Los datos que arroja Weka sobre este atributo se ven en la fig. 21.

Name: atr1	Distinct: 1	Type: Nominal
Missing: 0 (0%)		Unique: 0 (0%)

Figura 16: Atributo **atr1**

Name: atr2	Distinct: 2	Type: Nominal
Missing: 0 (0%)		Unique: 1 (1%)

Figura 17: Atributo **atr2**

Name: atr3	Distinct: 1	Type: Numeric
Missing: 0 (0%)		Unique: 0 (0%)

Figura 18: Atributo **atr3**

Name: atr4	Distinct: 200	Type: Nominal
Missing: 0 (0%)		Unique: 200 (100%)

Figura 19: Atributo **atr4**

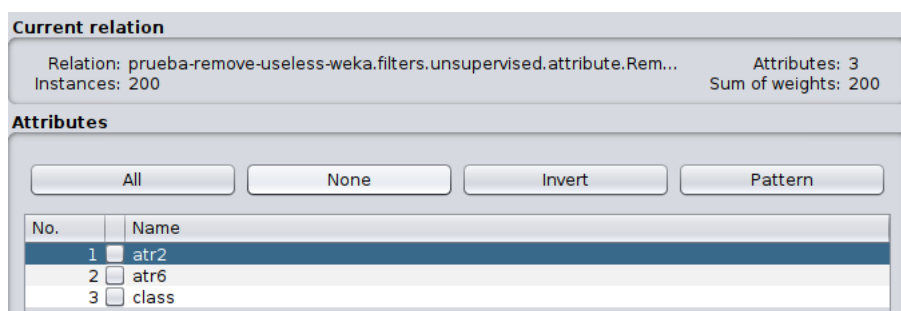
Name: atr5	Distinct: 199	Type: Nominal
Missing: 0 (0%)		Unique: 198 (99%)

Figura 20: Atributo **atr5**

Name: atr6	Distinct: 198	Type: Nominal
Missing: 0 (0%)		Unique: 197 (99%)

Figura 21: Atributo **atr6**

Se aplicará la configuración por defecto del filtro, que especifica el valor 99 para el parámetro `maximumVaiancePorcentajeAllowed`. Se espera que el filtro elimine los atributos 1 y 3 por ser constantes y 4 y 5 por tener una tasa de variación superior al 99% configurado en el parámetro. Como se puede comprobar en la figura 22, el filtro se ha comportado como se esperaba: eliminando los atributos **atr1**, **atr3**, **atr4** y **atr5** y manteniendo **atr2** y **atr6**.

Figura 22: Resultado de aplicar el filtro **RemoveUseless**.

### 3. Actividad 1-3

*Elija 3 filtros Supervisados de los que aparecen listados, explíquelos y describa cómo quedan los datos antes y después al aplicarlos sobre una o varias bases de datos.*

- Consulte el *UCI Machine Learning Repository* para una descripción de la base de datos y la transformación a `.arff`
- Si no puede aplicar un filtro elegido en ninguna base de datos describa por qué, y constrúyase una base de datos ficticia y pequeña donde si pueda aplicarlo.
- Use capturas de pantalla, salidas de Weka y todo lo que considere necesario para sus ejercicios.
- La puntuación variará en función de la argumentación y dificultad de los filtros elegidos.

1. `filters/supervised/attribute/Discretize`
2. `filters/supervised/attribute/NominalToBinary`
3. `filters/supervised/instance/SpreadSubsample`
4. `filters/supervised/instance/ClassBalancer`
5. `filters/supervised/instance/Resample`

#### 3.1. `filters/supervised/instance/Resample`

El filtro `Resample` supervisado produce un conjunto de datos mediante un remuestreo de la base de datos original teniendo en cuenta la clase. Este remuestreo supervisado se utiliza para aumentar el número de patrones (**oversampling**) mediante la creación de duplicados de patrones existentes o para reducirlo (**undersampling**) mediante la eliminación de patrones manteniendo ciertas proporciones entre los patrones de cada clase. La selección de patrones para **oversampling** se puede hacer con o sin repetición. Si es sin repetición, un patrón duplicado no podrá ser seleccionado de nuevo para su duplicación. Según la documentación, la clase debe ser de tipo nominal para poder utilizar este filtro. De lo contrario, deberemos utilizar la versión no supervisada 2.1.

Para comprobar el comportamiento del filtro observaremos cómo cambian las frecuencias relativas en las clases tras realizar diversos **undersamplings** y **oversamplings** en función del valor del parámetro `biasToUniformClass`. La hipótesis es que para el valor 0 la distribución de la clase se mantendrá igual mientras que para valor 1 se igualarán. Para valores intermedios del parámetro, mientras más cerca de 1, producirán un resultado más uniforme. En Weka, se ha cargado la base de datos `dermatology.arff`. En la base de datos hay inicialmente 366 patrones repartidos en 6 clases con las frecuencias que se pueden ver en la columna “Frec. Orig.” del cuadro 2. Posteriormente, se ha seleccionado el filtro `Resample` supervisado: `supervised/instance/Resample` cuyo formulario de configuración se puede ver en la figura 23.

El filtro `Resample` supervisado cuenta con los siguientes parámetros:

- `biasToUniformClass` (núm [0.0-1.0]): Establece el tipo de sesgo del remuestreo: para 0 se mantiene distribución original de la clase, para 1, la distribución es uniforme.
- `invertSelection` `False/True`: Invierte la selección (descartados por seleccionados).

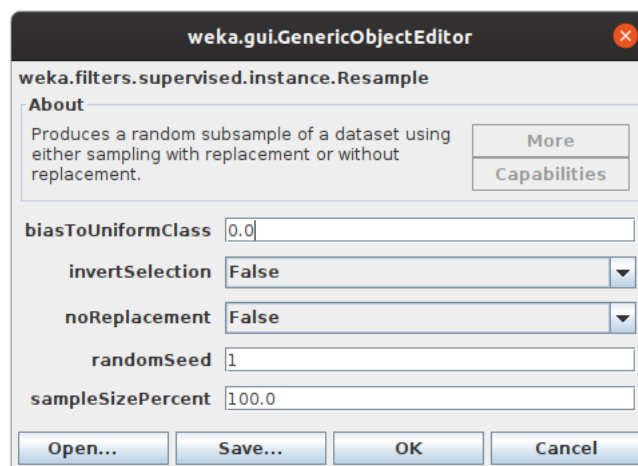


Figura 23: Configuración del filtro supervised/Resmaple.

- **noReplacement** True/False: Deshabilita el reemplazo de instancias, esto es, al seleccionar instancias para duplicar, permite o no que la misma instancia sea seleccionada más de una vez. Esto sólo tiene sentido al hacer oversampling (**sampleSizePercent**>100).
- **randomSeed** (núm 1): Semilla para la selección aleatoria.
- **sampleSizePercent** (núm 100): Tamaño del conjunto resultante como % del original.

Para este ejercicio se han establecido los siguientes valores de configuración:

- **biasToUniformClass**={0.0, 1.0}
- **randomSeed**=50
- **noReplacement**=True
- **invertSelection**=False
- **sampleSizePercent**={33, 66, 150, 200}.

En el cuadro 3 aparecen los datos de los distintos resamples realizados y se comparan las frecuencias de las clases con las originales. Estos datos se han obtenido con **biasToUniformClass**=0 y como se puede observar en el gráfico 24, las proporciones son bastante similares a las originales.

Clase	Frec. US33		Frec. US66		Frec. Orig.		Frec. OS150		Frec. OS200	
1	30	25,00 %	59	24,48 %	112	30,60 %	142	25,87 %	196	26,78 %
2	20	16,67 %	34	14,11 %	61	16,67 %	92	16,76 %	122	16,67 %
3	29	24,17 %	63	26,14 %	72	19,67 %	120	21,86 %	153	20,90 %
4	15	12,50 %	31	12,86 %	49	13,39 %	78	14,21 %	97	13,25 %
5	23	19,17 %	40	16,60 %	52	14,21 %	87	15,85 %	122	16,67 %
6	3	2,50 %	14	5,81 %	20	5,46 %	30	5,46 %	42	5,74 %
Total <sup>4</sup>	120	32,79 %	241	65,85 %	366	100,00 %	549	150,00 %	732	200,00 %

Cuadro 3: Frecuencias de clases con diferentes resamples y **biasToUniformClass**=0

<sup>4</sup>Porcentajes respecto al tamaño muestral inicial



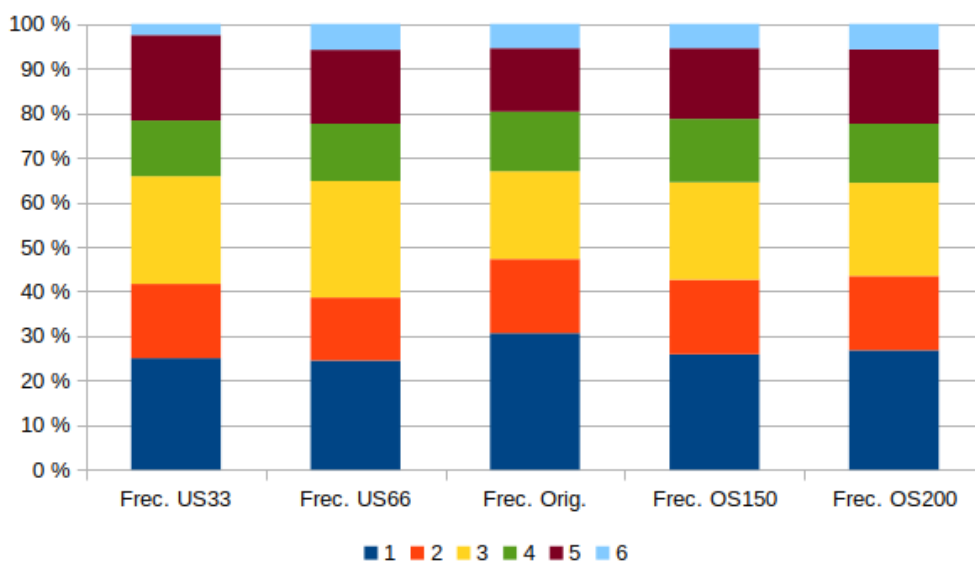


Figura 24: Frecuencias de clases al hacer resample con `biasToUniformClass=0`

En la tabla 4 aparecen los datos de los distintos resamples realizados y se comparan las frecuencias de las clases con las originales. Estos datos se han obtenido con `biasToUniformClass=1`. En este caso, los porcentajes de patrones de cada clase tienden a igualarse, tal como se puede ver en el gráfico 25.

Clase	Frec. US33		Frec. US66		Frec. Orig.		Frec. OS150		Frec. OS200	
1	16	13,33 %	43	17,84 %	112	30,60 %	93	16,94 %	122	16,67 %
2	16	13,33 %	30	12,45 %	61	16,67 %	88	16,03 %	120	16,39 %
3	26	21,67 %	40	16,60 %	72	19,67 %	93	16,94 %	114	15,57 %
4	17	14,17 %	36	14,94 %	49	13,39 %	93	16,94 %	132	18,03 %
5	24	20,00 %	48	19,92 %	52	14,21 %	97	17,67 %	123	16,80 %
6	21	17,50 %	44	18,26 %	20	5,46 %	85	15,48 %	121	16,53 %
Total <sup>5</sup>	120	32,79 %	241	65,85 %	366	100,00 %	549	150,00 %	732	200,00 %

Cuadro 4: Frecuencias de clases con diferentes resamples y `biasToUniformClass=1`

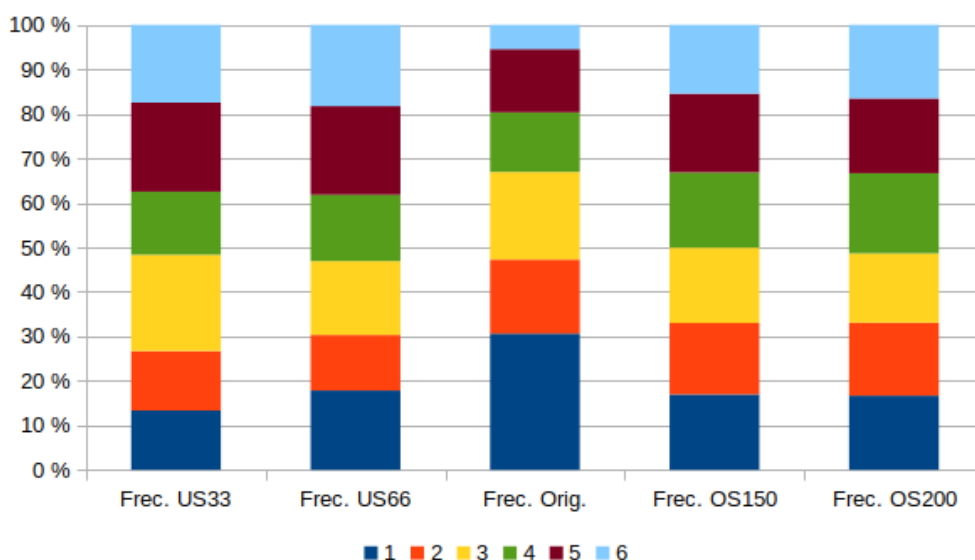


Figura 25: Frecuencias de clases al hacer resamples con `biasToUniformClass=1`

<sup>5</sup>Porcentajes respecto al tamaño muestral inicial

### 3.2. filters/supervised/instance/ClassBalancer

El filtro supervisado **ClassBalancer** asigna pesos ponderados a los patrones de manera que se equilibren las clases. Los patrones de una clase menos frecuente tendrán pesos ponderados mayores y viceversa, de manera que cada clase tenga en suma el mismo peso. En caso de que la clase fuese numérica, el filtro además la discretiza.

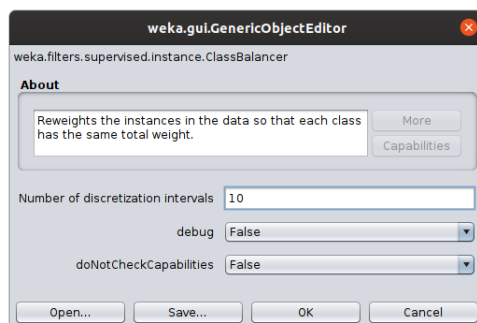


Figura 26: Diálogo de configuración del filtro supervisado **ClassBalancer**

Los parámetros del filtro son los que aparecen en la figura 26 y se describen a continuación:

- **Number of discretization intervals (10)**: N<sup>o</sup> de intervalos que se establecen al discretizar la clase, en caso de que sea de tipo numérico.
- **debug False/True**: si se pone a true saca más información en el log al ejecutar el filtro.
- **doNotCheckCapabilities (False/True)**: habilita o deshabilita la comprobación de los datos. Afecta al rendimiento del filtro.

Para probar este filtro se compondrá una base de datos de ejemplo, desbalanceada y con clase continua, de forma que podamos observar a la vez todas las capacidades del filtro. Cuando se aplica el filtro, la primera diferencia que se observa es que el gráfico de distribución de valores de la clase cambia (figs. 27 y 28).

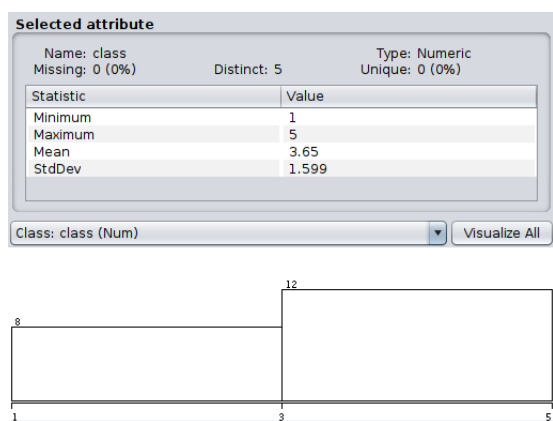


Figura 27: Datos de la clase antes de aplicar el filtro.

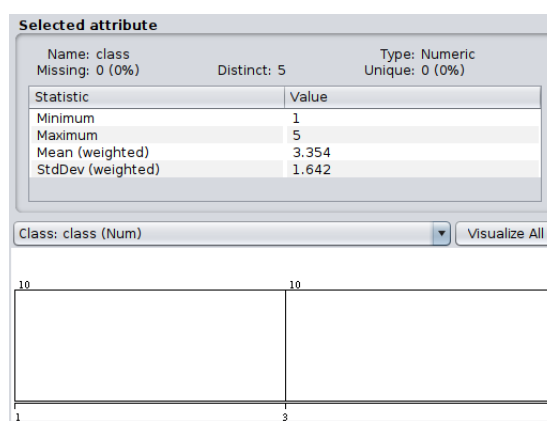


Figura 28: Datos de la clase después de aplicar el filtro.

Al aplicar el filtro, se ha añadido un nuevo campo llamado **Weight 1** con un valor numérico que representa el peso de la instancia para equilibrar la clase a la que pertenece. En la figura 29 podemos ver cómo los pesos asignados balancean la clase en dos intervalos:  $[1, 3]$  y  $(3, 5]$ . El primer intervalo tiene un peso de  $0,8\bar{3}$  y tiene 12 patrones  $0,8\bar{3} \cdot 12 = 10$ . El segundo intervalo tiene un peso de  $1,25$  y contiene 8

patrones:  $1,25 \cdot 8 = 10$ . En la figura 30 podemos ver cómo los pesos asignados balancean la clase en cinco grupos, tantos como valores diferentes toma la clase. A los 3 patrones clase 1 se les asigna peso  $1.\bar{3}$ , por lo que  $1.\bar{3} \cdot 3 = 4$ . A los 3 patrones con clase 2, les corresponde, obviamente, el mismo peso que a los de clase 1. A los 2 de clase 3, al igual que a los dos de clase 4, se les asigna peso 2, resultando  $2 \cdot 2 = 4$ . Por último, a los 10 patrones de clase 5 se les asigna peso 0,4, lo que resulta en  $0,4 \cdot 10 = 4$ . Como se puede observar, todos los intervalos tienen el mismo peso.

No.	Weight	1: atr1	2: atr2	3: atr3	4: class
		Numeric	Numeric	Numeric	Numeric
1	0.8...	8.0	67.0	63.0	4.0
2	1.25	50.0	43.0	53.0	3.0
3	0.8...	34.0	71.0	68.0	4.0
4	1.25	6.0	43.0	65.0	1.0
5	1.25	88.0	2.0	24.0	2.0
6	1.25	83.0	52.0	52.0	1.0
7	1.25	10.0	86.0	84.0	2.0
8	1.25	6.0	89.0	75.0	2.0
9	1.25	75.0	72.0	9.0	3.0
10	1.25	17.0	53.0	25.0	1.0
11	0.8...	42.0	70.0	10.0	5.0
12	0.8...	10.0	93.0	23.0	5.0
13	0.8...	2.0	72.0	98.0	5.0
14	0.8...	18.0	8.0	7.0	5.0
15	0.8...	19.0	33.0	7.0	5.0
16	0.8...	42.0	70.0	10.0	5.0
17	0.8...	10.0	93.0	23.0	5.0
18	0.8...	2.0	72.0	98.0	5.0
19	0.8...	18.0	8.0	7.0	5.0
20	0.8...	19.0	33.0	7.0	5.0

No.	Weight	1: atr1	2: atr2	3: atr3	4: class
		Numeric	Numeric	Numeric	Numeric
1	2.0	8.0	67.0	63.0	4.0
2	2.0	50.0	43.0	53.0	3.0
3	2.0	34.0	71.0	68.0	4.0
4	1.3...	6.0	43.0	65.0	1.0
5	1.3...	88.0	2.0	24.0	2.0
6	1.3...	83.0	52.0	52.0	1.0
7	1.3...	10.0	86.0	84.0	2.0
8	1.3...	6.0	89.0	75.0	2.0
9	2.0	75.0	72.0	9.0	3.0
10	1.3...	17.0	53.0	25.0	1.0
11	0.4	42.0	70.0	10.0	5.0
12	0.4	10.0	93.0	23.0	5.0
13	0.4	2.0	72.0	98.0	5.0
14	0.4	18.0	8.0	7.0	5.0
15	0.4	19.0	33.0	7.0	5.0
16	0.4	42.0	70.0	10.0	5.0
17	0.4	10.0	93.0	23.0	5.0
18	0.4	2.0	72.0	98.0	5.0
19	0.4	18.0	8.0	7.0	5.0
20	0.4	19.0	33.0	7.0	5.0

Figura 29: Datos balanceados en dos intervalos.      Figura 30: Datos balanceados en cinco intervalos.

### 3.3. filters/supervised/instance/SpreadSubsample

El filtro supervisado **SpreadSubsample** realiza remuestreos balanceados de la base de datos. Es decir, produce un nuevo conjunto a partir del anterior con un determinado balance de frecuencias de las clases.

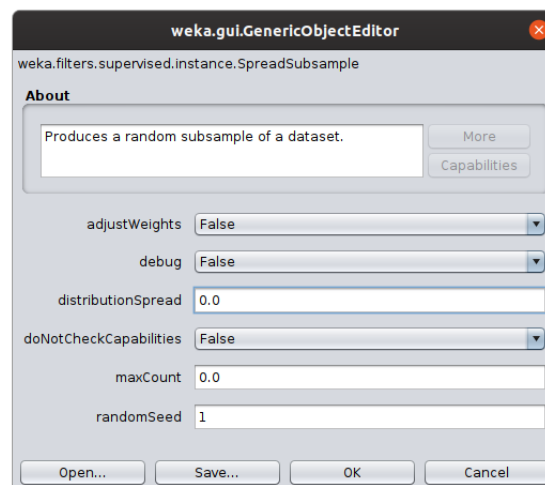


Figura 31: Configuración del filtro supervisado **SpreadSubsample**.

Como se puede observar en la fig. 31, el filtro **SpreadSubsample** cuenta con los siguientes parámetros:

- **adjustWeights** (False/True): si se pone a true, añade a la base de datos el campo **Weight** y le asigna un peso ponderado para mantener el mismo equilibrio entre clases que hubiera antes de aplicar el filtro (ver 3.2).
- **debug** (False/True): si se pone a true saca más información en el log al ejecutar el filtro.

- **distributionSpread** (num): 0 para no establecer proporción máxima,  $n|10 \geq n \geq 1$  para permitir un desequilibrio máximo entre clases de  $n : 1$ .
- **doNotCheckCapabilities** (False/True): habilita o deshabilita la comprobación de los datos. Afecta al rendimiento del filtro.
- **maxCount** (num): establece un límite máximo de patrones por clase. 0 para desactivar esta función.
- **randomSeed** (num): semilla para la selección aleatoria.

Para probar este filtro se utiliza la base de datos **breast-cancer** por ser su clase binaria y con una distribución 201-85 (fig. 32). En una primera prueba, con los parámetros por defecto, no se ha visto ningún cambio en la base de datos al aplicar el filtro. El primer cambio introducido ha sido el valor 50 en el parámetro **maxCount**, lo que ha producido subsampling que ha igualado el n° de patrones de cada clase a 50 (fig. 33).

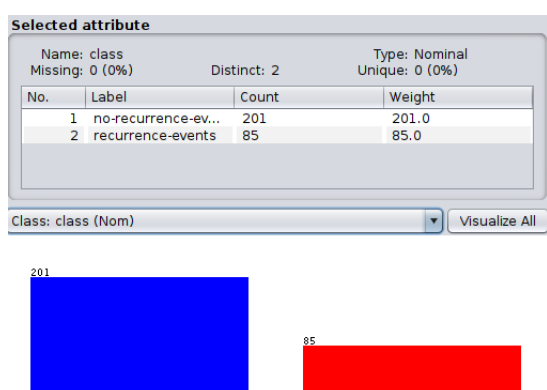


Figura 32: Distribución inicial de la clase en la base de datos **breast-cancer**.

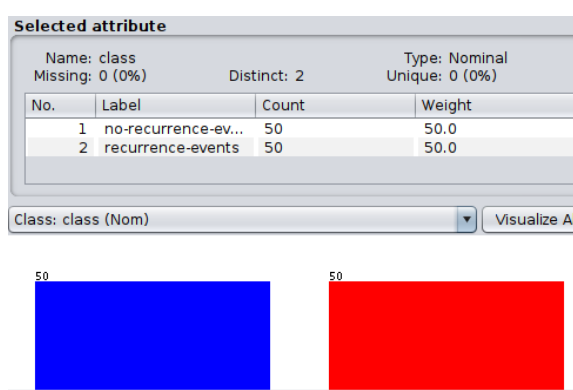


Figura 33: Resultado del filtro con **distributionSpread=0** y **maxCount=50**.

La segunda prueba ha consistido en fijar de nuevo **maxCount** a 0 y **distributionSpread** a 1. Esta configuración sirve para que la proporción máxima entre las clases sea 1 : 1, así que también equilibra la distribución de las clases, eliminando patrones de la clase mayoritaria hasta quedar tantos como tenga la minoritaria (fig. 34). Por último, se establece **distributionSpread** a 1.5 con el fin de obtener una proporción 3 : 2. Deberíamos tener 3 patrones de la clase **no-recurrence-events** por cada dos de la clase **recurrence-events**. Una vez aplicado el filtro con la configuración indicada (fig. 35), vemos que el resultado es el esperado: 127 frente a 85 patrones de cada clase.

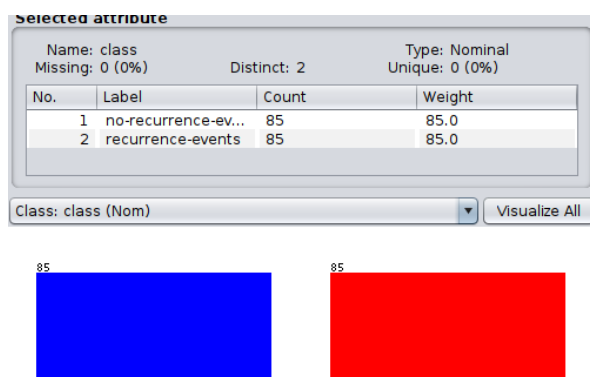


Figura 34: Resultado del filtro con **distributionSpread=1** y **maxCount=0**.

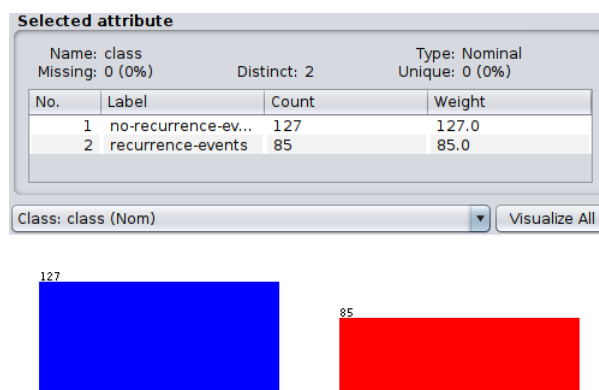


Figura 35: Resultado del filtro con **distributionSpread=1.5**.

## Parte II

# Práctica 2

## 4. Actividad 2-1

*Para esta práctica, utilice el conjunto de datos de altura de ola proporcionado en Moodle.*

*Describa las operaciones de preprocesamiento que ha realizado sobre la base de datos proporcionada y cómo queda la base de datos final ya preprocesada. Se deja a su elección el conjunto de técnicas a aplicar, así como el nivel de detalle y descripción que quiera dar a su trabajo.*

*Para probar rendimientos sobre su preprocesamiento, puede lanzar cualquier algoritmo de Weka, por ejemplo `classifiers.functions.Logistic`, y fijarse en la métrica “Correctly Classified Instances”. En la opción “Supplied test set” se indicaría el fichero del conjunto de test, mientras que el de entrenamiento corresponde al que se ha cargado desde la pestaña Preprocess.*

### 4.1. Introducción

En este ejercicio se van a poner en práctica los conceptos aprendidos sobre preprocesamiento de datos. Para ello se utilizará una base de datos de observaciones meteorológicas tomadas entre 2014 y 2016 a razón de 4 mediciones al día. El conjunto tiene 3559 patrones con 17 atributos numéricos y una clase nominal. Los atributos son:

- AIR: Temperatura del aire en la superficie ( $^{\circ}\text{K}$ ).
- PRES: Presión a nivel de superficie (Pa).
- RHUM: Humedad relativa a nivel de superficie (%).
- UWND: Velocidad del viento de oeste a este (eje X) (m/s).
- VWND: Velocidad del viento de sur a norte (eje Y) (m/s).
- WDIR: Dirección del viento en el sentido de las agujas del reloj (grad).
- WSPD: Velocidad del viento (m/s).
- GST: Velocidad de pico del viento (m/s).
- DPD: Periodo de ola dominante (s).
- APD: Periodo medio de ola dominante (s).
- MWD: Dirección de la que viene el periodo de ola dominante (grad).
- PRES: Presión a nivel de superficie (hPa).
- ATMP: Temperatura del aire en la parte superior de la boya ( $^{\circ}\text{C}$ ).
- WTMP: Temperatura a nivel del mar ( $^{\circ}\text{C}$ ).
- DEWP: Punto de rocío tomado a la misma altura que ATMP ( $^{\circ}\text{C}$ ).

- VIS: Visibilidad desde la boya (MN).
- TIDE: Nivel del agua por encima o por debajo de la media inferior del agua (Pies).

La variable de salida indica la altura de ola en las seis horas siguientes. Es discreta con valores Baja, Media, Moderada y MuyAlta.

Para valorar el preprocesamiento realizado, se generará a cada paso un clasificador logístico y se evaluará con un conjunto de pruebas con la misma estructura que contiene los datos correspondientes a los años 2017 y 2018. Este proceso se realiza desde la pestaña Classify de Weka Explorer. El clasificador seleccionado será Functions.Logistic con la configuración indicada en la figura 36 y utilizando como conjunto de test el archivo `alturaolatest.arff`, al que previamente se le habrá realizado el correspondiente tratamiento.

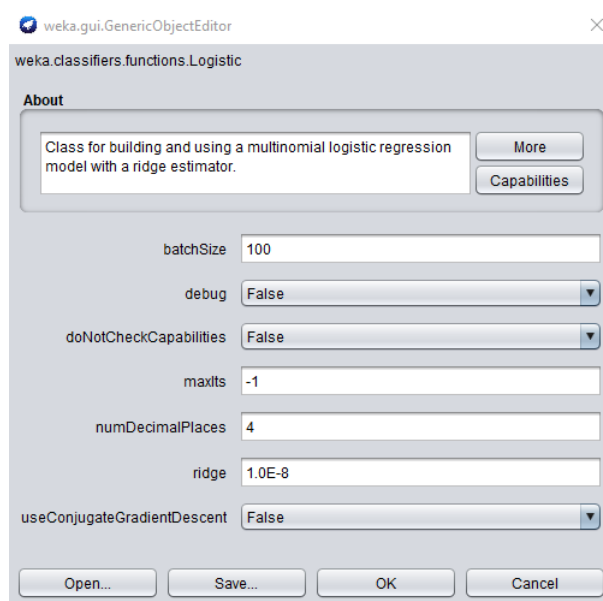


Figura 36: Configuración del clasificador logístico

## 4.2. Preparación

Como paso previo se han convertido los archivos `.csv` a `.arff`. Los dos archivos se han cargado en Weka para poder visualizar su contenido, como puede verse en las figuras 37 y 38.

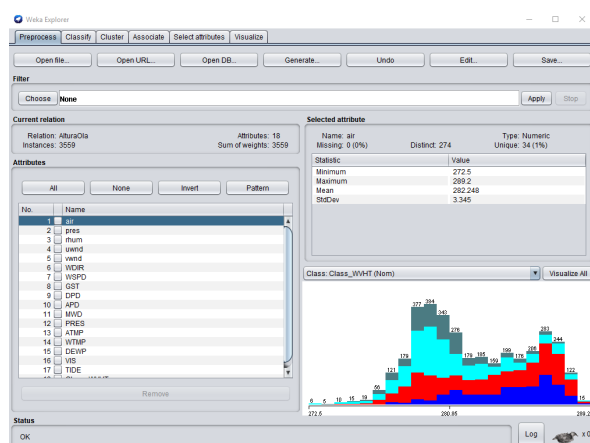


Figura 37: Captura de `alturaola.arff`.

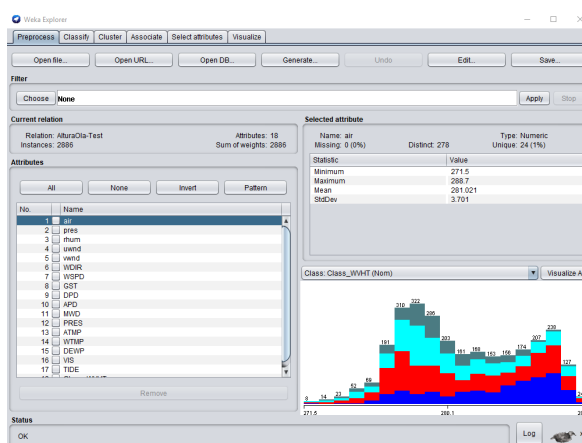


Figura 38: Captura de `alturaolatest.arff`.

Tras la conversión de los archivos y antes de empezar el tratamiento de los datos, se ha generado y testeado el clasificador logístico para ver el punto de partida. En la figura 39 se pueden observar los resultados: se tiene en el punto de partida un 71,9335 % de patrones correctamente clasificados y precisión mínima de 0,660 para la clase “Media”.

```

=== Summary ===

Correctly Classified Instances      2076          71.9335 %
Incorrectly Classified Instances    810          28.0665 %
Kappa statistic                    0.613
Mean absolute error                 0.1896
Root mean squared error             0.3132
Relative absolute error             51.8339 %
Root relative squared error         73.0895 %
Total Number of Instances          2886

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0,708    0,039    0,846     0,708    0,771     0,713    0,955    0,871    Baja
      0,733    0,194    0,660     0,733    0,695     0,526    0,853    0,694    Media
      0,696    0,134    0,666     0,696    0,681     0,555    0,878    0,673    Moderada
      0,749    0,030    0,814     0,749    0,780     0,744    0,964    0,865    MuyAlta
Weighted Avg.  0,719    0,117    0,728     0,719    0,721     0,611    0,901    0,755

=== Confusion Matrix ===

  a   b   c   d   <-- classified as
477 194   3   0 |   a = Baja
 82 719 178   2 |   b = Media
  3 168 557  72 |   c = Moderada
  2   8  98 323 |   d = MuyAlta

```

Figura 39: Resultados del clasificador logístico antes del tratamiento de datos.

### 4.3. Datos perdidos

En primer lugar se eliminan los tres atributos con 100 % de datos perdidos: TIDS, VIS y MWD tanto en el conjunto de entrenamiento como en el de test y se genera de nuevo el clasificador logístico. Como se puede observar en la figura 40, la eliminación de los atributos con 100 % de datos perdidos no tiene ningún efecto.

```

Correctly Classified Instances      2076          71.9335 %
Incorrectly Classified Instances    810          28.0665 %

```

Figura 40: Resultados tras eliminar TIDS, VIS y MWD.

A continuación se estudian los campos con valores perdidos que son en entrenamiento DPD, APD, PRES y DEWP con 8 (<1 %), 8 (<1 %), 553 (16 %) y 1189 (33 %) patrones con datos perdidos respectivamente y en pruebas WSPD, GST, DPD, APD, PRES, ATMP, WTMP Y DEWP. Para estos casos se reemplazan los valores perdidos por la media de la clase calculada sobre el conjunto de entrenamiento tanto en el conjunto de entrenamiento como en el conjunto de generalización (ver cuadro 5). Se realiza este procedimiento siguiendo el consejo del archivo *Imputación de valores perdidos mas justa en training y test.pdf*, aunque me preocupa en qué medida contribuirá al rendimiento del modelo cuando se exponga a datos nuevos.

Atributo	Datos perdidos				Medias en entrenamiento			
	Entrenamiento		Test		Baja	Media	Moderada	MuyAlta
WSPD	0	(0 %)	1	(<1 %)	4,6	6,6	8,4	11,9
GST	0	(0 %)	9	(<1 %)	5,6	8,1	10,5	14,9
DPD	8	(<1 %)	21	(1 %)	11,5	10,3	10,8	11,7
APD	8	(<1 %)	21	(1 %)	6,5	6,7	7,0	7,7
PRES	553	(16 %)	6	(<1 %)	1017,8	1013,1	1008,7	1007,2
ATMP	0	(0 %)	3	(<1 %)	11,5	9,9	7,8	7,4
WTMP	0	(0 %)	7	(<1 %)	12,3	10,8	8,8	8,2
DEWP	1189	(33 %)	2886	(100 %)	8,9	7,1	4,8	4,8

Cuadro 5: Datos perdidos y medias de atributos en entrenamiento por clase.

Una vez reemplazados todos los datos perdidos se obtienen en el clasificador los resultados mostrados en la figura 41, en los que se observa una mejora en las métricas del clasificador. Por ejemplo, el CCR sube casi un 3 %, la mínima sensibilidad aumenta ligeramente y el área bajo la curva ROC aumenta para tres de las cuatro clases.

```

=== Summary ===

Correctly Classified Instances      2160           74.8441 %
Incorrectly Classified Instances    726           25.1559 %
Kappa statistic                    0.654
Mean absolute error                 0.1766
Root mean squared error             0.2972
Relative absolute error             48.2634 %
Root relative squared error         69.3595 %
Total Number of Instances          2886

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0,792    0,052    0,823     0,792    0,807      0,750    0,964     0,891     Baja
      0,773    0,170    0,701     0,773    0,735      0,590    0,884     0,756     Media
      0,675    0,101    0,720     0,675    0,697      0,586    0,900     0,741     Moderada
      0,761    0,031    0,810     0,761    0,785      0,749    0,969     0,872     MuyAlta
Weighted Avg.   0,748    0,103    0,751     0,748    0,749      0,650    0,920     0,801

=== Confusion Matrix ===

  a   b   c   d  <-- classified as
534 140   0   0 |  a = Baja
109 758 112   2 |  b = Media
  5 180 540  75 |  c = Moderada
  1   4   98 328 |  d = MuyAlta

```

Figura 41: Resultados tras reemplazar los datos perdidos

#### 4.4. Unificación de medidas

El siguiente paso a seguir será la estandarización de las unidades de medida de los atributos que representan la misma magnitud. En nuestro dataset tenemos dos pares de atributos que representan temperatura y presión respectivamente, y utilizan diferentes unidades ( $^{\circ}\text{C}/^{\circ}\text{K}$  por un lado y  $\text{HPas}/\text{Pas}$ ), por lo que correspondería a unificarlas. Las transformaciones se han realizado mediante el filtro `MathExpression`. Primero se han pasado a Pascales los valores del atributo `PRES` que estaban en `HPas` y seguidamente se han pasado a  $^{\circ}\text{C}$  los valores del atributo `air` que se encontraban en  $^{\circ}\text{K}$ . Como era de esperar, estas operaciones no han tenido ningún efecto sobre el resultado del clasificador.



## 4.5. Selección de características

En esta etapa del preprocesamiento vamos a intentar detectar los atributos que influyen muy poco en la clase (atributos irrelevantes) para así eliminarlos y simplificar el modelo. Del mismo modo, intentaremos identificar pares de atributos con una correlación alta (atributos redundantes) para prescindir, de entre los dos, de el que menos influya en la clase.

### 4.5.1. Análisis de atributos irrelevantes

Para medir la influencia de cada atributo sobre la clase del dataset en Weka utilizaremos el evaluador de atributos `CorrelationAttributeEval` en combinación con `Ranker` como método de búsqueda en la pestaña “Select attributes” del explorador de Weka. La salida de este análisis se puede ver en la figura 42.

```

=== Attribute Selection on all input data ===

Search Method:
    Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 15 Class_WVHT):
    Correlation Ranking Filter
Ranked attributes:
0.3051  8 GST
0.2977 10 APD
0.2926  7 WSPD
0.2847  2 pres
0.2788 11 PRES
0.2486 13 WTMP
0.2374 14 DEWP
0.2358 12 ATMP
0.2292  1 air
0.1945  9 DPD
0.1434  5 vwnd
0.0837  6 WDIR
0.0658  4 uwnd
0.0397  3 rhum

Selected attributes: 8,10,7,2,11,13,14,12,1,9,5,6,4,3 : 14

```

Figura 42: Resultados de `CorrelationAttributeEval`

Lo siguiente será eliminar uno por uno atributos desde el menos influyente (`rhum`) hacia arriba comprobando el resultado del clasificador logístico hasta que comience a empeorar. Se eliminan `rhum` y `uwnd` para alcanzar un  $CCR = 75,23\%$  (fig. 43), pero al eliminar `WDIR` vemos que empeora, por lo que paramos ahí. No estoy seguro de si este paso para reducir la complejidad conviene o no conviene en este caso ya que a pesar de la leve mejora del CCR, vemos en la matriz de confusión que la única clase que ha mejorado en nº de patrones bien clasificados es `Media`, que ya era la que mejor clasificaba, a costa de un empeoramiento en el resto.

```

=== Summary ===

Correctly Classified Instances      2171           75.2252 %
Incorrectly Classified Instances    715           24.7748 %
Kappa statistic                    0.6588
Mean absolute error                0.179
Root mean squared error            0.2982
Relative absolute error            48.9215 %
Root relative squared error        69.5945 %
Total Number of Instances          2886

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
          0,786   0,046   0,840     0,786   0,812     0,758   0,963   0,887   Baja
          0,790   0,173   0,702     0,790   0,743     0,602   0,882   0,741   Media
          0,675   0,100   0,721     0,675   0,697     0,587   0,900   0,738   Moderada
          0,756   0,031   0,811     0,756   0,783     0,747   0,969   0,872   MuyAlta
Weighted Avg.   0,752   0,102   0,756     0,752   0,753     0,656   0,919   0,794

=== Confusion Matrix ===

  a  b  c  d  <-- classified as
530 144  0  0 |  a = Baja
 96 775 108  2 |  b = Media
  5 181 540 74 |  c = Moderada
  0  4 101 326 |  d = MuyAlta

```

Figura 43: Resultados tras eliminar atributos irrelevantes

#### 4.5.2. Análisis de atributos redundantes

A continuación se estudiará la correlación entre atributos. Para realizar este análisis en Weka utilizamos el evaluador de atributos **PrincipalComponents** junto con el buscador **Ranker**. Podemos ver la tabla de correlaciones obtenida en el cuadro 6, en el que se han marcado en rojo, naranja y amarillo las correlaciones con valor absoluto por encima de 0,6 según su importancia (rojo la más importante).

	air	pres	vwnd	WDIR	WSPD	GST	DPD	APD	PRES	ATMP	WTMP	DEWP
air	1	0,23	-0	0,1	-0,15	-0,18	-0,34	-0,44	0,23	0,98	0,94	0,77
pres	0,23	1	-0,2	0,25	-0,34	-0,37	-0,26	-0,43	0,95	0,26	0,26	0,22
vwnd	-0	-0,2	1	-0,15	0,31	0,31	0,07	0,13	-0,2	-0,01	-0,12	0,08
WDIR	0,1	0,25	-0,15	1	-0,06	-0,07	-0,04	-0,09	0,17	0,14	0,2	0,05
WSPD	-0,15	-0,34	0,31	-0,06	1	0,99	0,02	0,1	-0,34	-0,17	-0,21	-0,18
GST	-0,18	-0,37	0,31	-0,07	0,99	1	0,04	0,14	-0,36	-0,2	-0,23	-0,21
DPD	-0,34	-0,26	0,07	-0,04	0,02	0,04	1	0,71	-0,27	-0,33	-0,31	-0,3
APD	-0,44	-0,43	0,13	-0,09	0,1	0,14	0,71	1	-0,45	-0,44	-0,44	-0,41
PRES	0,23	0,95	-0,2	0,17	-0,34	-0,36	-0,27	-0,45	1	0,24	0,25	0,2
ATMP	0,98	0,26	-0,01	0,14	-0,17	-0,2	-0,33	-0,44	0,24	1	0,95	0,76
WTMP	0,94	0,26	-0,12	0,2	-0,21	-0,23	-0,31	-0,44	0,25	0,95	1	0,66
DEWP	0,77	0,22	0,08	0,05	-0,18	-0,21	-0,3	-0,41	0,2	0,76	0,66	1

Cuadro 6: Correlaciones entre atributos

El proceso de eliminación de atributos correlacionados será el siguiente:

1. Elegir un par atributos con correlación alta.
2. Del par elegido, eliminar el atributo menos correlado con la clase según la figura 42.
3. Generar y evaluar el clasificador logístico.

a) Si el resultado mejora, se mantiene la eliminación del atributo.

b) Si no mejora, se restituye el atributo.

El primer par procesado ha sido **pres-PRES** porque tiene una altísima correlación y no es de los atributos más influyentes sobre la clase, pero principalmente porque representan la misma magnitud. Tras la eliminación de **PRES**, el clasificador apenas cambia. Sólo mejora muy ligeramente, lo que nos da una idea de lo poco que servía el atributo eliminado, ganando el modelo en simplicidad.

El segundo par procesado ha sido **WSPD-GST** por tener la más alta correlación. De los dos atributos, se ha eliminado **WSPD** por ser el que menos influye sobre la clase. Tras la eliminación, el clasificador apenas ha cambiado. Sólo 8 patrones correctamente clasificados menos, lo que nos lleva a un  $CCR$  de 75,09 % con una sensibilidad mínima de 0,679 para la clase «Moderada».

La siguiente correlación más alta se da entre el par **air-ATMP**. **air** está fuertemente correlacionado con **WTMP** y **DEWP** también, y al mismo tiempo es el menos correlacionado con la clase de los cuatro, por lo que se va a eliminar. Tras la eliminación vemos que los estadísticos se ven muy poco afectados:  $CCR = 75,05\%$  y sensibilidad mínima de 0,686 para la clase «Moderada».

El par **WTMP-ATMP** es el siguiente más correlacionado. Como **ATMP** es el atributo que menos influye sobre la clase, se elimina éste. El clasificador empeora significativamente pasando a un  $CCR = 74,22\%$  por lo que se decide no mantener esta eliminación. La eliminación de **WTMP** produce un efecto similar ( $CCR = 74,64\%$ ), por lo tanto se mantendrán estos dos atributos.

El siguiente par estudiado ha sido **APD-DPD**, ya con correlación más baja. El atributo **DPD** es mucho menos influyente en la clase, por lo que se decide eliminarlo consiguiendo un ligero aumento de rendimiento en el porcentaje de patrones bien clasificados ( $CCR = 75,16\%$ ) así como en la sensibilidad mínima (0,690).

=== Summary ===

Correctly Classified Instances	2169	75.1559 %
Incorrectly Classified Instances	717	24.8441 %
Kappa statistic	0.6577	
Mean absolute error	0.1822	
Root mean squared error	0.3002	
Relative absolute error	49.7917 %	
Root relative squared error	70.0559 %	
Total Number of Instances	2886	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,774	0,042	0,847	0,774	0,809	0,756	0,963	0,881	Baja
	0,784	0,173	0,700	0,784	0,739	0,596	0,879	0,740	Media
	0,690	0,106	0,713	0,690	0,701	0,590	0,898	0,736	Moderada
	0,756	0,029	0,821	0,756	0,787	0,753	0,967	0,868	MuyAlta
Weighted Avg.	0,752	0,103	0,756	0,752	0,752	0,655	0,917	0,791	

=== Confusion Matrix ===

a	b	c	d	<-- classified as
522	152	0	0	a = Baja
89	769	122	1	b = Media
5	173	552	70	c = Moderada
0	5	100	326	d = MuyAlta

Figura 44: Resultados tras quitar atributos redundantes

El último par es **DEWP-WTMP**. **DEWP** influye menos en la clase y además, ya se comprobó la influencia negativa de la eliminación de **WTMP** en el rendimiento del clasificador, por lo que se elimina **DEWP**. Esta

eliminación ha tenido un impacto muy negativo sobre el clasificador  $CCR = 71,86\%$ . Por este motivo, la eliminación se revierte y no se eliminan más atributos.

En resumen, se han eliminado los atributos **pres**, **WSPD**, **air** y **DPD** reduciendo la complejidad del modelo. Al mismo tiempo, los indicadores del modelo como  $CCR$  y mínima sensibilidad han mejorado, alcanzando valores  $75,16\%$  y  $0,690$  respectivamente (ver figura 44).

## 4.6. Normalización

El siguiente paso del preprocesamiento consiste en normalizar los datos tanto de entrenamiento como de generalización. Queremos llevar cada atributo a un rango  $[0 - 1]$  en entrenamiento y luego aplicar la misma transformación al conjunto de generalización. Para ello, aplicaremos la fórmula 1 a cada atributo de ambos conjuntos pero utilizando siempre los máximos y mínimos del conjunto de entrenamiento, tal como aparecen en el cuadro 7. El proceso se realizará mediante el filtro **MathExpression**, campo a campo. Para el conjunto de entrenamiento se obtiene el mismo resultado utilizando el filtro **Normalize**.

$$a^* = \frac{a - a_{min}}{a_{max} - a_{min}} \quad (1)$$

	pres	vwnd	WDIR	GST	APD	ATMP	WTMP
min	96020	-17,9	0	0,4	3,10	-0,9	6
max	103850	21,1	359	26,1	11,09	15,8	16,1
ran	7830	39,0	359	25,7	7,99	16,7	10,1

Cuadro 7: Valores máximos, mínimos y rangos por atributo.

Como se puede comprobar en la figura 45, el proceso de normalización ha provocado un empeoramiento generalizado del rendimiento del clasificador, con casi un  $4\%$  menos de patrones correctamente clasificados. No tengo claro que necesariamente cualquier clasificador deba ser mejor con datos normalizados. A la vista de que el clasificador logístico que estamos utilizando para evaluar los efectos del preprocesamiento no se ve beneficiado por la normalización de nuestro conjunto de datos, continúo con los datos sin normalizar.

```

=== Summary ===

Correctly Classified Instances      2074           71.8642 %
Incorrectly Classified Instances    812           28.1358 %
Kappa statistic                    0.612
Mean absolute error                 0.1916
Root mean squared error             0.3133
Relative absolute error             52.3619 %
Root relative squared error         73.1132 %
Total Number of Instances          2886

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0,717    0,044    0,831    0,717    0,770      0,709    0,955    0,862    Baja
      0,727    0,192    0,661    0,727    0,692      0,523    0,852    0,700    Media
      0,698    0,136    0,663    0,698    0,680      0,553    0,877    0,678    Moderada
      0,742    0,026    0,831    0,742    0,784      0,751    0,967    0,869    MuyAlta
Weighted Avg.   0,719    0,117    0,727    0,719    0,721      0,609    0,900    0,757

=== Confusion Matrix ===

  a   b   c   d   <-- classified as
483 188   3   0 |   a = Baja
 93 713 175   0 |   b = Media
  5 172 558  65 |   c = Moderada
  0   6 105 320 |   d = MuyAlta

```

Figura 45: Resultados tras normalización de datos

## 4.7. Datos extremos

Para la detección de valores extremos, haremos uso del filtro `InterquartileRange`. Este filtro añade a la base de datos un par de atributos para indicar si el patrón es un dato extremo o atípico (outlier). Según las transparencias, se consideran valores extremos aquellos que estén a más del doble del rango intercuartílico por debajo del cuartil 1 o por encima del cuartil 3. Aplicando la configuración correspondiente al filtro de Weka, podemos ver que aparecen 33 patrones con datos extremos tal como se muestra en la figura 46. A continuación eliminaremos los patrones con `ExtremeValue=yes` utilizando el

Name: ExtremeValue Missing: 0 (0%)		Distinct: 2	Type: Nominal Unique: 0 (0%)
No.	Label	Count	Weight
1	no	3526	3526.0
2	yes	33	33.0

Figura 46: Detección de valores extremos con Weka

filtro `RemoveWithValues` y posteriormente eliminaremos el atributo `ExtremeValues` para poder ejecutar el clasificador logístico con el conjunto de generalización. En dicha ejecución se han obtenido los valores que aparecen en la figura 47.

```

=== Summary ===

Correctly Classified Instances      2165          75.0173 %
Incorrectly Classified Instances    721          24.9827 %
Kappa statistic                    0.6558
Mean absolute error                 0.1817
Root mean squared error             0.3
Relative absolute error             49.6623 %
Root relative squared error         70.0402 %
Total Number of Instances          2886

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
          0,774   0,042   0,847     0,774   0,809     0,756   0,963   0,881   Baja
          0,785   0,174   0,699     0,785   0,739     0,595   0,879   0,741   Media
          0,684   0,106   0,712     0,684   0,698     0,585   0,898   0,737   Moderada
          0,756   0,030   0,815     0,756   0,785     0,749   0,967   0,868   MuyAlta
Weighted Avg.   0,750   0,103   0,755     0,750   0,751     0,653   0,917   0,792

=== Confusion Matrix ===

  a   b   c   d   <-- classified as
522 152   0   0 |   a = Baja
 89 770 121   1 |   b = Media
  5 175 547  73 |   c = Moderada
  0   5 100 326 |   d = MuyAlta

```

Figura 47: Resultados tras eliminación de valores extremos

De nuevo observamos un ligero empeoramiento respecto a los resultados de partida (ver fig. 44), por lo que no parece conveniente, en este caso, eliminar los valores extremos. Esto parece lógico ya que también hay valores de este tipo en el conjunto de test, y si el modelo no los aprende, difícilmente podrá luego clasificarlos en generalización.

## Parte III

# Práctica 3

## 5. Actividad 3-1

*Escoja una de las bases de datos de clasificación para el trabajo de las dispuestas en Moodle (Breast Cancer, Dermatology, Fantasma, Glass, Vehicle, Wine, Zoo).*

*Se entiende que además de pasarla a formato .arff ya ha aplicado el preprocesamiento necesario en función del fichero “**Pistas sobre los datasets con posible preprocesamiento a simple vista.pdf**”, en el caso de que sea una de las bases de datos que lo requiera.*

- *Aplique el preprocesamiento adicional (si se puede aplicar sobre: 1) reemplazamiento de datos perdidos, 2) normalización y 3) paso de nominal a binario u ordinal a numérico. Explique el preprocesamiento que haya llevado a cabo en los aspectos citados, y de no tener que hacerlo explique también por qué.*

La base de datos seleccionada (Criaturas de competición Kaggle: dataset371.csv) contiene los siguientes campos:

- id: es un identificador único numérico autoincremental.
- bone\_length: longitud media del hueso en la criatura, normalizado entre 0 y 1.
- rotting\_flesh: Porcentaje de carne podrida en la criatura, normalizado entre 0 y 1.
- hair\_length: longitud media del pelo, normalizado entre 0 y 1.
- has\_soul: porcentaje de alma en la criatura, entre 0 y 1.
- color: color dominante de la criatura, nominal con seis valores posibles.
- type: clase asignada, con valores “Ghost”, “Goblin” y “Ghoul”.

El primer paso a simple vista es eliminar el campo id, ya que los valores únicos no nos dan ninguna información acerca de la clase del patrón. No se observan datos perdidos y los atributos numéricos ya se encuentran normalizados. En este caso, los dos algoritmos que se utilizarán posteriormente soportan atributos con valores tanto nominales como numéricos continuos, por lo que no es necesario realizar ninguna discretización o reemplazo por números.

Respecto a la selección de atributos, gracias al filtro `CorrelationAttributeEval` podemos ver que el atributo `color` tiene muy poca influencia sobre la clase (ver fig. 48) por lo que se procede a su eliminación. También se han estudiado las posibles correlaciones entre los atributos, pero ninguna es significativa (ver fig. 49) No se han encontrado valores extremos aunque sí 5 outliers, pero estos no requieren tratamiento.

Ranked attributes:		Correlation matrix			
0.4078	3 hair_length	1	-0.04	0.35	0.38
0.3821	4 has_soul	-0.04	1	-0.22	-0.13
0.3069	1 bone_length	0.35	-0.22	1	0.47
0.2616	2 rotting_flesh	0.38	-0.13	0.47	1
0.0303	5 color				

Figura 48: Influencia de los atributos en la clase

Figura 49: Matriz de correlaciones entre atributos



## 6. Actividad 3-2

Con la base de datos escogida anteriormente, use el algoritmo de clasificación **KNN** (IBK en Weka) con un 10-fold crossvalidation. Use un valor de vecinos  $k = 3$  dejando por defecto el resto de parámetros.

- Interprete la salida en cuanto a los valores de las métricas que proporciona Weka.

Tenga en cuenta si se clasifican bien todas las clases de su problema (TP Rate por clase) y fíjese también en la matriz de confusión.

Para explicar los resultados, haga uso de tablas donde se muestren los valores que está interpretando.

Para este ejercicio se ha utilizado la base de datos preprocesada del ejercicio anterior. Se va a ejecutar el algoritmo KNN (llamado IBk en Weka) considerando los 3 vecinos más cercanos. Esto significa que se va a dividir el conjunto de datos en 10 partes, los patrones de cada una de las cuales se clasificará en función de la clase de los 3 vecinos más cercanos del subconjunto complementario. Para determinar los vecinos más cercanos se empleará la distancia euclídea y un algoritmo de búsqueda por fuerza bruta llamado **LinearNNSearch** en Weka. La configuración empleada se puede ver en la figura 50.

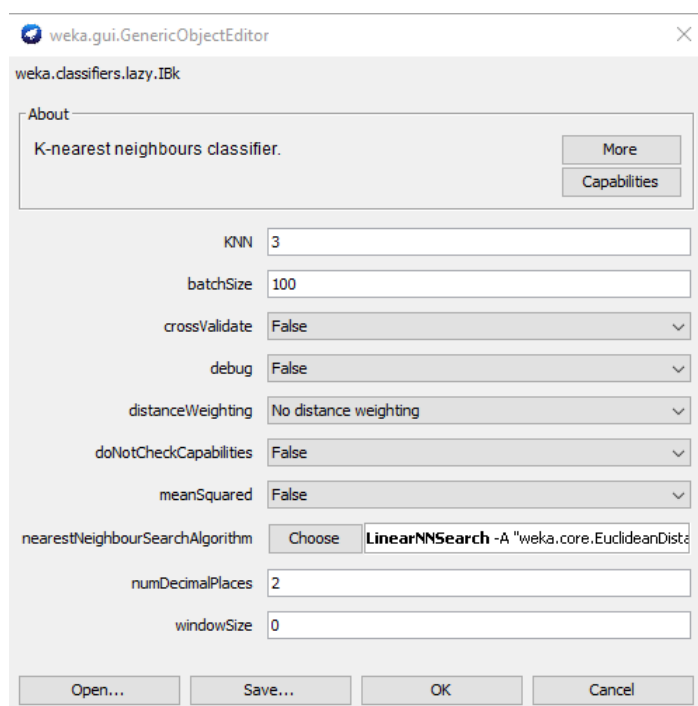


Figura 50: Configuración del clasificador IBk

Podemos ver los resultados que arroja el clasificador en la figura 51. Hay un 70,89 % de patrones bien clasificados (263) frente a un 29,11 % de patrones mal clasificados (108). El valor 0,5631 del estadístico Kappa nos indica que el clasificador está aproximadamente a medio camino entre el azar ( $\kappa = 0$ ) y un clasificador perfecto ( $\kappa = 1$ ).

Los TP Rates también nos indican que el clasificador es mejor que uno al azar, que arrojaría valores de 0,333 (igual para todas las clases). En estos datos podemos observar también que se reconoce peor la clase Goblin, lo que se confirma con el valor de precisión mínima —que corresponde a esta clase— y la matriz de confusión, donde podemos ver que sólo 73 de los 125 goblins han sido correctamente clasificados.

```
=== Summary ===

Correctly Classified Instances      263          70.8895 %
Incorrectly Classified Instances    108          29.1105 %
Kappa statistic                    0.5631
Mean absolute error                0.2301
Root mean squared error            0.3908
Relative absolute error            51.8003 %
Root relative squared error        82.9263 %
Total Number of Instances          371

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,829   0,087   0,815    0,829   0,822     0,739   0,909    0,801    Ghost
      0,584   0,207   0,589    0,584   0,586     0,377   0,731    0,530    Goblin
      0,721   0,145   0,727    0,721   0,724     0,577   0,855    0,718    Ghoul
Weighted Avg.   0,709   0,147   0,708    0,709   0,708     0,561   0,830    0,681

=== Confusion Matrix ===

  a  b  c  <-- classified as
97 19  1 | a = Ghost
18 73 34 | b = Goblin
 4 32 93 | c = Ghoul
```

Figura 51: Resultados del clasificador IBk con un 10-fold

## 7. Actividad 3-3

*Con la base de datos escogida anteriormente, ejecute el algoritmo SimpleLogistic con 10-fold crossvalidation.*

- *Analice los modelos obtenidos, las variables que podrían ser más influyentes (valores  $\beta$ ) variables que no se usan y métricas. Use tablas para explicar los resultados de manera que haya una lectura legible.*

En esta ocasión utilizaremos el clasificador **SimpleLogistic**. Un clasificador de tipo logístico funciona aplicando una transformación logística a la suma ponderada de los valores del patrón para cada clase. La diferencia entre Logistic y SimpleLogistic es que Logistic calcula la última clase como el complemento de la suma del resto de clases hasta 1. Se asigna al patrón la clase correspondiente al resultado más alto. En la figura 52 podemos ver el formulario de configuración del clasificador SimpleLogistic de Weka.

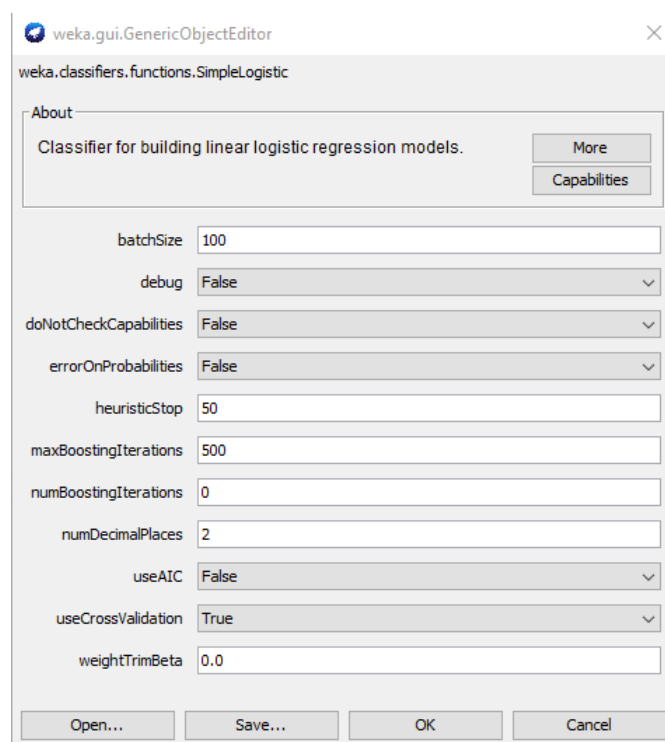


Figura 52: Configuración del clasificador SimpleLogistic

En la figura 53 y el cuadro 8 podemos ver las sumas ponderadas de cada clase, donde los pesos (valores  $\beta$ ) nos dan una idea de la influencia de cada variable independiente en la determinación de la clase. Así podemos ver que para la clase Ghost, los atributos bone\_length, hair\_length y has\_soul influyen muy negativamente mientras que rooting\_flesh lo hace positivamente, justo lo contrario que ocurre en la clase Ghoul. En la clase Goblin vemos que la variable bone\_length no influye, hair\_length y has\_soul lo hacen muy levemente y rooting\_flesh influye negativamente.

```

Class Ghost :
7.19 +
[bone_length] * -5.85 +
[rotting_flesh] * 3.46 +
[hair_length] * -7.67 +
[has_soul] * -7.41

Class Goblin :
2.26 +
[rotting_flesh] * -4.75 +
[hair_length] * 0.63 +
[has_soul] * 0.23

Class Ghoul :
-6.97 +
[bone_length] * 4.64 +
[rotting_flesh] * -2.78 +
[hair_length] * 6 +
[has_soul] * 5.68

```

Figura 53: Pesos de los atributos para cada clase

	$\beta_0$ sesgo	$\beta_1$ bone_length	$\beta_2$ rotting_flesh	$\beta_3$ hair_length	$\beta_4$ has_soul
Ghost	7,19	-5,85	3,46	-7,67	-7,41
Goblin	2,26	0	-4,75	0,63	0,23
Ghoul	-6,97	4,64	-2,78	6	5,68

Cuadro 8: Pesos de los atributos para cada clase

Los resultados del clasificador logístico (fig. 54) son sólo ligeramente mejores que los del clasificador KNN del ejercicio anterior. Vemos que hay 9 patrones correctamente clasificados más que en el anterior, lo que lleva a un  $CCR = 73,32\%$ . En línea con esto, tenemos también un índice Kappa superior:  $\kappa = 0,5996$ .

Por su parte, los TP Rates nos indican que, mientras que ha mejorado el reconocimiento de patrones de clase Ghost y Ghoul, ha empeorado el reconocimiento de patrones Goblin, que ya era la que peor clasificaba. Esto se refuerza con el descenso de la sensibilidad mínima a 0,536 y la bajada a 67 goblins bien clasificados de 125.

```
=== Summary ===
```

```

Correctly Classified Instances      272      73.3154 %
Incorrectly Classified Instances    99      26.6846 %
Kappa statistic                    0.5996
Mean absolute error                 0.2469
Root mean squared error             0.3411
Relative absolute error             55.5859 %
Root relative squared error         72.3816 %
Total Number of Instances          371

```

```
=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,880	0,098	0,805	0,880	0,841	0,764	0,973	0,947	Ghost
	0,536	0,146	0,650	0,536	0,588	0,411	0,818	0,646	Goblin
	0,791	0,157	0,729	0,791	0,758	0,623	0,906	0,830	Ghoul
Weighted Avg.	0,733	0,135	0,726	0,733	0,727	0,596	0,897	0,805	

```
=== Confusion Matrix ===
```

```

a  b  c  <-- classified as
103 13  1 | a = Ghost
 21 67 37 | b = Goblin
  4 23 102 | c = Ghoul

```

Figura 54: Resultados del clasificador SimpleLogistic con un 10-fold

## Parte IV

# Práctica 4

## 8. Actividad 4-1

*Escoja una de las bases de datos de clasificación para el trabajo de las dispuestas en Moodle (Breast Cancer, Dermatology, Fantasmas, Glass, Vehicle, Wine, Zoo).*

*Se entiende que además de pasarla a formato .arff ya ha aplicado el preprocesamiento necesario en función del fichero “**Pistas sobre los datasets con posible preprocesamiento a simple vista.pdf**”, en el caso de que sea una de las bases de datos que lo requiera.*

- *Cargue la base de datos y ejecute el algoritmo **C4.5** usando un 75 % para entrenar y un 25 % para generalizar, con los parámetros por defecto.*
- *Analice y muestre el árbol obtenido con los parámetros por defecto: nodo principal, número de nodos u hojas, variables presentes y omitidas. Comente también los resultados de las métricas obtenidas.*

Para la realización de esta actividad se va a utilizar la misma base de datos preprocesada de la práctica 3 (fantasmas). El objetivo es construir en Weka un árbol de decisión mediante el algoritmo C4.5, para lo cuál utilizaremos el clasificador llamado “J48” en Weka. Tal como indica el enunciado, se divide el conjunto en dos, seleccionando un 75 % (278 patrones) para la construcción del árbol (train) y el 25 % restante (93 patrones) para generalización (test). Weka realiza esta división manteniendo idénticas frecuencias de las clases en cada subconjunto.

El algoritmo construye el árbol mediante un proceso recursivo en el que, para establecer cada nodo se busca el atributo con mayor porcentaje de ganancia de información ( $I(C, X_i)/H(X_i)$ ), generando una partición en dos del conjunto de datos. En caso de que el atributo sea continuo, el algoritmo es capaz de determinar el mejor umbral. Para cada división del conjunto se añade una arista desde el nodo generado y se repite el proceso con el subconjunto correspondiente. Cuando todos los patrones asociados aun determinado recorrido en profundidad tienen la misma clase, se añade un nodo hoja. El algoritmo C4.5 también permite realizar podas que o bien sustituyen un sub-árbol por un nodo hoja o bien por una parte éste (elevación de sub-árbol). Es interesante mencionar que con este algoritmo, a diferencia de C3, sí que podemos encontrar varias veces el mismo atributo al realizar un recorrido del árbol en profundidad, al menos en el caso de atributos continuos y con diferentes umbrales.

Tal como podemos observar en la figura 55, en este caso de estudio, el atributo de mayor ganancia es hair\_length, por tanto se establece como nodo raíz. El árbol generado cuenta con 53 nodos, 27 de los cuáles son nodos hoja. Cada nodo hoja indica la cantidad de patrones que, siendo de la clase indicada, han alcanzado el nodo así como la cantidad de patrones de otras clases que lo han alcanzado, en caso de haber alguno. Los nodos hoja muestran los resultados para todos los patrones, tanto de generalización como de entrenamiento.

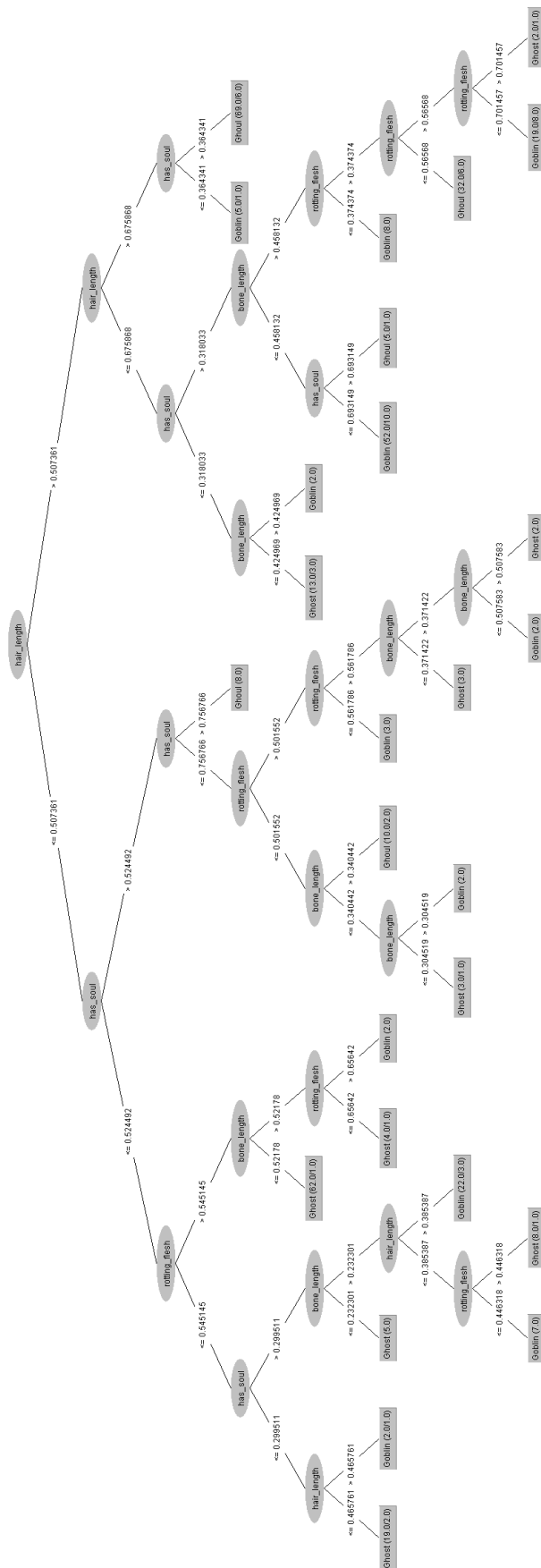


Figura 55: Árbol de decisión obtenido mediante C4.5

De los resultados del clasificador (fig. 56) podemos comentar que se han clasificado correctamente 65 patrones (69,89 %) de un total de 93 del conjunto de generalización. El estadístico Kappa es  $\kappa = 0,5473$ , comparable aunque peor que el obtenido en los ensayos con otros tipos de clasificadores para el mismo conjunto de datos. La clase que peor se reconoce es de nuevo Goblin, con un TP Rate de 0,618, más elevado que el obtenido en los clasificadores anteriores, aunque hay que observar que en este caso, el FP Rate también es más alto, lo que da el peor resultado en AUC para todos los clasificadores utilizados con este conjunto. Esta comparación no es del todo justa porque se ha utilizado una técnica distinta para la generalización. En la matriz de confusión podemos ver que sólo 21 de los 34 Goblins han sido correctamente clasificados.

```

=== Summary ===

Correctly Classified Instances      65           69.8925 %
Incorrectly Classified Instances    28           30.1075 %
Kappa statistic                    0.5473
Mean absolute error                 0.234
Root mean squared error            0.4427
Relative absolute error            52.6526 %
Root relative squared error        93.8336 %
Total Number of Instances         93

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
          0,724   0,125   0,724     0,724   0,724     0,599   0,797   0,616   Ghost
          0,618   0,220   0,618     0,618   0,618     0,397   0,623   0,467   Goblin
          0,767   0,111   0,767     0,767   0,767     0,656   0,822   0,657   Ghoul
Weighted Avg.   0,699   0,155   0,699     0,699   0,699     0,544   0,742   0,575

=== Confusion Matrix ===

  a  b  c  <-- classified as
21  8  0 | a = Ghost
 6 21  7 | b = Goblin
 2  5 23 | c = Ghoul

```

Figura 56: Resultados del clasificador

## 9. Actividad 4-2

Escoja una de las bases de datos de clasificación para el trabajo de las dispuestas en Moodle (*Breast Cancer, Dermatology, Fantasma, Glass, Vehicle, Wine, Zoo*).

Se entiende que además de pasarla a formato *.arff* ya ha aplicado el preprocesamiento necesario en función del fichero “**Pistas sobre los datasets con posible preprocesamiento a simple vista.pdf**”, en el caso de que sea una de las bases de datos que lo requiera.

- Cargue la base de datos con un 75/25 % y ejecute el algoritmo Multilayer-Perceptron con los valores por defecto.
- ¿Qué observa al ir modificando sólo el **TrainingTime**? ¿Cambia el valor de *Correctly Classified instances* al modificar el parámetro? ¿Se estanca el aprendizaje o sobreentrena?
- ¿Qué observa al ir modificando sólo el **LearningRate**? ¿Cambia el valor de *Correctly Classified instances* al modificar el parámetro? ¿Se estanca el aprendizaje o sobreentrena?

Para este ejercicio se utilizará el mismo conjunto de datos preprocesado (fantasmas) que en el ejercicio anterior. La partición del conjunto se realizará al 75/25 % tal como indica el enunciado, por lo que dispondremos de un conjunto de entrenamiento de 278 patrones y un conjunto de generalización de 93 con las mismas frecuencias de clases. El perceptrón tendrá tantos nodos en la capa de entrada como atributos la base de datos y tantos nodos en la capa de salida como valores puede tomar la clase (ver fig. 57). Tanto el número de capas ocultas como el número de nodos en cada una de ellas es configurable, como veremos más adelante.

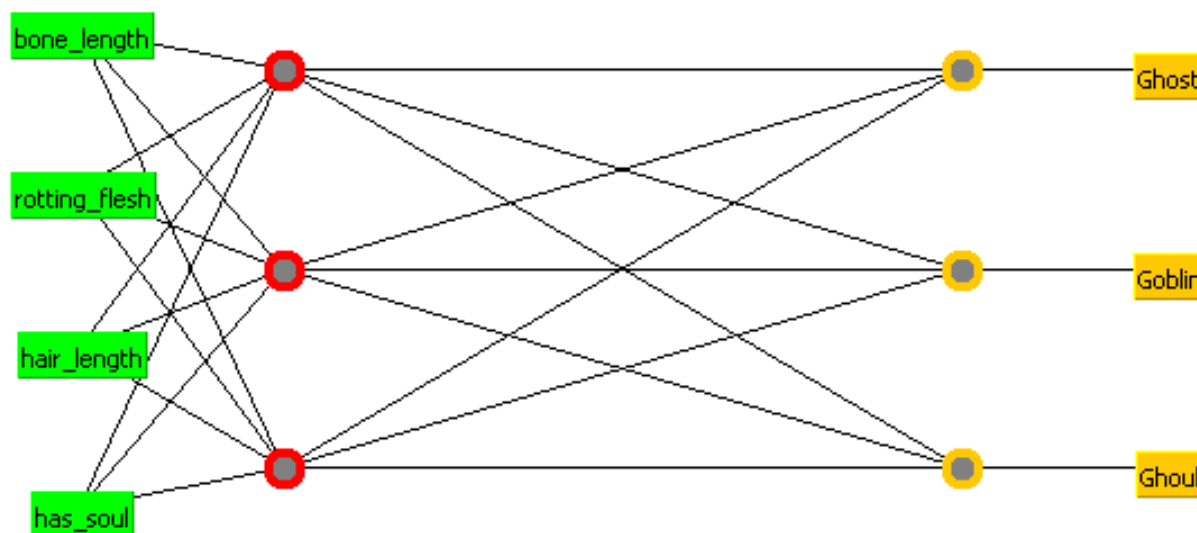


Figura 57: Topología de la red

A continuación repasaremos los parámetros de configuración más importantes y sus valores por defecto<sup>6</sup> en MultilayerPerceptron (fig. 58).

<sup>6</sup>En la captura se ha modificado el valor de GUI a true para poder capturar el diagrama de la red.



- El parámetro `hiddenLayers` establece la configuración topológica de la red, en este caso, una única capa oculta con “a” nodos, donde el comodín “a” toma el valor del punto medio entre el número de atributos independientes y el número de clases. Para nuestro caso,  $(3 + 3/2) = 3$  nodos (ver fig. 57).
- El parámetro `learningRate` o tasa de aprendizaje indica cuánto se modifican los pesos de las aristas durante el aprendizaje y viene a 0,3.
- El parámetro `decay` a `false` indica que el factor de aprendizaje se mantendrá constante a lo largo del aprendizaje.
- El parámetro `momentum` representa una especie de inercia que nos permite salir de un mínimo local, y toma valor por defecto 0,2.
- El parámetro `trainingTime` indica el número de épocas de entrenamiento, donde una época consiste en la evaluación de un patrón del conjunto de entrenamiento y el posterior ajuste de los pesos de la red mediante `back propagation`.
- El parámetro `validationThreshold` indica un número de épocas tal que, si el error de validación sube durante más épocas que las indicadas en el umbral, el aprendizaje se para aunque no se haya alcanzado el `trainingTime`. En el caso por defecto, este umbral se fija a 20 épocas.

The screenshot shows the 'weka.gui.GenericObjectEditor' window for the 'weka.classifiers.functions.MultilayerPerceptron' classifier. The 'About' section describes it as a classifier using backpropagation. The configuration parameters are as follows:

Parameter	Value
GUI	True
autoBuild	True
batchSize	100
debug	False
decay	False
doNotCheckCapabilities	False
hiddenLayers	a
learningRate	0.3
momentum	0.2
nominalToBinaryFilter	True
normalizeAttributes	True
normalizeNumericClass	True
numDecimalPlaces	2
reset	False
resume	False
seed	0
trainingTime	500
validationSetSize	0
validationThreshold	20

Buttons at the bottom: Open..., Save..., OK, Cancel.

Figura 58: Formulario de configuración para MultilayerPerceptron

Al ejecutar el algoritmo con los valores por defecto, observamos unos resultados (ver fig. 59) casi idénticos a los que se obtuvieron con J48 (ver fig. 56).  $CCR = 69,89\%$ , 65 patrones correctamente clasificados frente a 28 incorrectamente clasificados,  $\kappa = 0,5455$ , peor TP Rate para la clase Goblin con un valor de 0,676 y 23 goblins correctamente clasificados de 34. Dos más que con J48, aunque hay dos ghouls mal clasificados más y los mismos ghost, de ahí la coincidencia del CCR.

```

=== Summary ===

Correctly Classified Instances      65           69.8925 %
Incorrectly Classified Instances    28           30.1075 %
Kappa statistic                    0.5455
Mean absolute error                 0.2445
Root mean squared error             0.3755
Relative absolute error             55.0125 %
Root relative squared error         79.594 %
Total Number of Instances          93

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0,724    0,078    0,808    0,724    0,764      0,667    0,939    0,899    Ghost
      0,676    0,271    0,590    0,676    0,630      0,396    0,772    0,600    Goblin
      0,700    0,111    0,750    0,700    0,724      0,600    0,886    0,812    Ghoul
Weighted Avg.   0,699    0,159    0,709    0,699    0,702      0,546    0,861    0,762

=== Confusion Matrix ===

  a  b  c  <-- classified as
21  8  0 | a = Ghost
 4 23  7 | b = Goblin
 1  8 21 | c = Ghoul

```

Figura 59: Resultados de MultilayerPerceptron con configuración por defecto

A continuación vamos a comprobar cómo afectan a los resultados las variaciones en el parámetro `trainingTime`. Para ello vamos a ejecutar el algoritmo variando dicho parámetro y tomaremos como medida de la bondad del resultado el número de patrones bien clasificados. Los datos obtenidos se muestran en el cuadro 9, donde se presenta el número de patrones correctamente clasificados (PCC) en función del valor de training time utilizado.

Training Time	5	10	50	100	200	300	400	500	1000	2000	5000
PCC	67	66	64	64	64	64	65	65	66	65	64

Cuadro 9: Patrones correctamente clasificados en función del training time

Aunque las variaciones son pequeñas dado que se trata de un conjunto pequeño, a la vista de los datos obtenidos, se deduce que existe un mínimo muy cerca de los pesos iniciales, que se alcanza a las pocas iteraciones pero del que sale también muy pronto, presumiblemente debido al momentum. A partir de las 10 épocas, el proceso está aparentemente estancado aunque mejora poco a poco hasta decaer de nuevo a partir de las 2000. Podría decirse que vemos sobre-entrenamiento a partir de la época 10, ya que el resultado empeora aunque habiendo entrenado tan poco ya es una casualidad.

Finalmente realizaremos un estudio similar al anterior pero esta vez manteniendo el valor por defecto de `trainingTime` y variando `learningRate`. De nuevo tomaremos como medida de bondad el número de patrones bien clasificados. Los datos obtenidos se muestran en el cuadro 10.

Si antes cambiábamos el número de pasos, ahora cambiamos el tamaño de éstos, y los datos vienen a confirmar lo que se observó en el experimento anterior, ya que con un `learningRate` extremadamente

Learning Rate	0,05	0,1	0,2	0,3	0,4	0,5	1
PCC	67	66	66	65	64	62	64

Cuadro 10: Patrones correctamente clasificados en función del learning rate

pequeño alcanzamos un mínimo local que debe estar cercano al punto de partida. Según aumenta el learning rate empeora. Se podría decir que a partir de cierto learning-rate, un salto eventualmente nos hace salir del valle de la función de coste. Según aumenta el learning rate, los saltos son tan grandes que el resultado puede tanto mejorar como empeorar de época en época, y el aprendizaje deja de ser dirigido para volverse aleatorio.

## Bibliografía

- [1] “Weka: General documentation.” <https://waikato.github.io/weka-wiki/documentation/>. Accedido: 27/5/2020.
- [2] D. García Morate, “Manual de weka.” <https://knowledgesociety.usal.es/sites/default/files/MANUAL%20WEKA.pdf>. Accedido: 27/5/2020.
- [3] C. L. Corso and S. L. Alfaro, “Alternativa de herramienta libre para la implementación de aprendizaje automático.” [http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/congresos\\_labsis/cynthia/Alternativa\\_de\\_herramienta\\_para\\_Mineria\\_Datos\\_CNEISI\\_2009.pdf](http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/congresos_labsis/cynthia/Alternativa_de_herramienta_para_Mineria_Datos_CNEISI_2009.pdf). Accedido: 27/5/2020.