

Introducción al Aprendizaje Automático

Prácticas

Juan José Méndez Torrero
i42metoj@uco.es
Grado en Ingeniería Informática
Universidad de Córdoba

Base de datos usada: Ecoli

February 2, 2019

Contents

1	Práctica 1: Introducción a Weka	6
1.1	Ejercicio 1	6
1.2	Ejercicio 2	8
1.3	Ejercicio 3	10
1.4	Ejercicio 4	11
1.5	Ejercicio 5	16
1.6	Ejercicio 6	23
1.7	Ejercicio 7	26
2	Práctica 2.1: Clasificación, Regresión y Clustering con Weka	27
2.1	Ejercicio 1	27
2.2	Ejercicio 2	31
2.3	Ejercicio 3	33
2.4	Ejercicio 4	36
3	Práctica 2.2: Agrupamiento o Clustering	39
3.1	Ejercicio 1	39
3.2	Ejercicio 2	45
3.3	Ejercicio 3	47
3.4	Ejercicio 4	47
3.5	Ejercicio 5	49
3.6	Ejercicio 6	53
3.7	Ejercicio 7	54
3.8	Ejercicio 8	57
3.9	Ejercicio 9	59
4	Práctica 3: Árboles de Decisión y Redes Neuronales	60
4.1	Ejercicio 1	60
4.2	Ejercicio 2	75
4.3	Ejercicio 3	78
4.4	Ejercicio 4	79
4.5	Ejercicio 5	80
5	Práctica 4: Preprocesamiento	83
6	Bibliografía:	84

List of Figures

1	Atributos	6
2	Estadísticas	7
3	Plot matrix: petalwidth, sepallenght	8
4	Atributos e instancias	9
5	Atributos e instancias con filtro	9

6	Atributos <i>bone</i> y <i>boneAbnormal</i> después de aplicar el filtro . . .	10
7	<i>nfolds</i> = 5	11
8	Atributos e instancias al aplicar el filtro <i>StratifiedRemoveFolds</i> .	11
9	Configuración del filtro <i>StratifiedRemoveFolds</i>	12
10	Archivo .arff	14
11	Atributos e instancias de Ecoli	14
12	Tabla con los valores de los atributos	14
13	Diagrama de las clases	15
14	Plot matrix de la base de datos Ecoli	15
15	Configuración del filtro Remove	16
16	Atributos Ecoli antes de aplicar el filtro Remove	17
17	Atributos Ecoli después de aplicar el filtro Remove	17
18	Configuración del filtro RemoveUseless	17
19	Configuración del filtro Normalize	18
20	Atributo <i>gvh</i> antes de aplicar el filtro Normalize	18
21	Atributo <i>gvh</i> después de aplicar el filtro Normalize	19
22	Configuración del filtro <i>RemovePercentage</i>	20
23	Instancias de la base de datos Ecoli antes de aplicar el filtro <i>RemovePercentage</i>	20
24	Instancias de la base de datos Ecoli después de aplicar el filtro <i>RemovePercentage</i>	21
25	Configuración del filtro <i>Resample</i>	22
26	Diagrama de las clases después de aplicar el filtro <i>Resample</i> . . .	22
27	Configuración del filtro <i>Discretize</i>	23
28	Atributo <i>aac</i> después de aplicarle el filtro <i>Discretize</i>	24
29	Atributos de la base de datos Ecoli tras aplicar el filtro <i>NominalToBinary</i>	24
30	Configuración del filtro <i>SpreadSubsample</i>	25
31	<i>Currentrelation</i> de la base de datos Ecoli después de aplicar el filtro <i>SpreadSubsample</i>	25
32	Instancias por cada clase	26
33	Diagrama de las clases después de aplicar el filtro <i>ClassBalancer</i>	26
34	Configuración para el clasificador <i>IBK</i>	27
35	Configuración para el tipo de test	28
36	Información sobre la base de datos Ecoli	28
37	Información sobre el algoritmo de clasificación IB1 aplicado sobre la base de datos Ecoli	29
38	Configuración de Experiment Environment	32
39	Métricas para IB2	32
40	Métricas para IB3	33
41	Información general sobre el resultado del algoritmo <i>Logistic</i> . . .	34
42	Tablas con métricas sobre el resultado del algoritmo <i>Logistic</i> . .	34
43	Métricas y matriz de confusión al aplicar el algoritmo <i>Logistic</i> .	35
44	Influencia de cada atributo en cada clase	37
45	Métricas y matriz de confusión al aplicar el algoritmo <i>SimpleL- ogistic</i>	38
46	Información general sobre los dos algoritmos	38

47	Resultado de aplicar el algoritmo <i>SimpleKMeans</i> con $K = 2$. . .	40
48	Resultado de aplicar el algoritmo <i>SimpleKMeans</i> con $K = 3$. . .	41
49	Resultado de aplicar el algoritmo <i>SimpleKMeans</i> con $K = 4$. . .	42
50	Gráfica resultante de aplicar el algoritmo <i>SimpleKMeans</i> con $K = 2$	43
51	Gráfica resultante de aplicar el algoritmo <i>SimpleKMeans</i> con $K = 3$	44
52	Gráfica resultante de aplicar el algoritmo <i>SimpleKMeans</i> con $K = 4$	45
53	Gráfica resultante de aplicar el algoritmo <i>SimpleKMeans</i> ignorando el atributo <i>class</i>	46
54	Gráfica resultante de aplicar el algoritmo <i>SimpleKMeans</i> sin ignorar el atributo <i>class</i>	46
55	Plano inicial	49
56	Primera agrupación	50
57	Primera iteración	50
58	Segunda agrupación	51
59	Segunda iteración	52
60	Segunda iteración	52
61	Tercera iteración	53
62	Tercera agrupación	54
63	Dendograma obtenido usando <i>Simple linkage</i>	55
64	Dendograma obtenido usando <i>Complete linkage</i>	57
65	Gráfia obtenida comparando <i>Instance_number</i> y <i>SepalLength</i> . .	57
66	Gráfia obtenida comparando <i>Instance_number</i> y <i>SepalWidth</i> . .	58
67	Gráfia obtenida comparando <i>Instance_number</i> y <i>PetalLength</i> . .	58
68	Gráfia obtenida comparando <i>Instance_number</i> y <i>PetalWidth</i> . .	58
69	Matriz de confianza obtenida con <i>Epsilo</i> = 0.1 y <i>minPoint</i> = 5 .	59
70	Árbol resultante al aplicar el algortimo C4.5	61
71	Árbol resultante	62
72	Métricas resultantes	63
73	Árbol resultante	64
74	Métricas resultantes	65
75	Árbol resultante	65
76	Métricas resultantes	66
77	Árbol resultante	67
78	Métricas resultantes	67
79	Árbol resultante	68
80	Métricas resultantes	69
81	Árbol resultante	70
82	Métricas resultantes	70
83	Árbol resultante	71
84	Métricas resultantes	72
85	Árbol resultante	73
86	Métricas resultantes	74
87	Árbol resultante	74
88	Métricas resultantes	75
89	Métricas y árbol resultante	76
90	Métricas y árbol resultante	77

91	Árbol resultante	78
92	Métricas obtenidas al aplicar el algoritmo <i>RandomForest</i>	79
93	Diagrama de la red neural multicapa para la base de datos <i>Iris</i> .	80
94	Gráfico comparativo entre CCI y trainingTime	82

1 Práctica 1: Introducción a Weka

1.1 Ejercicio 1

Cargue la base de datos Iris (disponible en Moodle). Observe los atributos.

1. ¿Cuántos atributos caracterizan los datos de esta base de datos?

Una vez cargada la base de datos Iris, dentro de Weka podemos encontrar el número de atributos que contiene esta base de datos dentro del apartado de Current relation, como se puede observar en la Figura 1. Como podemos observar, el número de atributos es 5, aunque en realidad, los atributos que caracterizan dichas instancias son 4, ya que la última es el atributo que define las clases a las que pertenecen cada instancia.

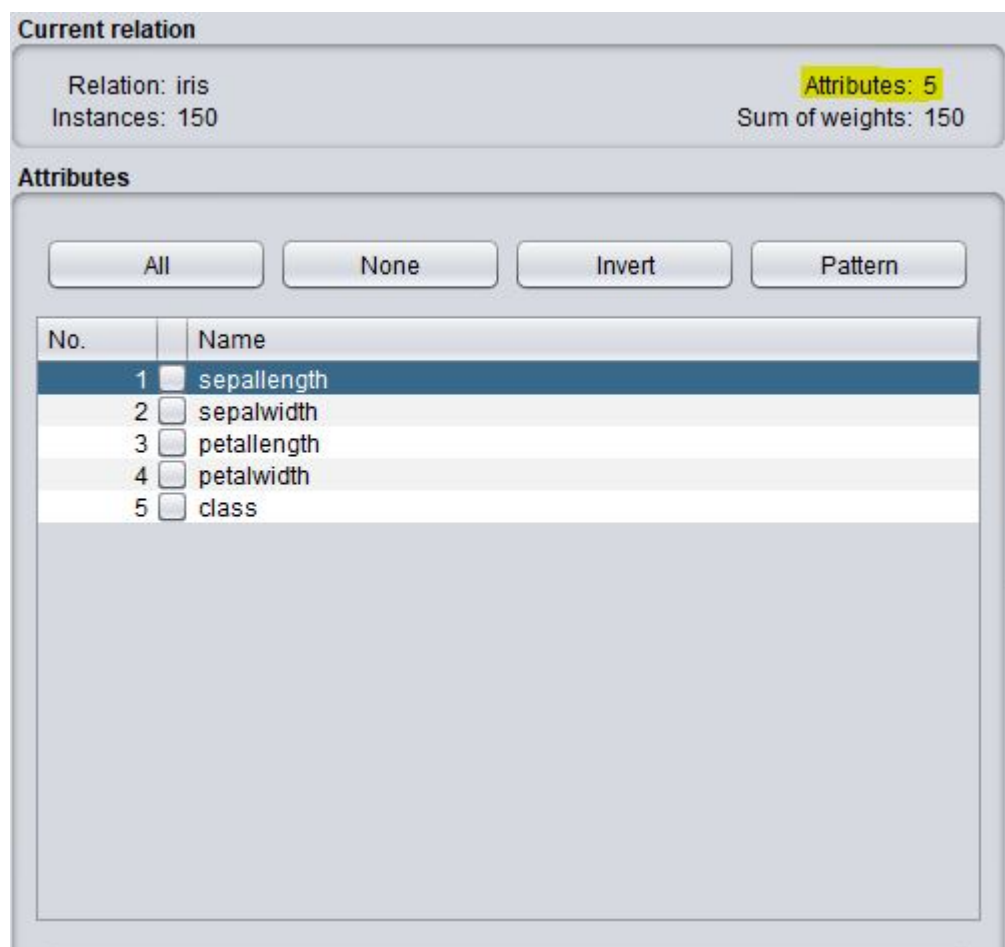


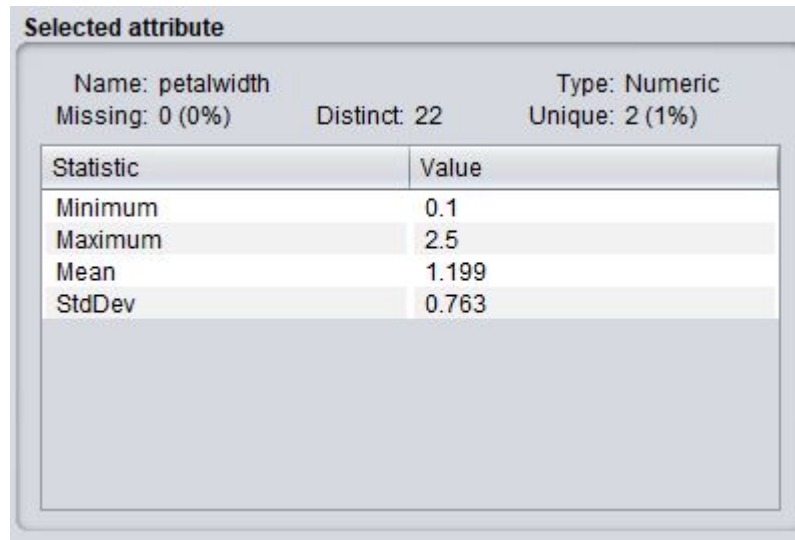
Figure 1: Atributos

2. Si suponemos que queremos predecir el último atributo a partir de los anteriores, ¿se trata de regresión o clasificación?

Como hemos comentado en el ejercicio anterior, el último atributo del dataset es la clase a la que pertenece cada instancia, con lo que estaríamos ante un problema de clasificación.

3. ¿Cuál es el rango de valores del atributo *petalwidth*? ¿Y su media y desviación típica?

Para ver el rango de valores del atributo *petalwidth*, primero hay que seleccionarlo en la sección de *Attributes*. Posteriormente, dichas estadísticas aparecen en la sección *Selected attribute*, como podemos observar en la Figura 2. Viendo esto, podemos decir que dicho atributo toma unos valores comprendidos entre [0.1,2.5]. Su media también podemos encontrarla en dicha sección y tiene un valor de 1.199. Igualmente, su desviación típica toma un valor de 0.763.



The screenshot shows a window titled "Selected attribute" with a light blue border. Inside, the attribute "petalwidth" is selected. It shows "Missing: 0 (0%)", "Distinct: 22", and "Type: Numeric". Below this, a table lists statistics: Minimum (0.1), Maximum (2.5), Mean (1.199), and StdDev (0.763).

Selected attribute		
Name: petalwidth		Type: Numeric
Missing: 0 (0%)		Distinct: 22
		Unique: 2 (1%)
Statistic	Value	
Minimum	0.1	
Maximum	2.5	
Mean	1.199	
StdDev	0.763	

Figure 2: Estadísticas

4. ¿Que diferencia hay entre instancias *Distinct* y *Unique*? **Fabríquese una base de datos pequeña y para poner un ejemplo.**

Las instancias *Distinct* y *Unique* podemos divisarlas cuando hemos seleccionado un atributo de nuestra base de datos. El término *Distinct* hace referencia al número de los diferentes valores que puede tomar dichas instancias dentro de una base de datos, mientras que el término *Unique* hace referencia al número de elementos que sólo se encuentran una vez en la base de datos.

Para entenderlo mejor, imaginemos que tenemos una base de datos con las siguientes instancias:

{PERRO,PERRO,PERRO,GATO,AVE,AVE,CERDO,JIRAFa,JIRAFa}
 En este caso, la instancia *Distinct* tomaría el valor de 5, mientras que el valor de la instancia *Unique*, tendría el valor de 2, ya que {GATO,CERDO} sólo aparecen una vez (son únicos) en nuestra base de datos.

5. **Utilizando el entorno Weka Explorer → Visualize, determinar que atributo permite discriminar linealmente entre la clase iris-setosa y las otras dos clases.**

Utilizando el entorno *Visualize* de Weka, podemos observar que las variables que mejor discriminan las clases son *sepalwidth* y *sepalength*. Esto es así ya que podemos realizar una separación lineal entre la clase *iris-setosa* y las otras dos.

6. **¿Es posible separar linealmente la clase iris-versicolor de la clase iris-virginica?**

Como hemos visto en el entorno *Visualize*, estas dos clases no pueden ser separadas linealmente, esto es debido a que sus patrones están mezclados entre sí.

7. **¿Con qué dos atributos te quedarías para discriminar entre las tres clases del problema?**

En la Figura ?? podemos observar que los dos atributos que mejor discriminan las tres clases son *petalwidth* y *sepalength*. Esto se puede observar de mejor manera en la Figura 3.

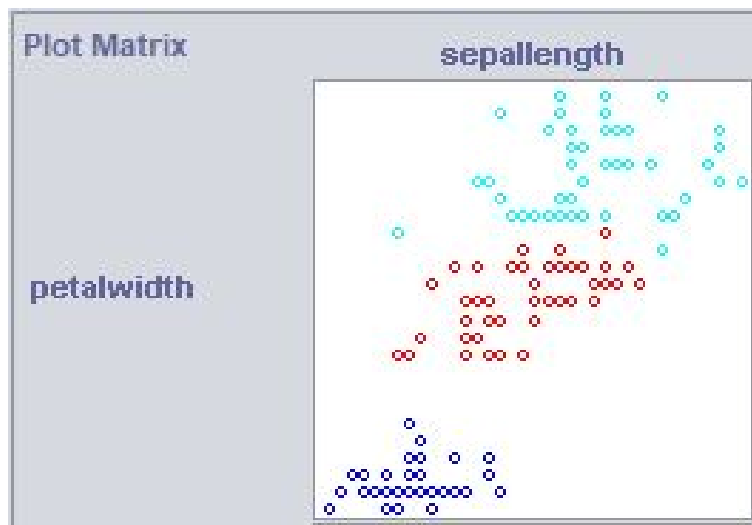


Figure 3: Plot matrix: petalwidth, sepalength

1.2 Ejercicio 2

Cargue la base de datos audiology (disponible en Moodle)

1. **Aplique el filtro `filters/unsupervised/attribute/NominalToBinary` y describa como quedan ahora los atributos.**

Cuando cargamos la base de datos, podemos observar en la sección *Current relation* que nuestra base de datos consta de 11 atributos y de 226 instancias, todos ellos de tipo Nominal. Podemos ver esto en la Figura 4. El filtro que vamos a aplicar a esta base de datos es *NominalToBinary*,



Figure 4: Atributos e instancias

que, como su nombre indica, convierte las variables nominales a variables binarias. Una vez aplicado, como la Figura 5 indica, los atributos se han duplicado. Esto es debido a que anteriormente, atributos que podían tomar más de un valor, ahora dichos valores son atributos independientes que pueden tomar los valores 0 o 1.



Figure 5: Atributos e instancias con filtro

2. **¿Podría saber con antelación el número de atributos finales al aplicar este filtro?**

Sí se podría saber con antelación. Para ello, habría que ir atributo por atributo viendo qué número de valores puede tomar. Si el atributo puede tomar más de 2 valores, al aplicar el filtro, cada valor será contado como un atributo independiente. Pero en el caso de que sólo pudiera tomar dos valores, como en el caso del atributo *boneAbnormal*, al aplicarle el filtro sólo quedaría un atributo, siendo lo mismo que no aplicando el filtro. Un ejemplo de de un atributo con más valores sería *bone*, que cada valor, al aplicar el filtro, se convertiría en un atributo independiente. Esto podemos observarlo en la Figura 6.

14	<input type="checkbox"/>	bone=mild
15	<input type="checkbox"/>	bone=moderate
16	<input type="checkbox"/>	bone=normal
17	<input type="checkbox"/>	bone=unmeasured
18	<input type="checkbox"/>	boneAbnormal=t

Figure 6: Atributos *bone* y *boneAbnormal* después de aplicar el filtro

1.3 Ejercicio 3

Particione la base de datos Iris usando `filters/ supervised/instance/StratifiedRemoveFolds`

1. **Divida el dataset en train y test mediante un `numFolds=5`. Describa el proceso realizado.**

Cuando aplicamos el filtro *StratifiedRemoveFolds* lo que estamos haciendo es dividir la base de datos de manera estratificada. Esto significa que el filtro va a dividir el conjunto de datos en *folds*, de una manera proporcionada. Estos *folds* van a ser usados, unos para el entrenamiento de los datos y otros para su generalización. Todo esto lo hacemos para poder aplicar la validación cruzada. En cada iteración se va a seleccionar unos *folds* para la validación. Seguidamente, a la hora de aplicar el filtro, debemos indicar el número de *folds* que vamos a utilizar, esto es representado en la Figura 7.

Como podemos observar en la Figura 8, una vez aplicado la validación cruzada con *nfolds=5*, el número de instancias ha sido reducido de 150 a 30. Esto es debido por lo explicado anteriormente, el conjunto de datos ha sido dividido en 5 *fold* iguales, conteniendo cada *fold* 30 instancias.

2. **Divida el dataset en train y test mediante un `holdOut=5` con un 75% train y 25% test. Describa el proceso realizado.**

Una vez cargada la base de datos, lo que tenemos que hacer es cambiar los parámetros del filtro *StratifiedRemoveFolds*.

Para comenzar, lo que tenemos que hacer es declarar la variable *numFolds = 4*, para poder así sacar 4 de los 5 *holdOut* que son pedidos. Seguidamente, para conseguir el primer *fold*, debemos declarar la variable *fold=1*, obteniendo así la primera división entre el conjunto de datos *Train* y *Test*. Finalmente, para conseguir la división del 75%, 25%, tenemos que modificar la variable *invertSelection*, que, declarandola *False*, obtenemos el 25% de las instancias para *Test*. Inversamente, si la declaramos como *True*, obtendremos el 75% restante para *Train*.

Para conseguir el último *holdOut*, debemos declarar la variable *seed=1*, obteniendo así los 5 *holdOut*. Para ver la configuración final, debemos fijarnos en la Figura 9.

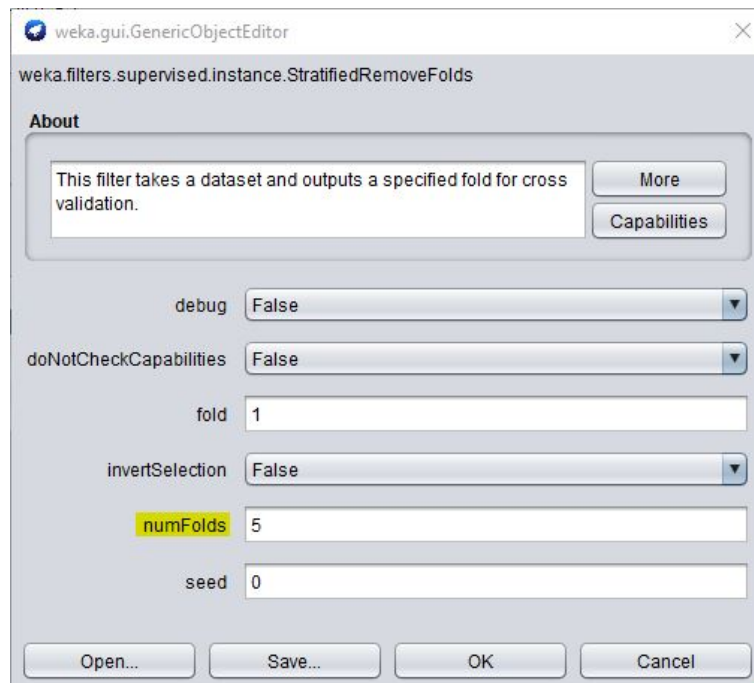


Figure 7: $\text{nfolds} = 5$

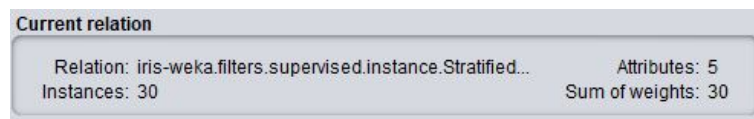


Figure 8: Atributos e instancias al aplicar el filtro *StratifiedRemoveFolds*

1.4 Ejercicio 4

Ecoli dataset como base de datos para los guiones (disponible en Moodle)

1. Consulte el UCI Machine Learning Repository y haga una descripción completa y detallada de la base de datos.

Una vez que hemos entrado en la página del UCI, podemos encontrar el conjunto de datos Ecoli. En ella se muestra una gran variedad de información para poder entender la base de datos antes de procesarla.

Con respecto al conjunto de datos Ecoli, contiene la información sobre un sistema de clasificación probabilístico para la predicción de la localización celular de las proteínas. Su creador y la persona que lo mantiene es Kenta Nakai, del instituto molecular y biología celular en Osaka, Japón. Los datos que podemos encontrar en ella son de Paul horton, y dichos datos

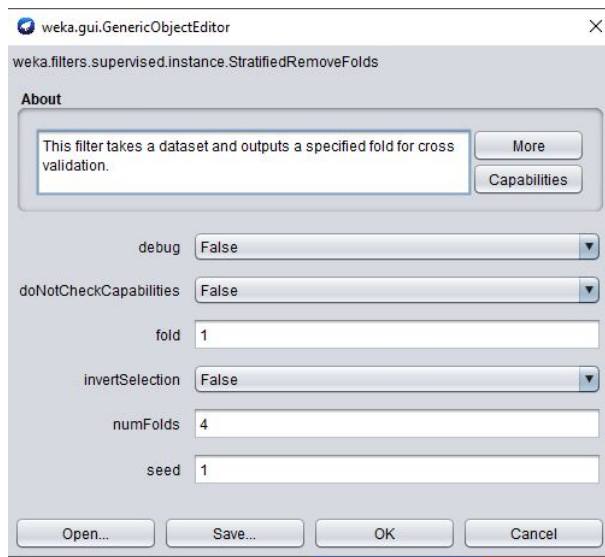


Figure 9: Configuración del filtro *StratifiedRemoveFolds*

fueron donados en Septiembre del año 1996.

La base de datos contiene 8 atributos, 7 predictivos y 1 nominal. Además, esta base de datos contiene 336 instancias.

La distribución de las instancias es la siguiente:

- (a) cp: 143 pertenecen a la clase citoplasma.
- (b) im: 77 pertenecen a la clase de membrnas internas sin secuencia de señal.
- (c) pp: 52 pertenecen a la clase periplasmas.
- (d) imU: 35 pertenecen a la clase de la membrana interna con secuencia indivisible de señal.
- (e) om: 20 pertenecen a la clase de la membrana exterior.
- (f) omL: 5 pertenecen a la clase de la membrana exterior de lipoproteínas.
- (g) imL: 2 pertenecen a la clase de la membrana interna de lipoproteínas.
- (h) imS: 2 pertencen a la clase de la membrana interior con secuencias divisibles de señal.

Con respecto a los 8 atributos que podemos encontrar son:

- (a) Sequence name: Número de acceso para la base de datos SWISS-PROT.
- (b) mcg: Método de McGeoch para el reconocimiento de secuencias de las señales.

- (c) gvh: Método de Von Heijne para el reconocimiento de secuencias de las señales.
 - (d) lip: Puntuación de la secuencia consenso de Signal Peptidase II de von Heijne. Es un atributo binario
 - (e) chg: Presencia de carga en N-terminos en lipoproteínas pronosticadas. Es un atributo binario.
 - (f) aac: Puntuación del análisis discriminante del contenido de aminoácidos en la membrana exterior y proteínas preiplasmáticas.
 - (g) aml1: Puntuación del programa de predicción de la región de membrana ALOM.
 - (h) alm2: Puntuación del programa ALOM después de excluir las supuestas regiones de señal escindibles de la secuencia.
2. **Construya a partir de la información que haya en la UCI Machine Learning Repository un fichero .arff para Weka. Ponga nombres de atributos descriptivos y use las herramientas que considere necesarias.**

Para la construcción del archivo *.arff*, lo que deberíamos de hacer primero es copiar dicha base de datos en un editor de texto. Seguidamente, dicho contenido lo debemos de ir transformando en el deseado fichero *.arff*. Una vez copiado, como en la Figura 10 se indica, debemos de declarar el nombre de la base de datos junto con los atributos. Para la declaración de los datos, hemos eliminado el atributo *Sequence name*, ya que no es significativo.

3. **Abra el entorno Weka Explorer → Preprocess de Weka, cargue la base de datos y describa de forma concienzuda tanto los atributos como las clases.**

Una vez creado el archivo *.arff*, al cargarlo en Weka Explorer si tiene algún error, el programa nos avisará de dónde se encuentra dicho error. En el caso de cargarse correctamente quiere decir que hemos creado el archivo bien. Para saber si el contenido es correcto, tenemos que fijarnos en la sección *Current relation*, como la Figura 11 indica, la base de datos contiene 336 instancias y 8 atributos, contando con la eliminación del *Sequence name* y la adición del atributo *Class*.

Para ver qué valores tiene cada atributo, vamos a realizar una tabla que contiene tanto el rango, la media, la desviación típica, la variable *Distinct* y la variable *Unique*. La Figura 12 representa dicha tabla.

En cuanto a la distribución de las clases, en la Figura 13 podemos observar la cantidad de datos que contiene cada clase. Siendo las clases en el siguiente order: cp, im, pp, imU, om, omL, imL y imS.

```

relation ecoli

@attribute mcg numeric
@attribute gvh numeric
@attribute lip numeric
@attribute chg numeric
@attribute aac numeric
@attribute alm1 numeric
@attribute alm2 numeric
@attribute class {cp,im,pp,imU,om,omL,imL,imS}

@data
0.49 0.29 0.48 0.50 0.56 0.24 0.35 cp
0.07 0.40 0.48 0.50 0.54 0.35 0.44 cp
0.56 0.40 0.48 0.50 0.49 0.37 0.46 cp
0.59 0.49 0.48 0.50 0.52 0.45 0.36 cp
0.23 0.32 0.48 0.50 0.55 0.25 0.35 cp
0.67 0.39 0.48 0.50 0.36 0.38 0.46 cp
0.29 0.28 0.48 0.50 0.44 0.23 0.34 cp
0.21 0.34 0.48 0.50 0.51 0.28 0.39 cp

```

Figure 10: Archivo .arff

Current relation	
Relation: ecoli	Attributes: 8
Instances: 336	Sum of weights: 336

Figure 11: Atributos e instancias de Ecoli

ATRIBUTO	RANGO	MEDIA	DEV. TÍPICA	DISTINCT	UNIQUE
mcg	[0-0'89]	0'5	195	78	12
gvh	[0'16-1]	0'5	148	63	12
lip	[0'45-1]	0'495	0'088	2	0
chg	[0'5-1]	0'501	0'027	2	1
aac	[0-0'88]	0'5	0'122	59	15
alm1	[0'03-1]	0'5	0'216	82	17
alm2	[0-0'99]	0'5	0'209	77	17

Figure 12: Tabla con los valores de los atributos

4. Observe si hay atributos identificadores. Si no existen diga por qué.

Como sabemos, un atributo indentificador es aquel atributo que sólo define a un individuo. Sabiendo esto, en nuestra base de datos podemos encontrar que el atributo *Sequencename*, el que hemos eliminado, es un atributo

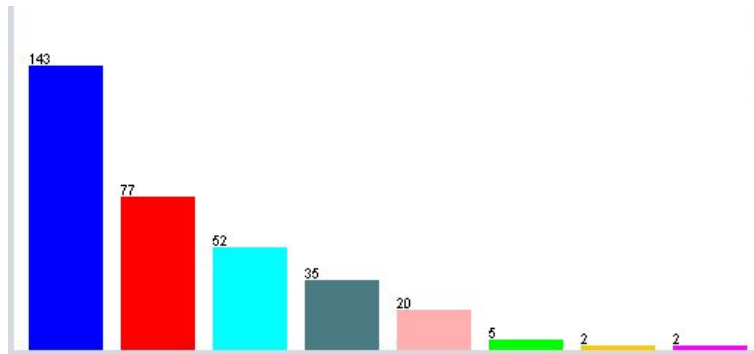


Figure 13: Diagrama de las clases

identificador, ya que no aportaba ninguna información útil para nuestro análisis. A diferencia de éste, los demás atributos son descriptivos.

5. Use el entorno Visualice, ¿Hay alguna relación que sea visualmente significativa?

Como podemos observar en la Figura 14, no existe ninguna relación significativa que podamos ver a simple vista.

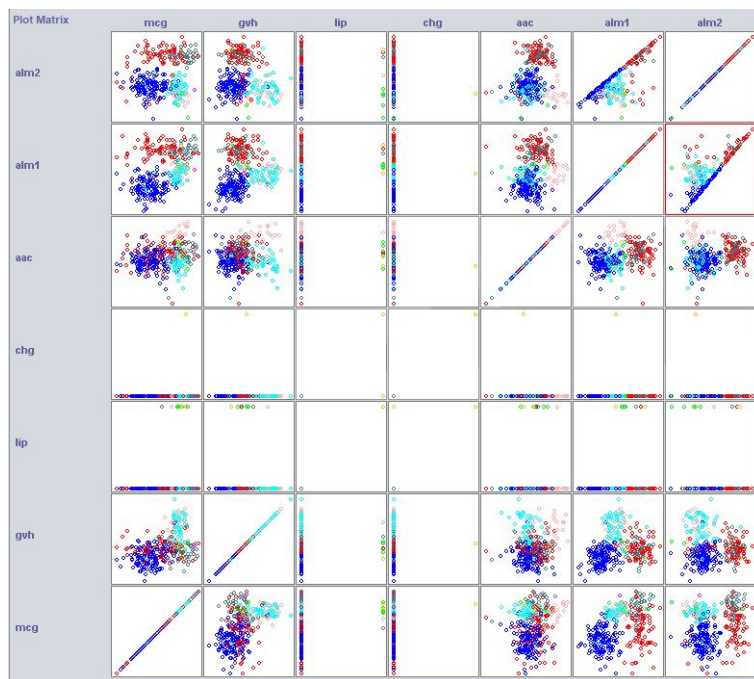


Figure 14: Plot matrix de la base de datos Ecoli

1.5 Ejercicio 5

Describe con sus propias palabras los siguientes filtros No Supervisados y acto seguido describe cómo quedan los datos al aplicar los filtros sobre la base de datos de sus prácticas.

1. filters/unsupervised/attribute/Remove

Como ya sabemos, los filtros no supervisados no tienen en cuenta el último atributo del conjunto de datos, ya que lo toma como una clase o valor numérico de salida para poder así usarlo en regresión. El filtro *Remove* consiste en eliminar un rango de atributos que han sido previamente seleccionados. Este rango se puede seleccionar de varias maneras, pero como podemos ver en la Figura 15, hemos decidido eliminar desde el primer al tercer atributo.

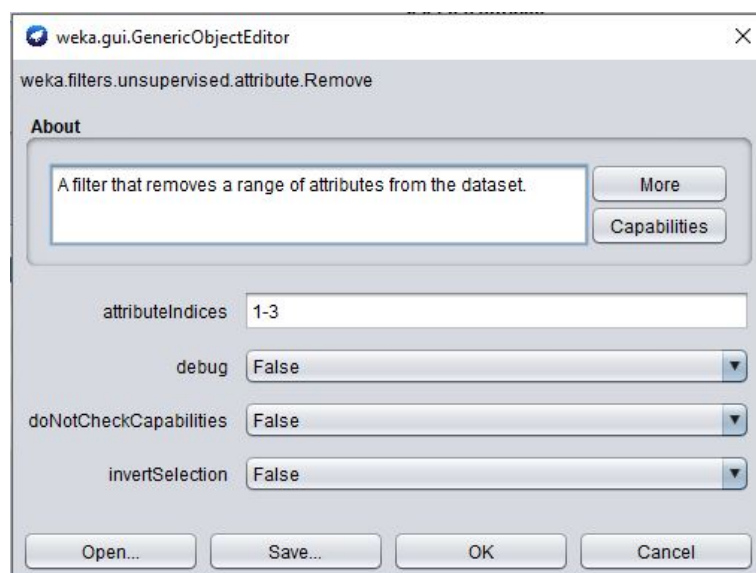


Figure 15: Configuración del filtro Remove

Esto nos debería de eliminar tres de los ocho atributos de los que consta nuestra base de datos, y, como podemos observar en las Figuras 16 y 17, esto ha sucedido correctamente. La Figura 16 representa los atributos antes de aplicar el filtro *Remove*, en cambio, la Figura 17 muestra el resultado al aplicar el filtro.

Como nota hay que añadir que también podemos eliminar sólo un atributo, no siendo siempre obligatorio declarar un rango de atributos.

Este filtro no tiene mucho sentido aplicarlo en la base de datos Ecoli, ya que todos los atributos son significativos.

2. filters/unsupervised/attributes/RemoveUseless

Este filtro, como su nombre indica, elimina atributos cuyos valores son

No.	Name
1	<input checked="" type="checkbox"/> mcg
2	<input type="checkbox"/> gvh
3	<input type="checkbox"/> lip
4	<input type="checkbox"/> chg
5	<input type="checkbox"/> aac
6	<input type="checkbox"/> alm1
7	<input type="checkbox"/> alm2
8	<input type="checkbox"/> class

Figure 16: Atributos Ecoli antes de aplicar el filtro Remove

No.	Name
1	<input checked="" type="checkbox"/> chg
2	<input type="checkbox"/> aac
3	<input type="checkbox"/> alm1
4	<input type="checkbox"/> alm2
5	<input type="checkbox"/> class

Figure 17: Atributos Ecoli después de aplicar el filtro Remove

inútiles, esto quiere decir que dichos atributos o varían muy poco, o varían demasiado. Esta cantidad de variación viene dada por el parámetro `maximumVariancePercentageAllowed`, como podemos observar en la Figura 18. Este parámetro, declara la máxima varianza que pueden tener los atributos para considerarlos útiles.

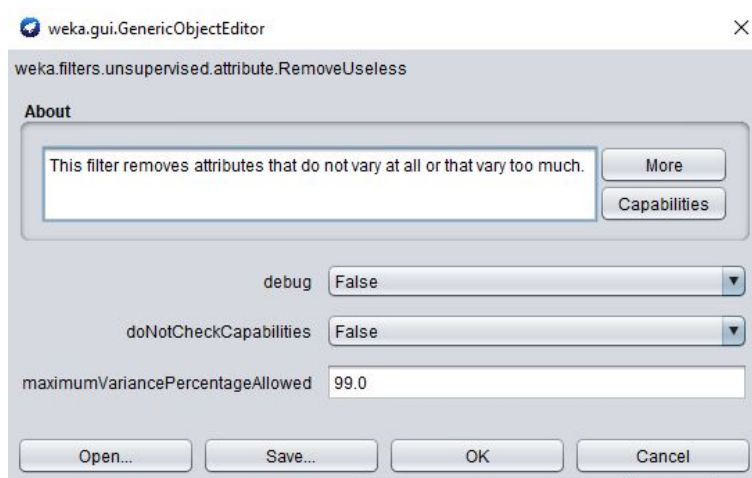


Figure 18: Configuración del filtro RemoveUseless

Este filtro no podemos aplicarlo a nuestra base de datos. Esto se debe a que sólo es aplicable a los conjunto de datos nominales, siendo no este

nuestro caso, ya que la conjunto de datos de Ecoli son numéricos.

3. filters/unsupervised/attribute/Normalize

Este filtro se encarga de normalizar el conjunto de datos a un rango que nosotros podemos elegir. Como podemos observar en la Figura 19, las variables *scale* y *translation* son las que se encargan de declarar el rango en el que queremos normalizar. Normalmente, para el filtro *Normalize* se suele utilizar el rango $[0,1]$.

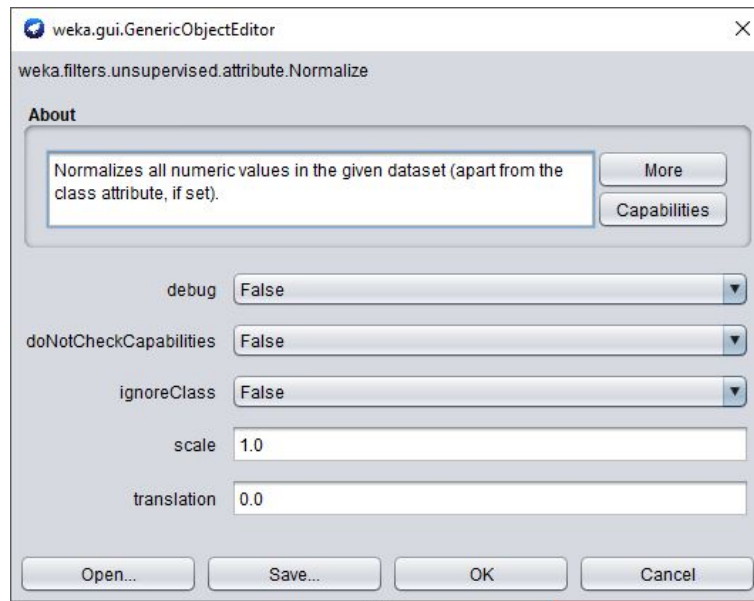


Figure 19: Configuración del filtro Normalize

Aplicando este filtro con el rango $[0,1]$, podemos observa que las variables que tomaban distintos valores de 0 y 1, ahora han sido normalizadas. Un ejemplo sería la variable *gvh*, como podemos observar en las Figuras 20 y 21.

Name: gvh		Type: Numeric
Missing: 0 (0%)		Unique: 12 (4%)
Distinct: 63		
Statistic	Value	
Minimum	0.16	
Maximum	1	
Mean	0.5	
StdDev	0.148	

Figure 20: Atributo *gvh* antes de aplicar el filtro Normalize

Name: gvh		Type: Numeric
Missing: 0 (0%)		Unique: 12 (4%)
Distinct: 63		
Statistic	Value	
Minimum	0	
Maximum	1	
Mean	0.405	
StdDev	0.176	

Figure 21: Atributo *gvh* después de aplicar el filtro Normalize

Este filtro no nos conviene, ya que todos los atributos ya están normalizados en el rango $[0,1]$. Además, hay que decir que este filtro mejora el análisis de nuestra base de datos, si esta no hubiera estado normalizada, ya que todos los atributos trabajarían en un rango similar.

4. filters/unsupervised/attribute/ReplaceMissingValues

A la hora de utilizar este filtro, tenemos que saber si dentro de nuestra base de datos existen atributos valores 0 o corruptos. Si ese es el caso, podríamos aplicar el filtro *ReplaceMissingValues*, el cual se encarga de solventar este problema transformando esos valores en unos que se puedan computar. Un ejemplo podría ser sustituir dichos valores corruptos por la media de esos valores.

En el caso de la base de datos Ecoli, no existe ningún atributo corrupto o con valor 0, con lo que sería inútil aplicar este filtro.

5. filters/unsupervised/attributes/NominalToBinary

Este filtro, como su nombre indica, y como hemos explicado en la pregunta 1.2.1, transforma los atributos nominales en atributos binarios. En nuestro caso, con el conjunto de datos de la base Ecoli, no podemos aplicarlo, ya que en esta base de datos los atributos son de tipo numérico, no nominal. Un ejemplo de este filtro ha sido explicado en el ejercicio 1.2.1.

6. filters/unsupervised/instance/RemovePercentage

A diferencia de los filtros aplicados hasta ahora, el filtro *RemovePercentage* se aplica a las instancias, no a los atributos. Este filtro se encarga de eliminar un porcentaje de instancias definido previamente, como podemos observar en la Figura 22. Dicho filtro es utilizado en bases de datos con un número muy elevado de instancias, que, al eliminar cierto porcentaje, hace el análisis de esa base de datos mucho más simple.

El resultado de este filtro lo hemos analizado con el atributo *Class*, el cual muestra el número de instancias pertenecientes a cada clase. En la Figura 23 podemos observar las instancias antes de aplicar este filtro. En la Figura 24 observamos el resultado después de aplicarlo. Como podemos observar, las instancias se han reducido a la mitad, eliminando por completos todas las instancias pertenecientes a la clase *cp*. Esto es debido a que nuestra

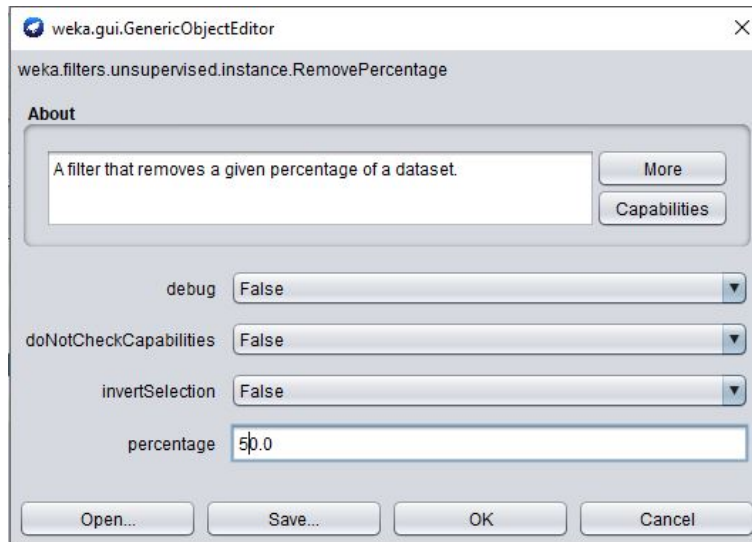


Figure 22: Configuración del filtro *RemovePercentage*

base de datos está ordenada, y que este filtro no elimina las instancias de manera proporcional, sino que sigue un orden. Por esta razón, han desaparecido todas las instancias de esa clase.

Name: class		Type: Nominal	
Missing: 0 (0%)		Unique: 0 (0%)	
		Distinct: 8	
No.	Label	Count	Weight
1	cp	143	143.0
2	im	77	77.0
3	pp	52	52.0
4	imU	35	35.0
5	om	20	20.0
6	omL	5	5.0
7	imL	2	2.0
8	imS	2	2.0

Figure 23: Instancias de la base de datos Ecoli antes de aplicar el filtro *RemovePercentage*

Name: class		Type: Nominal	
Missing: 0 (0%)		Distinct: 7	
		Unique: 0 (0%)	
No.	Label	Count	Weight
1	cp	0	0.0
2	im	52	52.0
3	pp	52	52.0
4	imU	35	35.0
5	om	20	20.0
6	omL	5	5.0
7	imL	2	2.0
8	imS	2	2.0

Figure 24: Instancias de la base de datos Ecoli después de aplicar el filtro *RemovePorcentaje*

7. filters/unsupervised/instance/RemoveDuplicates

Como en el apartado anterior, es un filtro que es aplicado a las instancias y no a los atributos. Este filtro se encarga de eliminar datos repetidos o redundantes de las bases de datos. Al aplicar el filtro a nuestra base de datos Ecoli podemos observar que el número de instancias no varía, significando que no tiene datos redundantes. Para ver si esto es cierto, bastaría con copiar un dato de nuestra base de datos y copiarlo dentro de ella. Esto aumenta el número de instancias en N , siendo N el número de veces que hemos copiado dicho dato. Una vez hecho esto, y aplicando el filtro *RemoveDuplicates*, podemos observar cómo el número de instancias vuelve a ser el mismo que al principio.

8. filters/unsupervised/instance/Resample

Este filtro se encarga de generar un nuevo conjunto de datos en el que se intenta balancear la cantidad de instancias que pertenecen a cada clase. Para ello, hay que tener en cuenta dos factores, la semilla a utilizar que genera nuestra muestra aleatoria y el porcentaje del conjunto de datos original que va a generar nuestro nuevo conjunto de datos. En la Figura 25 podemos ver la configuración de este filtro y los comentados factores importantes.

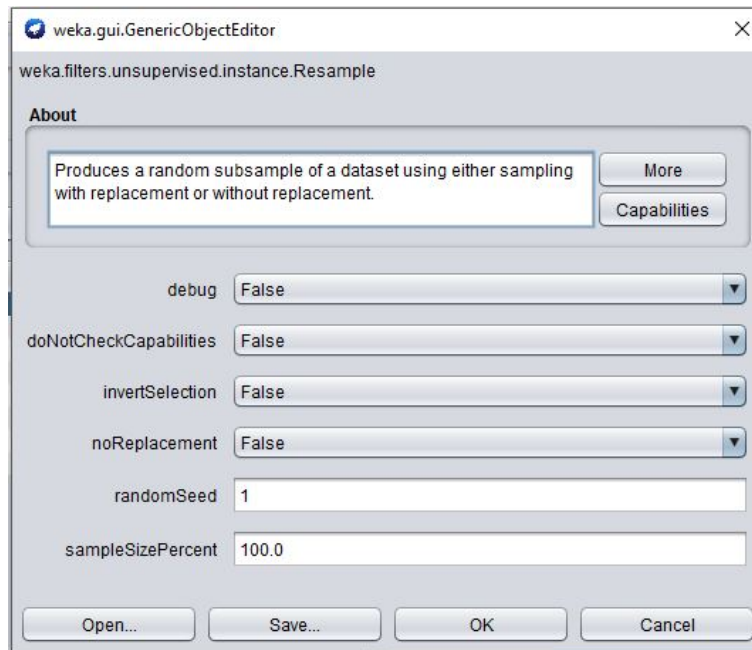


Figure 25: Configuración del filtro *Resample*

Al aplicarlo, podemos observar que clases como cp, que antes tenía 143 instancias, ahora tiene 142. Las diferencias pueden ser observadas entre las Figuras 13 y 26.

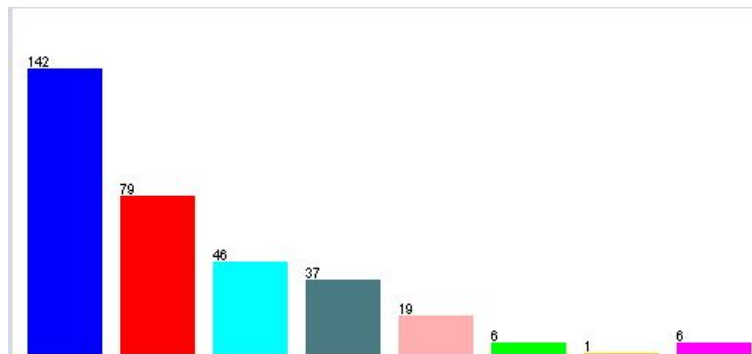


Figure 26: Diagrama de las clases después de aplicar el filtro *Resample*

1.6 Ejercicio 6

Describa con sus propias palabras los siguientes filtros Supervisados y acto seguido describa cómo quedan los datos al aplicar los filtros sobre la base de datos de sus prácticas

1. filters/supervised/attribute/Discretize

A diferencia de los filtros comentados hasta ahora, el filtro *Discretize* es un filtro supervisado, es decir, que a diferencia de los no supervisados, este necesita del último atributo del conjunto de datos para poder tratarlos. Este filtro se encarga de dividir las instancias de las clases en rangos definidos previamente. Estas divisiones clarifican qué número de instancias de las clases pertenecen a un valor u a otro, facilitando así el análisis de los datos. La configuración de este filtro puede ser observada en la Figura 27, como se puede observar, podemos declarar la precisión con la que se va a trabajar, que por defecto es 6. Además, con este filtro también podemos convertir los valores nominales en binarios, declarando la variable *makeBinary* a estado *True*.



Figure 27: Configuración del filtro *Discretize*

Como podemos observar en la Figura 28 la variable *aac* ahora está dividida en 3 rangos diferentes: $[-inf, 0.565]$, $(0.565, 0.715]$, $(0.715, inf)$, teniendo cada uno una cantidad diferente de instancias. Con esto sabemos que 246 instancias pertenecen al primer rango, 73 al segundo y finalmente 17 al tercero. Esto hace, como hemos comentado anteriormente, el análisis de estos datos más claro.

Name: aac		Type: Nominal	
Missing: 0 (0%)		Unique: 0 (0%)	
Distinct: 3			
No.	Label	Count	Weight
1	'(-inf-0.565]'	246	246.0
2	'(0.565-0.715]'	73	73.0
3	'(0.715-inf]'	17	17.0

Figure 28: Atributo *aac* después de aplicarle el filtro *Discretize*

2. filters/supervised/attribute/NominalToBinary

Para la aplicación de este filtro, vamos a dejar aplicado el filtro anterior, *Discretize*, ya que, no como en la pregunta 1.5.5, los atributos ahora son de tipo nominal, lo que hace que podamos aplicar este filtro. En definitiva, como hemos explicado en los anteriores apartados, este filtro convierte los atributos nominales en binarios. El resultado de este filtro lo podemos observar en la Figura 29.

No.	Name
1	<input type="checkbox"/> mcg='(-inf-0.555]'
2	<input type="checkbox"/> mcg='(0.555-0.755]'
3	<input type="checkbox"/> mcg='(0.755-inf]'
4	<input type="checkbox"/> gvh='(0.565-inf]'
5	<input type="checkbox"/> lip='(0.74-inf]'
6	<input type="checkbox"/> chg
7	<input type="checkbox"/> aac='(-inf-0.565]'
8	<input type="checkbox"/> aac='(0.565-0.715]'
9	<input type="checkbox"/> aac='(0.715-inf]'
10	<input type="checkbox"/> alm1='(-inf-0.355]'
11	<input type="checkbox"/> alm1='(0.355-0.575]'
12	<input type="checkbox"/> alm1='(0.575-inf]'
13	<input type="checkbox"/> alm2='(0.615-inf]'
14	<input type="checkbox"/> class

Figure 29: Atributos de la base de datos Ecoli tras aplicar el filtro *NominalToBinary*

Como podemos observar, este filtro no es adecuado ya que dificulta el entendimiento de los atributos, con lo que no vamos a aplicarlo de aquí en adelante.

3. filters/supervised/instance/SpreadSubsample

Este filtro trabaja de manera similar al filtro *Resample*, ya que divide las instancias en rangos. La diferencia de este filtro con el anterior es que, ahora, podemos elegir la diferencia de número de instancias entre la

clase que más instancias tiene y la que menor número de instancias tiene. Para ello, en la configuración de este filtro debemos declarar la variable $spread = 1$. Esto hace que solamente haya una instancia por cada valor de las clases. Equilibrando así el conjunto de datos. La configuración de este filtro podemos verla en la Figura 30.

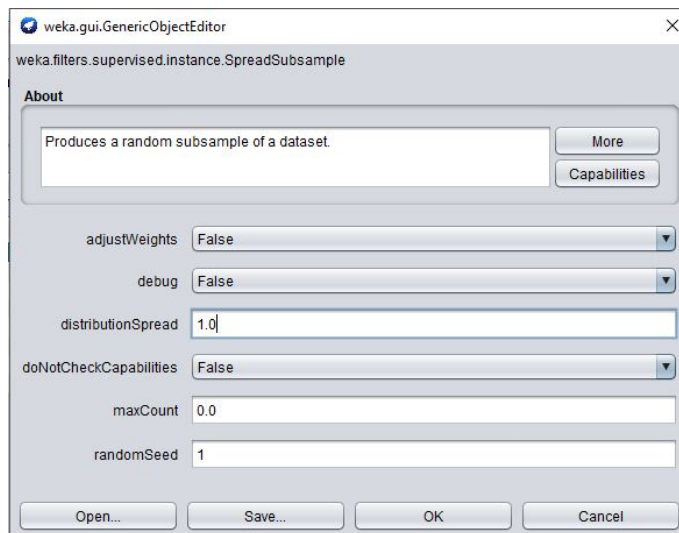


Figure 30: Configuración del filtro *SpreadSubsample*

Como podemos observar en la Figura 31, el número de instancias se ha reducido a 16. Esto quiere decir que se ha cumplido lo anteriormente descrito, una instancia por cada valor que puede tomar la clase.

Relation: ecoli-weka.filters.supervised.attribute.Discretiz...	Attributes: 8
Instances: 16	Sum of weights: 16

Figure 31: *Currentrelation* de la base de datos Ecoli después de aplicar el filtro *SpreadSubsample*

4. filters/supervised/instance/ClassBalancer

Con este filtro, como su nombre indica, conseguimos crear un conjunto de datos totalmente equilibrado, es decir, el mismo número de instancias por cada clase. En la Figura 32 podemos observar que el número de instancias de cada clase ahora es 42. Igualmente se puede ver en la Figura 33, que comparandola con la Figura 13, todas las clases tienen el mismo número de instancias.

Este filtro sería adecuado para este tipo de base de datos, ya que existe mucha diferencia entre el número de instancias por clase. Un ejemplo sería que la clase *cp* que antes tenía 143 instancias, ahora tiene 42, por

Name: class Missing: 0 (0%)		Distinct: 8	Type: Nominal Unique: 0 (0%)
No.	Label	Count	Weight
1	cp	143	42.0
2	im	77	42.0
3	pp	52	42.0
4	imU	35	42.0
5	om	20	42.0
6	omL	5	42.0
7	imL	2	42.0
8	imS	2	42.0

Figure 32: Instancias por cada clase

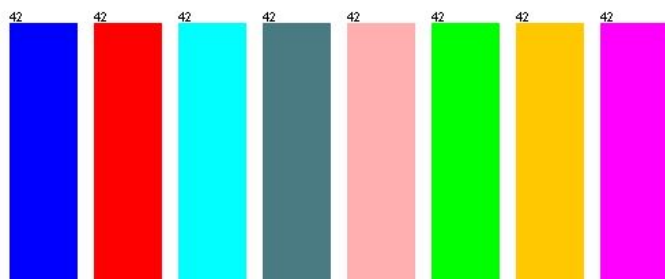


Figure 33: Diagrama de las clases después de aplicar el filtro *ClassBalancer*

otro lado, la clase *imL* que antes tenía sólo 2 instancias, ahora tiene 42. Como vemos, con este filtro hemos conseguido equilibrar la base de datos.

5. **filters/supervised/instance/Resample** Como hemos visto en el ejercicio 1.5, el filtro *Resample* crea un nuevo conjunto de datos aleatorio. La diferencia con la anterior, es que, al ser un filtro supervisado va a tener en cuenta el último atributo. Aplicando este filtro pasaría como antes, la base de datos no quedaría equilibrada dejando algunas clases con 0 instancias, cosa que no nos interesa para el análisis de las bases de datos.

1.7 Ejercicio 7

Actualize su base de datos con los filtros anteriores que crea que tienen sentido con lo que sabe hasta ahora , indicando cuales va a usar.

Una vez analizados todos los filtros anteriores, y con los conocimientos que tengo, ahora mismo me quedaría con el filtro *ClassBalancer*, ya que es el único que hace que la base de datos desequilibrada, tenga un poco más de sentido. Esto no puedo afirmarlo ya que no he estudiado la pestaña de *classify*. Pero, a ciegas, es el único filtro que puede hacer que el análisis de la base de datos Ecoli sea un poco más clara.

2 Práctica 2.1: Clasificación, Regresión y Clustering con Weka

2.1 Ejercicio 1

Con su base de datos elegida, en el entorno Explorer utilice el algoritmo de clasificación IB1 con un 5-fold crossvalidation. Visualice la clasificación realizada, interpretando los resultados.

Para esta práctica, la base de datos elegida es Ecoli. Una vez cargada en el entorno Explorer de Weka, procederemos a aplicar el algoritmo de clasificación IB1. Este algoritmo se encarga de clasificar el conjunto de datos según el vecino más K más cercano. Además, este algoritmo utiliza la distancia Euclídea para encontrar la instancia del conjunto *Train* a la instancia del conjunto *Test* que hayamos seleccionado, para así poder predecir a qué clase pertenece la instancia del conjunto *Train*.

Para usar este clasificador, nos vamos al apartado *Classify*, una vez ahí, tenemos que elegir el clasificador a usar. En nuestro caso se encuentra en el apartado *Classifier* \rightarrow *lazy* \rightarrow *IBK*. La configuración de dicho clasificador puede ser observada en la Figura 34, como podemos observar, podemos elegir el número de *Neighbour* que queremos utilizar, en nuestro caso lo hemos igualado a 1.

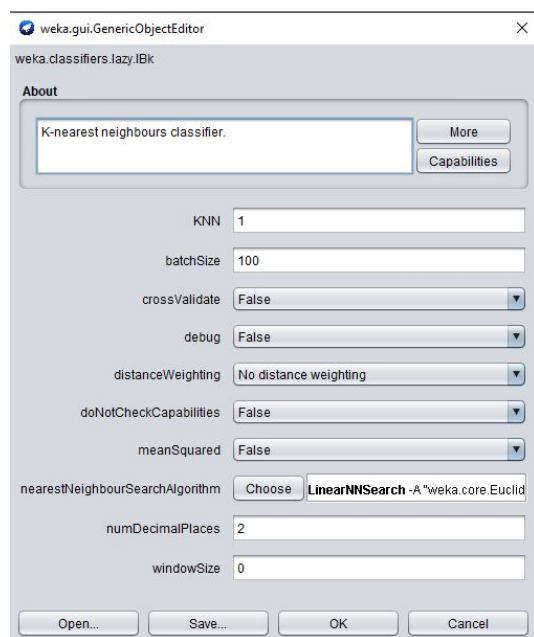


Figure 34: Configuración para el clasificador *IBK*

Seguidamente, hay que seleccionar el tipo de test que vamos a utilizar. Esto se encuentra en el apartado *Test options*, en el cual, como la Figura 35 indica,

hemos seleccionado el test *Cross-validation*, el cual hemos elegido que contenga 5 *folds*. Es decir, nuestro algoritmo va a utilizar 5 subconjuntos del conjunto de datos como *Test* en cada iteración.

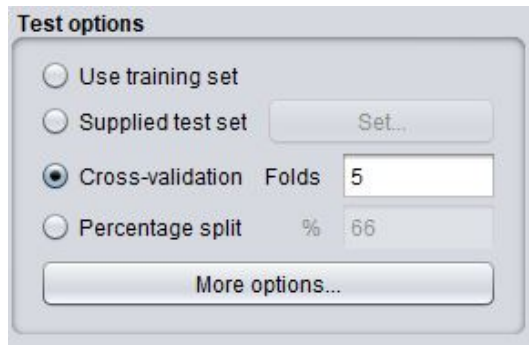


Figure 35: Configuración para el tipo de test

Una vez configurado todo, procedemos a aplicar el clasificador. Como salida, obtenemos varia información sobre cómo este algoritmo ha clasificado la base de datos. A continuación, procedemos a explicar cada uno de los apartados de la información resultante.

Primero, como la Figura 36 representa, podemos observar la información sobre nuestra base de datos: nombre, número de instancias, atributos y el test utilizado.

```

=== Run information ===

Scheme:      weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Relation:    ecoli
Instances:   336
Attributes:  8
             mcg
             gvh
             lip
             chg
             aac
             alml
             alml2
             class
Test mode:   5-fold cross-validation

```

Figure 36: Información sobre la base de datos Ecoli

Seguidamente, podemos encontrarnos un resumen en donde se puede observar la precisión, error y más, relacionados con la clasificación de la base de datos. Dicha información se puede observar en la Figura 37. A continuación procederemos a explicar uno a uno dicha información.

1. **Correctly classified instances:** Este apartado nos indica la cantidad de instancias que han sido clasificadas correctamente. Como vemos, hemos obtenido que el 81% de las instancias han sido correctamente clasificadas, siendo una cantidad de 272 instancias. Esto quiere decir que nuestro algoritmo tiene un gran porcentaje de instancias bien clasificadas.

```

=== Summary ===

Correctly Classified Instances      272          80.9524 %
Incorrectly Classified Instances    64           19.0476 %
Kappa statistic                    0.7372
Mean absolute error                 0.0526
Root mean squared error             0.2153
Relative absolute error             28.7227 %
Root relative squared error         71.3566 %
Total Number of Instances          336

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,944    0,052    0,931     0,944    0,938      0,891    0,943     0,908     cp
      0,740    0,077    0,740     0,740    0,740      0,663    0,822     0,624     im
      0,827    0,046    0,768     0,827    0,796      0,758    0,897     0,653     pp
      0,486    0,050    0,531     0,486    0,507      0,454    0,739     0,350     imU
      0,750    0,006    0,882     0,750    0,811      0,803    0,871     0,668     om
      1,000    0,003    0,833     1,000    0,909      0,911    0,998     0,807     omL
      0,000    0,006    0,000     0,000    0,000      -0,006    0,645     0,010     imL
      0,000    0,003    0,000     0,000    0,000      -0,004    0,647     0,010     imS
Weighted Avg.  0,810    0,052    0,805     0,810    0,807      0,757    0,880     0,719

=== Confusion Matrix ===

  a   b   c   d   e   f   g   h   <-- classified as
135  3   5   0   0   0   0   0 | a = cp
 3  57  2  13   0   0   1   1 | b = im
 6   1  43   0   2   0   0   0 | c = pp
 1  15   1  17   0   0   1   0 | d = imU
 0   0   4   1  15   0   0   0 | e = om
 0   0   0   0   0   5   0   0 | f = omL
 0   1   0   0   0   1   0   0 | g = imL
 0   0   1   1   0   0   0   0 | h = imS

```

Figure 37: Información sobre el algoritmo de clasificación IB1 aplicado sobre la base de datos Ecoli

2. **Incorrectly classified instances:** Cantidad de instancias incorrectamente clasificadas. El porcentaje es contrario al apartado anterior, siendo el porcentaje un 19%, dejando 64 de las 336 instancias mal clasificadas.
3. **Kappa statistic:** Este apartado expresa el estadístico *KAPPA*, el cual compara la concordancia del conjunto de datos con un modelo que podría ocurrir por azar. El rango de este estadístico está comprendido entre $[-1, 1]$, siendo un parámetro a maximizar. En nuestro caso, hemos conseguido 0.7372, esto quiere decir que nuestro modelo tiene una mejor concordancia que si hubiéramos utilizado el azar.
4. **Mean absolute error:** Este tipo de error representa la diferencia entre el valor real y el valor que hemos obtenido, o de misma manera, la diferencia entre la clase predicha y la clase real. En nuestro caso, hemos obtenido un error del 0.0526, siendo un error muy bajo, lo que indica que nuestro clasificador ha clasificado bien nuestra base de datos.
5. **Root mean squared error:** Es otro tipo de error que indica lo mismo

que el anterior. La diferencia de este con el anterior es la manera de calcularlo.

6. **Relative absolute error:** Indica el error absoluto, esto indica la relación entre el error absoluto y el valor medio. En nuestro caso es 28.72%.
7. **Root relative squared error:** Es una error que calcula lo mismo que el apartado anterior, la diferencia es la fórmula que se utiliza para su cálculo. En nuestro caso, hemos obtenido un 71.36%.

Después de todos estos datos, podemos encontrarnos con las métricas para la clasificación multiclase. A continuación, procedemos a explicar el significado de cada columna:

1. **TP Rate:** Es columna representa el conjunto de patrones positivos que han sido predichos como positivos, es decir, la cantidad de patrones que han sido predichos correctamente. Como podemos observar, en la última fila de esta columna aparece el *TP Rate* global, que como hemos visto antes, es de un 81%. De aquí podemos destacar las clases *cp* y *omL*, ya que se cuentan con el mayor porcentaje. Por el contrario, las clases minoritarias *imL* y *imS*, cuentan con un porcentaje de 0. Esto es debido a que contamos con un conjunto de datos desbalanceado, ocurriendo la posibilidad de que ningún patrón sea predicho como perteneciente de estas dos últimas clases.
2. **FP Rate:** Esta columna representa el conjunto de patrones negativos que han sido predichos como positivos, es decir, representa los patrones que no han sido predichos correctamente. Como ya sabemos, es un porcentaje que hay que minimizar, el cual, en nuestra clasificación tiene un valor de 0.052, siendo este un valor muy pequeño. Esto quiere significar que nuestro conjunto de datos está bien clasificando y dejando muy pocos patrones negativos como positivos.
3. **Precision:** Como ya sabemos, esta columna representa el porcentaje de patrones positivos predichos como positivos, pero esta vez comparados con el total de patrones predichos como positivos. Esta métrica es calculada dividiendo el número de patrones positivos predichos como positivos (*TP Rate*), entre la suma de los patrones positivos y negativos predichos como positivos. Como podemos observar, es un porcentaje bastante similar al *TP Rate* (0.805 frente a 0.810), lo cual indica que nuestro conjunto de datos ha sido clasificado de una manera bastante correcta. Esto ocurre al tener un valor de *FP Rate* muy bajo.
4. **Recall:** Nos devuelve el mismo valor que *TP Rate*.
5. **F-Measure:** Es una métrica que nos interesa maximizar, ya que junta las métricas *TP Rate* y *Precision* y nos proporciona información sobre lo bien que está clasificado nuestro modelo. En nuestro caso el porcentaje global

obtenido es 0.807, siendo este un buen valor pero no lo suficientemente bueno. Esto es debido, como ya hemos repetido varias veces, a que nuestro modelo está desbalanceado.

6. **MCC:** En esta columna está representado *Matthews Correlation Coefficient*, o en español, Coeficiente de Correlación de Matthews. Este coeficiente es usado para calcular la asociación entre dos variables binarias.
7. **ROC Area:** Esta columna representa el *AUC* (Area Under the Curve). Es una métrica usada para calcular la probabilidad de que una instancia aleatoria sea clasificada como una instancia positiva. En nuestro caso, hemos obtenido un valor bastante alto, 0.880. Como podemos observar, todos los valores están que han de ser maximizados, tienen un valor entorno al 80%, dejando nuestro conjunto de datos bastante bien clasificado, aunque no perfectamente.

Por último, podemos observar la matriz de confusión, la cual nos indica el número de instancias de cada clase que han sido correcta o incorrectamente clasificadas. Un ejemplo sería la clase *cp*, la cual de las 143 instancias, 135 han sido clasificadas correctamente, 3 como instancias de la clase *im* y 5 como instancias de la clase *pp*. Esto da lugar al 94.4% que hemos comentado anteriormente en la columna *TP Rate*.

Además, también se puede observar lo ya comentado, las clases minoritarias (*imL* y *imS*) no se han clasificado correctamente, explicando así el 0.0% de las instancias bien clasificadas.

2.2 Ejercicio 2

Ejecuta con la misma configuración en el entorno *Experimenter* el algoritmo IBK con $k=2$ y $k=3$. Fije el número de repeticiones de cada 5-fold crossvalidation a 1.

1. **Calcule la media y la desviación típica de las medidas: Accuracy, Kappa, RMSE, F-Measure. Use para ello el fichero .csv generado, interpretando los resultados.**

Para realizar este ejercicio, primero abrimos el entorno *Experiment* de Weka. Seguidamente, como la Figura 38 representa, hay que realizar ciertas configuraciones. Primero, debemos seleccionar la ruta donde se guardará el archivo *.csv* con los resultados obtenidos. Después, tenemos que configurar el número de *folds* con el que vamos a clasificar junto con el número de repeticiones a realizar. Además, tenemos que señalar qué tipo de clasificadores vamos a utilizar, el cual tiene una configuración exactamente igual que en la Figura 34. Aquí es donde le decimos al programa el número de vecinos con el que queremos trabajar (2 y 3). Finalmente, tenemos que añadir la base de datos a la cual le vamos a aplicar esta clasificación.

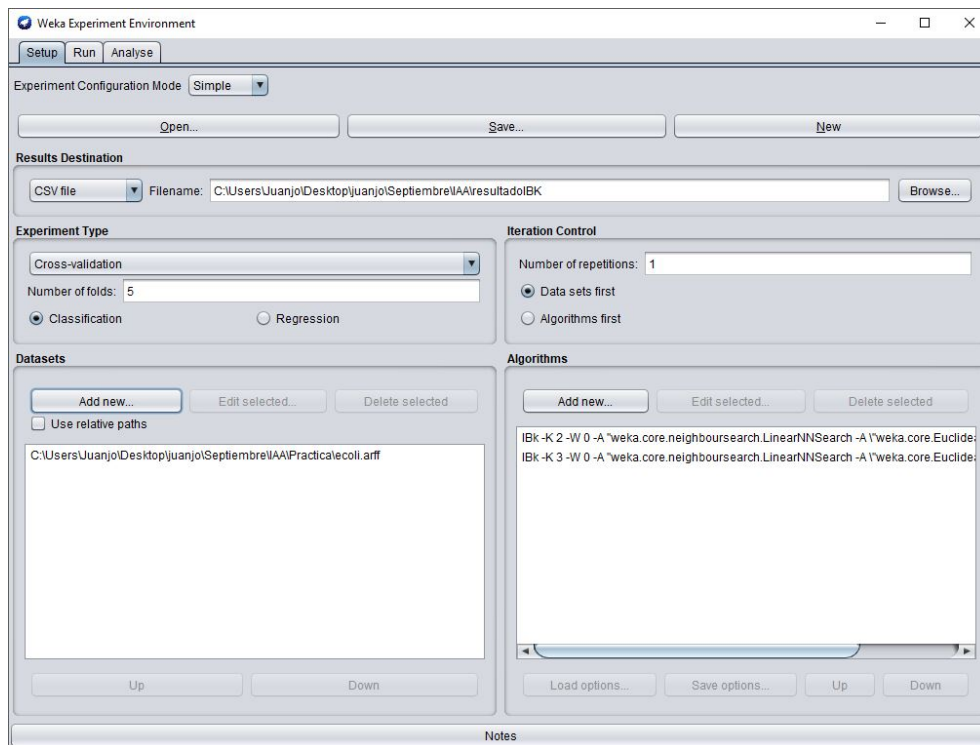


Figure 38: Configuración de Experiment Environment

2. Realice una tabla resumen con los resultados obtenidos, de manera que se puedan comparar éstos según el “K” elegido (2, 3). Comente la tabla, interprete los resultados.

Para realizar este ejercicio, nos tenemos que ir al apartado *Analyse* del entorno *Experiment*. Una vez ahí, iremos añadiendo a las tablas los valores de la media y desviación típica de las medidas Accuracy, Kappa, RMSE y F-Measure. Seguidamente, podemos observar, en las figuras 39 y 40 las métricas obtenidas.

	ACCURACY	KAPPA	RMSE	F-MEASURE
Media	83'0509	0'7611	0'1835	0'9468
Desv. Típica	5'3633	0'00770	0'0278	0'0240

Figure 39: Métricas para IB2

Seguidamente, vamos a explicar el significado de las métricas utilizadas:

- (a) **Accuracy:** Es el número de instancias bien clasificadas. Como pode-

	ACCURACY	KAPPA	RMSE	F-MEASURE
Media	86'6155	0'8135	0'1699	0'9632
Desv. Típica	4'3112	0'0611	0'0232	0'0272

Figure 40: Métricas para IB3

mos observar, al aumentar K , este porcentaje se ve aumentado, mejorando así la fiabilidad de la clasificación realizada.

- (b) **Kappa:** Como ya sabemos, indica la concordancia de nuestra clasificación con respecto a lo que podemos esperar aleatoriamente. Como ocurre con la anterior métrica, al aumentar K , éste también aumenta, mejorando así nuestra clasificación.
- (c) **RMSE:** Representa el la raíz cuadrada el derror medio. Como podemos observar, no existe mucha diferencia entre los dos valores, aunque podemos observar que a mayor número de K , menor es este error.
- (d) **F-Measure:** Como podemos observar, el valor de esta métrica aumenta cuando aumentamos el valor de K . Esto no es malo ya que esta métrica indica el número de instancias positivas clasificadas como positivas, lo cual es algo que hay que maximizar.

2.3 Ejercicio 3

Usando el entorno Explorer ejecute el algoritmo Logistic con su base de datos, use un 5-fold crossvalidation como ya se hizo anteriormente con el algoritmo IBK.

1. Analice los modelos obtenidos, métricas, las variables que podrían ser más influyentes (valores beta), variables que no se usan, etc, y compare ambos métodos (use tablas legibles).

Para analizar este ejercicio, debemos de irnos al apartado *Classify* de Weka. Una vez ahí, debemos seleccionar el algoritmo *Logistic* dentro del conjunto de clasificadores, hemos dejado la configuración por defecto. Una vez hecho esto, bastaría, como en los casos anteriores, cambiar el número de *folds* dentro de la sección *Test options*. Seguidamente, procedemos a ver los resultados.

Como podemos observar en la Figura 41, al principio Weka nos presenta la información general sobre nuestra base de datos: algoritmo elegido, nombre de la base de datos elegida, número de instancias, número de atributos y finalmente el test elegido, que en nuestro caso es *Cross Validation with 5 folds*.

Acto seguido, podemos encontrarnos con dos tablas, como podemos observar en la Figura 42, una denominada *Coefficients* y la otra *Odds Ratios*.

```

=== Run information ===

Scheme:      weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4
Relation:    ecoli
Instances:   336
Attributes:  8
              mcg
              gvh
              lip
              chg
              aac
              alm1
              alm2
              class
Test mode:   5-fold cross-validation

```

Figure 41: Información general sobre el resultado del algoritmo *Logistic*

```

=== Classifier model (full training set) ===

Logistic Regression with ridge parameter of 1.0E-8
Coefficients...

```

Variable	Class cp	im	pp	imU	om	omL	imL
mcg	-14.8037	-15.7923	-8.3117	-3.6939	-5.7806	31.442	76.2085
gvh	-3.681	1.8815	11.78	-3.1536	19.3118	-53.071	-258.2916
lip	-44.3372	17.4508	-33.1851	21.6587	28.1532	91.3962	197.6524
chg	-44.8643	-25.0369	-17.3309	-10.8965	6.8927	68.2399	264.5702
aac	-2.6992	-3.5114	-2.1942	-3.9119	37.8913	17.0628	-63.8654
alm1	-25.4379	10.1341	-5.8038	3.2236	-16.9811	15.8884	-41.756
alm2	8.7662	4.8733	-2.5595	7.1817	-17.3419	-57.5945	7.074
Intercept	66.9778	8.4836	31.8368	-2.8371	-30.8192	-90.975	-148.7293

```

Odds Ratios...

```

Variable	Class cp	im	pp	imU	om	omL	imL
mcg	0	0	0.0002	0.0249	0.0031	4.5195663930222125E13	1.2501170184768773E33
gvh	0.0252	6.5635	130609.2894	0.0427	243796053.5993	0	0
lip	0	46308026.8153	0	2548255005.0434	1.6056182531013005E12	4.9302549668310795E39	2.934684670913565E68
chg	0	0	0	0	985.0365	4.3271527363465875E29	7.968820986267856E114
aac	0.0673	0.0299	0.1114	0.02	3.1265032360402252E16	25719626.1289	0
alm1	0	25188.1328	0.003	25.1183	0	7947611.6253	0
alm2	6413.5211	130.746	0.0773	1315.1823	0	0	1180.9146

```

Time taken to build model: 0.3 seconds

```

Figure 42: Tablas con métricas sobre el resultado del algoritmo *Logistic*

Lo que primero cabe destacar de estas dos tablas, es que solamente hay 7 de las 8 clases, esto es debido a que la última clase es calculada restandole a 1 el resto de probabilidades.

Con respecto a la tabla *Coefficients* hay que decir que muestra el valor que representa cada uno de los atributos en nuestro modelo. Es decir, a más valor, más influye a la hora de decidir si ese atributo pertenece a esa clase o no. Un ejemplo podría ser la clase *im*, la cual, como podemos observar, que el atributo *lip* es el que más valor tiene a la hora de decidir si esa instancia pertenece a esa clase o no. Por el contrario, el atributo que menor valor de decisión tiene es *chg*.

Con respecto a la segunda tabla, *Odds Ratio*, presenta la influencia que tiene un atributo cuando lo cambiamos en la predicción del modelo. Por seguir con el mismo ejemplo, si nos fijamos en la clase *im*, el atributo

lip es el más influyente, esto quiere decir que un cambio en este atributo supondrá una predicción muy distinta. Como nota, si el valor de los atributos es 0, esto quiere decir que un cambio en dicho atributo no influiría para nada en la predicción.

Por último, como la Figura 43 representa, podemos encontrarlo con las métricas y con la matriz de confusión al aplicar el test *Cross Validation with 5 folds*.

```

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      291          86.6071 %
Incorrectly Classified Instances    45          13.3929 %
Kappa statistic                    0.8153
Mean absolute error                 0.0474
Root mean squared error             0.164
Relative absolute error             25.8998 %
Root relative squared error         54.3619 %
Total Number of Instances          336

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,965    0,031    0,958    0,965    0,962      0,933    0,988    0,976    cp
      0,844    0,058    0,813    0,844    0,828      0,776    0,957    0,906    im
      0,885    0,025    0,868    0,885    0,876      0,853    0,951    0,898    pp
      0,600    0,033    0,677    0,600    0,636      0,598    0,929    0,609    imU
      0,800    0,009    0,842    0,800    0,821      0,810    0,919    0,859    om
      1,000    0,006    0,714    1,000    0,833      0,843    0,998    0,853    omL
      0,000    0,006    0,000    0,000    0,000      -0,006    0,883    0,069    imL
      0,000    0,000    ?        0,000    ?          ?        0,141    0,005    imS
Weighted Avg.    0,866    0,034    ?        0,866    ?          ?        0,959    0,890

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  <-- classified as
138  1  3  0  1  0  0  0  |  a = cp
  2 65  0  9  0  0  1  0  |  b = im
  3  1 46  0  2  0  0  0  |  c = pp
  1 12  0 21  0  0  1  0  |  d = imU
  0  0  3  1 16  0  0  0  |  e = om
  0  0  0  0  0  5  0  0  |  f = omL
  0  0  0  0  0  2  0  0  |  g = imL
  0  1  1  0  0  0  0  0  |  h = imS

```

Figure 43: Métricas y matriz de conguisión al aplicar el algortimo *Logistic*

2. Asocie las formulas de las salidas por clase aportadas en las transparencias de esta práctica con los modelos de probabilidad obtenidos en la salida de Weka.

Para calcular la probabilidad de cada clase, hay que utilizar dos fórmulas. La primera sería la siguiente:

$$f_1(\mathbf{x}, \hat{\theta}) = \hat{\beta}_0 + \sum_{i=1}^n \hat{\beta}_i x_i$$

Donde β_0 es el sesgo o intercept de la clase, β_i es el valor de cada atributo en la clase y x_i es el valor que toma la variable. Con esto, conseguimos

el valor de f_1 , pero para calcular la probabilidad de cada clase, hay que aplicar este resultado a la siguiente fórmula:

$$p_1(\mathbf{x}, \hat{\theta}) = \frac{e^{f_1(\mathbf{x}, \hat{\theta})}}{1 + \sum_{k=1}^{K-1} e^{f_k(\mathbf{x}, \hat{\theta})}}$$

2.4 Ejercicio 4

Usando el entorno Explorer ejecute el algoritmo SimpleLogistic con su base de datos, use un 5-fold crossvalidation como ya se hizo anteriormente con el algoritmo IBK.

1. Analice los modelos obtenidos, métricas, las variables que podrían ser más influyentes (valores beta), variables que no se usan, etc, y compare ambos métodos (use tablas legibles)

Como hemos hecho en el ejercicio anterior, lo primero es establecer la configuración de este algoritmo. Como antes, hemos dejado la configuración por defecto, y aparte, hemos seleccionado el número de *folds* que queremos para nuestro test, 5. Una vez hecho esto, podemos comenzar a aplicar nuestro algoritmo.

Como en la Figura 41, lo primero que nos muestra el resultado de este algoritmo, es la información general. Esta no ha cambiado con respecto al anterior. Seguidamente, como la Figura 44 expone, podemos observar la influencia de cada atributo en cada una de las clases. A diferencia con el algoritmo *Logistic*, ahora podemos observar las 8 clases con las que cuenta nuestra base de datos. Además, como podemos ver, no todos los atributos influyen en las clases, es por esto por lo que no aparecen todos los atributos en las clases. Cabe decir que, el primer número que vemos después del nombre de la clase, es el correspondiente al sesgo.

Podemos destacar que hay ciertos atributos que influyen más en las clases, como por ejemplo el atributo *alm1* dentro de la clase *im*. Otro ejemplo podría ser el atributo *aac* dentro de la clase *om*. Como antes, a mayor valor, mayor influencia dentro de la clase.

Class cp :	Class om :
11.47 +	-11.77 +
[mcg] * -6.7 +	[gvh] * 6.52 +
[gvh] * -4.41 +	[aac] * 16.45 +
[alm1] * -9.9	[alm2] * -3.95
Class im :	Class omL :
-3.37 +	-6.01 +
[mcg] * -5.31 +	[lip] * 10.81 +
[alm1] * 11.97	[alm2] * -5.84
Class pp :	Class imL :
-2.84 +	-14.32 +
[mcg] * 1.38 +	[lip] * 9.77 +
[gvh] * 10.5 +	[chg] * 7.19 +
[aac] * -4.87 +	[alm1] * 3.46
[alm2] * -1.8	
Class imU :	Class imS :
-6.52 +	-7.71 +
[mcg] * 4.84 +	[mcg] * 8.48
[gvh] * -2.61 +	
[alm1] * 4.24 +	
[alm2] * 3.92	

Figure 44: Influencia de cada atributo en cada clase

Seguidamente, como podemos observar en la Figura 45, lo siguiente que obtenemos como resultado son las métricas y la matriz de confusión. Como comparativa, vamos a realizar una tabla en la cual vamos a comparar las métricas de los algoritmos *Logistic* y *SimpleLogistic*. Dichas métricas han sido obtenidas de la información general que nos aporta dichos algoritmos.

```

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      286          85.119 %
Incorrectly Classified Instances    50           14.881 %
Kappa statistic                    0.7935
Mean absolute error                 0.0577
Root mean squared error             0.1648
Relative absolute error             31.5084 %
Root relative squared error         54.6459 %
Total Number of Instances          336

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,965    0,047    0,939     0,965    0,952     0,915    0,990     0,985     cp
      0,857    0,073    0,776     0,857    0,815     0,758    0,964     0,903     im
      0,808    0,032    0,824     0,808    0,816     0,782    0,957     0,893     pp
      0,543    0,027    0,704     0,543    0,613     0,580    0,949     0,626     imU
      0,800    0,003    0,941     0,800    0,865     0,860    0,992     0,929     om
      1,000    0,003    0,833     1,000    0,909     0,911    0,998     0,810     omL
      0,000    0,009    0,000     0,000    0,000     -0,007    0,587     0,014     imL
      0,000    0,000    ?         0,000    ?         ?         0,563     0,010     imS
Weighted Avg.    0,851    0,045    ?         0,851    ?         ?         0,970     0,897

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  <-- classified as
138  1  4  0  0  0  0  0 | a = cp
  2 66  0  7  0  0  2  0 | b = im
  5  4 42  0  1  0  0  0 | c = pp
  1 13  1 19  0  0  1  0 | d = imU
  1  0  3  0 16  0  0  0 | e = om
  0  0  0  0  0  5  0  0 | f = omL
  0  0  0  1  0  1  0  0 | g = imL
  0  1  1  0  0  0  0  0 | h = imS

```

Figure 45: Métricas y matriz de congucción al aplicar el algoritmo *SimpleLogistic*

Como podemos observar en la Figura 46, existe muy poca diferencia con respecto a los dos algoritmos, aunque tenemos que decir que el algoritmo *Logistic* obtiene unos errores más bajos, con lo que podríamos decir que obtenemos un mejor modelo para nuestra base de datos usando una regresión logística que con una regresión logística simple. Aunque, como hemos dicho, no podemos observar mucha diferencia entre los dos.

	CCR	ICI	KAPPA	MAE	RMSE	RAE	RRSE
Logistic	86'6071	13'3929	0'8153	0'0474	0'164	25'8998	54'3619
SimpleLogistic	85'119	14'881	0'7935	0'0577	0'1648	31'5084	54'6459

Figure 46: Información general sobre los dos algoritmos

2. Asocie las formulas de las salidas por clase aportadas en las transparencias de esta práctica con los modelos de probabilidad obtenidos en la salida de Weka.

Como antes, tenemos que utilizar dos fórmulas para el cálculo de las prob-

habilidades. La primera sería:

$$f_1(\mathbf{x}, \hat{\theta}) = \hat{\beta}_0 + \sum_{i=1}^n \hat{\beta}_i x_i$$

Donde β_0 es el sesgo o intercept de la clase, β_i es el valor de cada atributo en la clase y x_i es el valor que toma la variable. Una vez calculado el valor de f_1 , para calcular la probabilidad de la clase hay que utilizar la siguiente fórmula:

$$p_1(\mathbf{x}, \hat{\theta}) = \frac{e^{f_1(\mathbf{x}, \hat{\theta})}}{1 + \sum_{k=1}^{K-1} e^{f_k(\mathbf{x}, \hat{\theta})}}$$

3 Práctica 2.2: Agrupamiento o Clustering

3.1 Ejercicio 1

Use el algoritmo K-means y seleccione la opción Use training set para la base de datos Iris. Para ello ignore el atributo de clase.

Para la realización de este ejercicio, debemos abrir el entorno *Explorer* de Weka. Una vez ahí, abrimos nuestra base de datos *Iris* y finalmente deberemos de ir a la sección *Cluster*.

Dentro de éste, deberemos elegir el algoritmo *SimpleKMeans*, el cuál deberemos de ir configurando dependiendo del número de cluster que deseemos. Además, al tener una atributo categórico, debemos de ingonarlo, para ello, como se ha explicado en la práctica, deberemos hacer click sobre el apartado *Ignore attributes* y una vez ahí, seleccionar el atributo *class*. Finalmente, antes de ejecutar nuestro algoritmo, debemos de configurarlo, para ello, vamos a utilizar un máximo de 500 iteraciones, y *seed=10*. Damos por supuesto que el número de clusters hay que ir cambiándolo. Seguidamente, comenzamos a visualizar los datos pertinentes.

Como en todos los algortimos anteriores, lo primero que podemos observar como resultado, es la información general usada, pero nosotros nos vamos a centrar a partir de la sección *KMeans*.

1. Número de cluster = 2

Como podemos ver en la Figura 47, se observa, en el siguiente orden, *SSE*, coordenadas inciales de los centroides, coordenadas finales de los centroides y finalmente el número de instancias por cluster. Esta información tendrá siempre el mismo orden, dando igual el número de cluster usado.

```

kMeans
=====

Number of iterations: 7
Within cluster sum of squared errors: 62.1436882815797

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute          Full Data          Cluster#
                   (150.0)          0          1
                   (100.0)         (50.0)
=====
sepalength          5.8433          6.262          5.006
sepalwidth           3.054          2.872          3.418
petallength          3.7587          4.906          1.464
petalwidth           1.1987          1.676          0.244
class               Iris-setosa Iris-versicolor  Iris-setosa

Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      100 ( 67%)
1       50 ( 33%)

```

Figure 47: Resultado de aplicar el algoritmo *SimpleKMeans* con $K = 2$

2. Número de cluster = 3


```

kMeans
=====

Number of iterations: 3
Within cluster sum of squared errors: 7.817456892309574

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica

Missing values globally replaced with mean/mode

Final cluster centroids:

```

Attribute	Full Data (150.0)	Cluster# 0 (50.0)	1 (50.0)	2 (50.0)
sepalwidth	5.8433	5.936	5.006	6.588
sepalwidth	3.054	2.77	3.418	2.974
petalwidth	3.7587	4.26	1.464	5.552
petalwidth	1.1987	1.326	0.244	2.026
class	Iris-setosa Iris-versicolor		Iris-setosa	Iris-virginica

```

=====

Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances
0      50 ( 33%)
1      50 ( 33%)
2      50 ( 33%)

```

Figure 48: Resultado de aplicar el algoritmo *SimpleKMeans* con $K = 3$

3. Número de cluster = 4

```

KMeans
=====

Number of iterations: 4
Within cluster sum of squared errors: 6.613823274690356

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4,Iris-versicolor
Cluster 1: 6.2,2.9,4.3,1.3,Iris-versicolor
Cluster 2: 6.9,3.1,5.1,2.3,Iris-virginica
Cluster 3: 5.5,4.2,1.4,0.2,Iris-setosa

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute      Full Data      Cluster#
                (150.0)        0          1          2          3
                (24.0)        (26.0)        (50.0)        (50.0)
=====
sepal.length    5.8433         6.3292         5.5731         6.588         5.006
sepal.width     3.054          2.9792         2.5769         2.974         3.418
petal.length    3.7587         4.6            3.9462         5.552         1.464
petal.width     1.1987         1.4625         1.2            2.026         0.244
class           Iris-setosa  Iris-versicolor Iris-versicolor Iris-virginica  Iris-setosa

Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      24 ( 16%)
1      26 ( 17%)
2      50 ( 33%)
3      50 ( 33%)

```

Figure 49: Resultado de aplicar el algoritmo *SimpleKMeans* con $K = 4$

Como se puede observar en las figuras 47, 48 y 49, la suma cuadrada del error, es disminuida conforme el número de cluster aumenta. Éste se ve disminuído drásticamente cuando cambiamos el número de cluster de 2 a 3, siendo *SSE* un parámetro a disminuir.

Además, también se puede observar que la cantidad de instancias en cada clase, se ve más equitativa cuando utilizamos 3 clusters, siendo esto también un parámetro que hay que equilibrar.

Viendo estos resultados, podríamos afirmar que el número óptimo de cluster a utilizar para nuestra base de datos *Iris*, es de 3. Ya que aunque tiene mayor *SSE*, el número de instancias en cada clase se ve equilibrado (siendo un 33% en cada cluster), cosa que no pasa cuando utilizamos 4 clusters.

Seguidamente procedemos a ver gráficamente qué instancias pertenecen a cada cluster. Los atributos utilizados para ello son *petal.length* y *petal.width*.

1. Número de cluster = 2

Como podemos observar en la Figura 50, las coordenadas de los clusters finales son: para el *Cluster0* = (4.906,1.676) y para *Cluster1* = (1.464,0.244). Esto también puede ser observado en la Figura 47.

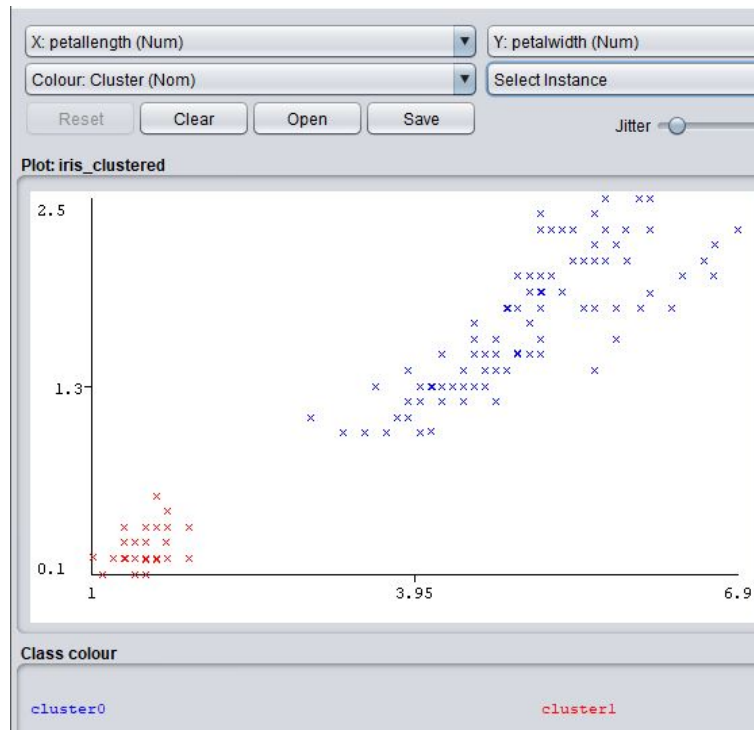


Figure 50: Gráfica resultante de aplicar el algoritmo *SimpleKMeans* con $K = 2$

2. Número de cluster = 3

Ahora, las coordenadas de los tres centroides son: $Cluster0 = (4.26, 1.326)$, $Cluster1 = (1.464, 0.244)$ y para $Cluster2 = (5.552, 2.026)$.

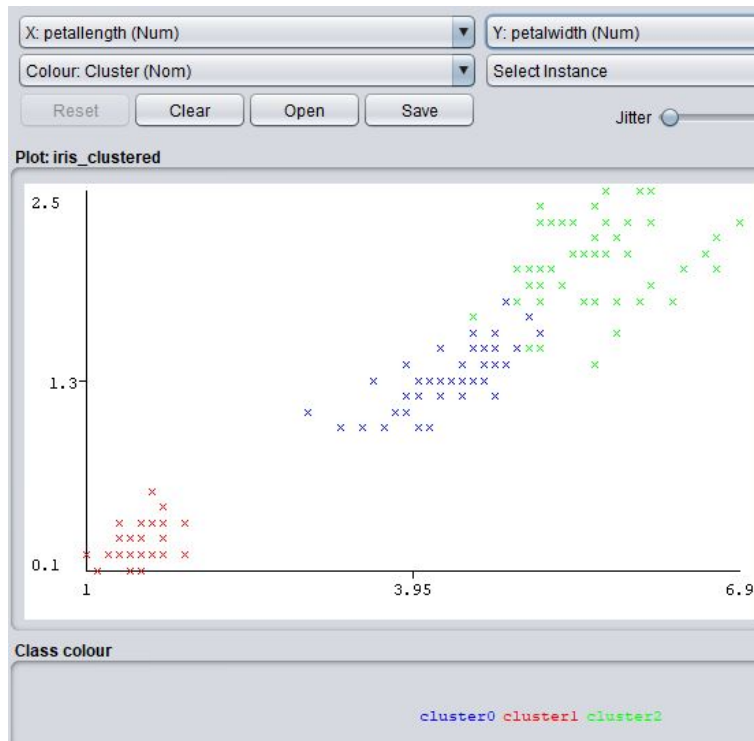


Figure 51: Gráfica resultante de aplicar el algoritmo *SimpleKMeans* con $K = 3$

3. Número de cluster = 4

Finalmente, para $K = 4$, las coordenadas de los centroides son: $Cluster0 = (4.6, 1.4625)$, $Cluster1 = (3.9462, 1.2)$, $Cluster2 = (5.552, 2.026)$ y para $Cluster3 = (1.464, 0.244)$.

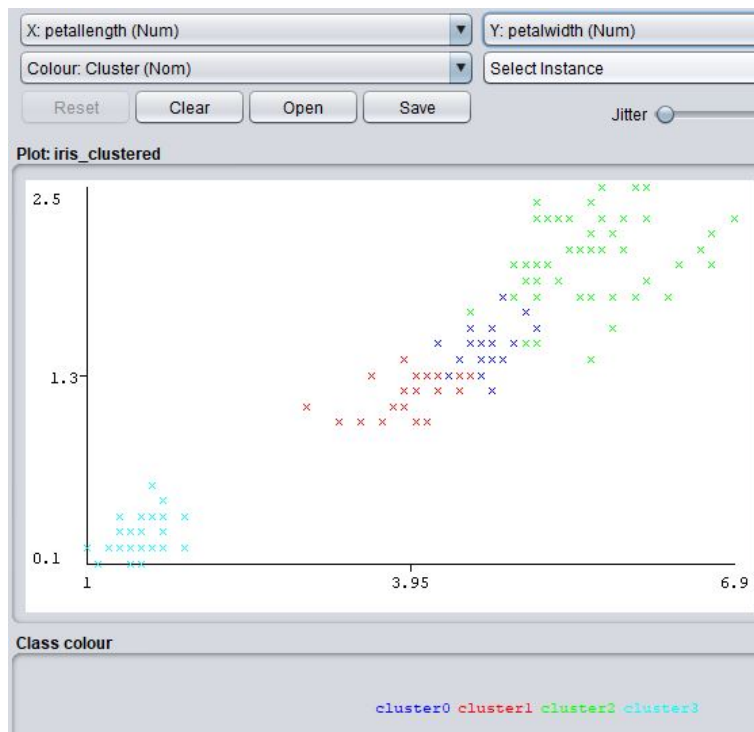


Figure 52: Gráfica resultante de aplicar el algoritmo *SimpleKMeans* con $K = 4$

Una vez visto también las gráficas, podemos decir que el número optimos de cluster a utilizar para la base de datos *Iris* es $K = 3$, ya que los datos no son entremezclados como en el caso de $K = 4$.

3.2 Ejercicio 2

Cargue su base de datos y analice qué ocurre al aplicar K-means, fijando k =número-de-clases de la base de datos. Discuta los clusters creados.

Para este ejercicio, tenemos que hacer como en el anterior. Primero debemos cargar nuestra base de datos *Ecoli*, y seguidamente irnos al apartado *Cluster*, en el cuál, lo configuraremos de tal manera que aplique el algoritmo *SimpleK-Means*, pero esta vez con 8 clusters (los 8 atributos de los que consta esta base de datos). Una vez hecho esto, realizaremos el experimento pero, primero sin ignorar el atributo *class* y segundo, ignorándolo. El resultado puede ser observado en las figuras 53 y 54.

Como podemos observar gráficamente, cuando ignoramos el atributo *class*, Figura 53, los clusters que hemos obtenido están muy dispersos, esto es debido a que cada atributo tiene un gran peso a la hora de realizar estos cluster. En cambio, si no ignoramos este atributo, Figura 54, podemos observar que los clusters están

más juntos.

Esto también puede ser observado viendo los resultados analíticos de cada algoritmo. A diferencia de cuando ignoramos el atributo, ($SSE=48.2646$), cuando no es ignorado, el parámetro SSE, es mucho menor (18.5648), lo que hace que cuando no es ignorado, el error producido por el algoritmo sea mucho menor, es decir, que los clusters son más similares cuando el atributo *class* no es ignorado, siendo éste, un factor deseado.

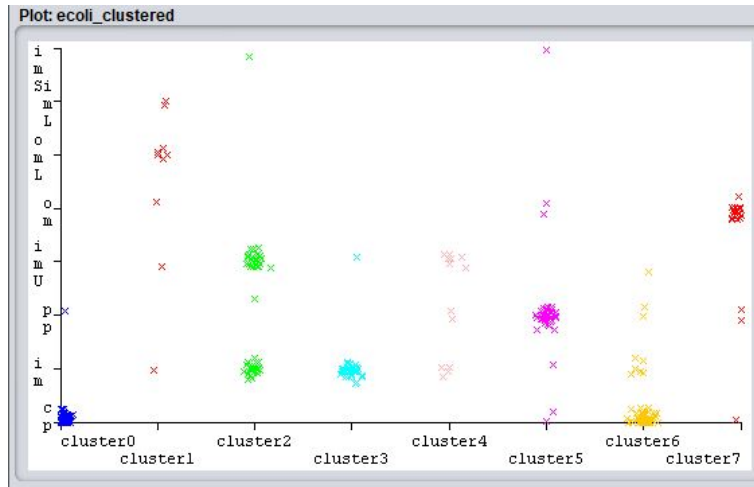


Figure 53: Gráfica resultante de aplicar el algoritmo *SimpleKMeans* ignorando el atributo *class*

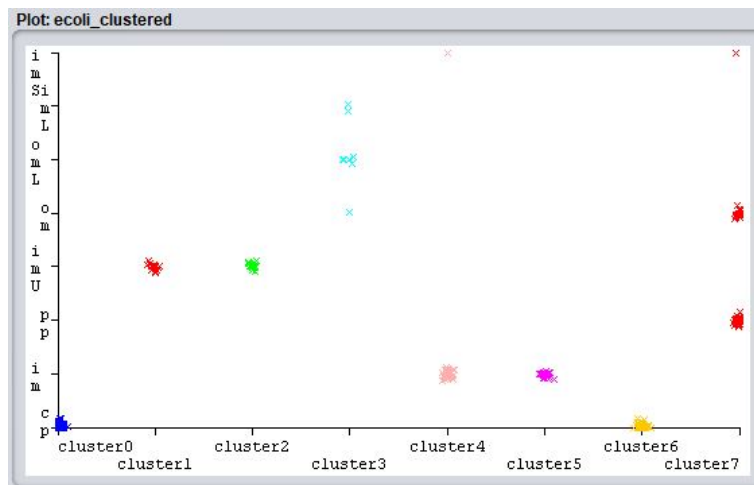


Figure 54: Gráfica resultante de aplicar el algoritmo *SimpleKMeans* sin ignorar el atributo *class*

3.3 Ejercicio 3

¿Qué ocurriría en K-means si fijásemos el número de clusters igual al número de patrones de una base de datos?, ¿De qué depende que se encuentren unos clusters u otros?.

Como ya sabemos, los centroides son inicializados de manera aleatoria. Al ser de esta manera, podría ocurrir que cada patrón se convirtiera en un cluster, ocurriendo así un sobretratamiento de los datos. Esto no es así siempre porque, como hemos dicho al principio, los centroides son inicializados de manera aleatoria.

3.4 Ejercicio 4

Utilizando su base de datos, el algoritmo K-means, y seleccionando la variable de clase (opción “Classes to clusters evaluation”), ¿con qué número de clusters y con qué métrica de distancia obtiene mejores resultados?.

Para la realización de este ejercicio, vamos a utilizar las siguientes configuraciones:

1. Como hemos observado, obtenemos un mejor valor de SSE cuando ignoramos el atributo *class*, con lo que para este ejercicio siempre va a ser ignorado.
2. Las semillas utilizadas serán 10, 15 y 20.
3. El número de clusters se hará de tal manera en la que veamos cómo cambia si utilizamos un número menor, igual o mayor que los atributos que componen nuestra base de datos *Ecoli*, estos valores serán 6, 8 y 10.
4. Además, también vamos a realizar un cambio en la función distancia utilizada. Vamos a utilizar tanto la distancia Euclídea y la distancia Manhattan, ya que la distancia de Chebyshev no puede ser utilizada para este algoritmo.

Dividiremos este ejercicio dependiendo del número de cluster, distancia y semilla utilizadas.

1. $K = 6$ y *Semilla* = 10 Distancia = Euclídea: Obtenemos un valor de $SSE = 29.0814$.
2. $K = 6$ y *Semilla* = 15 Distancia = Euclídea: Obtenemos un valor de $SSE = 20.88$.
3. $K = 6$ y *Semilla* = 20 Distancia = Euclídea: Obtenemos un valor de $SSE = 28.212$.

La media usando 6 clusters y con la distancia Euclídea de la variable $SSE = 26.0578$.

1. $K = 6$ y $Semilla = 10$ Distancia = Manhattan: Obtenemos un valor de $SSE = 146.4952$.
2. $K = 6$ y $Semilla = 15$ Distancia = Manhattan: Obtenemos un valor de $SSE = 144.4464$.
3. $K = 6$ y $Semilla = 20$ Distancia = Manhattan: Obtenemos un valor de $SSE = 142.4925$.

La media usando 6 clusters y con la distancia Manhattan de la variable $SSE = 144.478$.

1. $K = 8$ y $Semilla = 10$ Distancia = Euclídea: Obtenemos un valor de $SSE = 18.5648$.
2. $K = 8$ y $Semilla = 15$ Distancia = Euclídea: Obtenemos un valor de $SSE = 17.4504$.
3. $K = 8$ y $Semilla = 20$ Distancia = Euclídea: Obtenemos un valor de $SSE = 18.6215$.

La media usando 8 clusters y con la distancia Euclídea de la variable $SSE = 18.2122$.

1. $K = 8$ y $Semilla = 10$ Distancia = Manhattan: Obtenemos un valor de $SSE = 135.2691$.
2. $K = 8$ y $Semilla = 15$ Distancia = Manhattan: Obtenemos un valor de $SSE = 132.1412$.
3. $K = 8$ y $Semilla = 20$ Distancia = Manhattan: Obtenemos un valor de $SSE = 132.7337$.

La media usando 8 clusters y con la distancia Manhattan de la variable $SSE = 133.3813$.

1. $K = 10$ y $Semilla = 10$ Distancia = Euclídea: Obtenemos un valor de $SSE = 17.481$.
2. $K = 10$ y $Semilla = 15$ Distancia = Euclídea: Obtenemos un valor de $SSE = 15.868$.
3. $K = 10$ y $Semilla = 20$ Distancia = Euclídea: Obtenemos un valor de $SSE = 15.8844$.

La media usando 10 clusters y con la distancia Euclídea de la variable $SSE = 16.441$.

1. $K = 10$ y $Semilla = 10$ Distancia = Manhattan: Obtenemos un valor de $SSE = 131.1239$.

2. $K = 10$ y $Semilla = 15$ Distancia = Manhattan: Obtenemos un valor de $SSE = 126.108$.
3. $K = 10$ y $Semilla = 20$ Distancia = Manhattan: Obtenemos un valor de $SSE = 124.025$.

La media usando 10 clusters y con la distancia Manhattan de la variable $SSE = 127.085$.

Como podemos observar, a cuanto mayor sea el número de clusters, menor valor de SSE. Al ser mayor el valor utilizando la distancia Manhattan, hemos decidido guiarnos por la distancia Euclídea, y como vemos, el número idóneo de clúster es 10, en el cuál se consigue una menor media para el parámetro SSE. Si lo mirásemos en una gráfica, veríamos que el *Codo* de la función se encontraría en 10 clusters.

3.5 Ejercicio 5

Agrupe los 8 puntos de la figura en 3 clusters usando el algoritmo K-means. Los centroides iniciales se encuentran sobre los puntos A1, A4 y A7.

En la Figura 55 se ve representados los centroides iniciales para los 8 puntos representados.

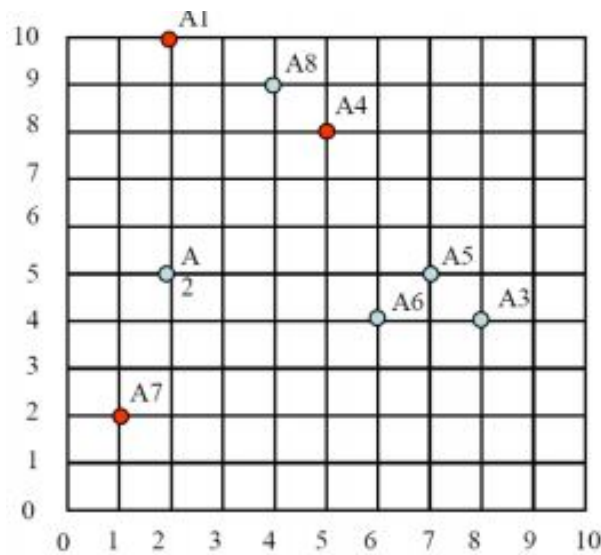


Figure 55: Plano inicial

Seguidamente, en la Figura 56 podemos observar cómo quedaría la agrupación de estos 8 puntos. Se crearían tres cluster cuyos centroides serían: A1(2, 10), A4(5, 8) y A7(1, 2)

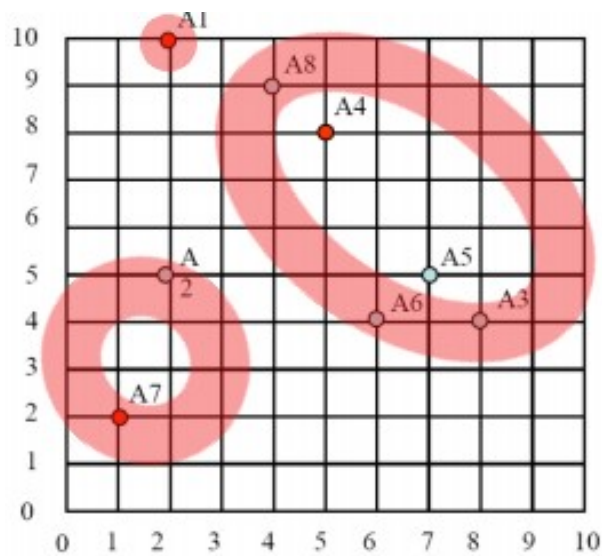


Figure 56: Primera agrupación

Para la primera iteración, hemos utilizado el cálculo de la media de las coordenadas, con esto, hemos conseguido calcular la nueva localización de los nuevos centroides. Como se puede observar en la Figura 57, los nuevos centroides serían: $A1(2, 10)$, $c2(6, 6)$ y $c3(1.5, 3.5)$.

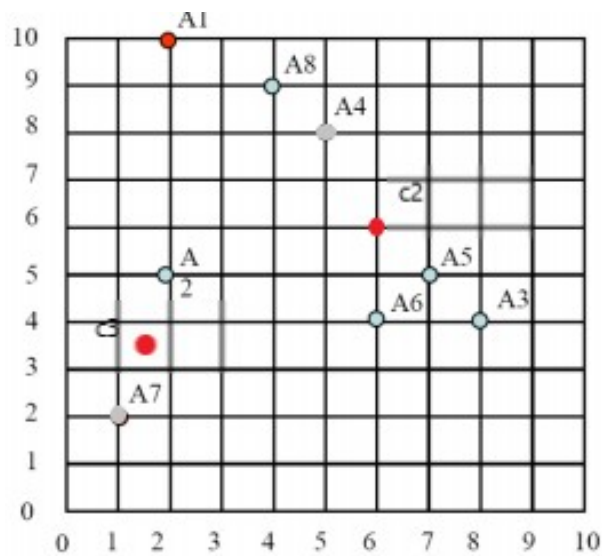


Figure 57: Primera iteración

En la Figura 58 podemos observar cómo quedaría la nueva agrupación de los cluster. Para ello, calculamos las distancias de los centroides con respecto a los puntos:

$$Dist(A1, A8) = 2.23$$

$$Dist(c2, A8) = 3.6$$

$$Dist(A1, A4) = 3.6$$

$$Dist(c2, A4) = 2.23$$

Como vemos, la distancia que existe entre A4 y c2 es menor que la existente entre A4 y A1, con lo que el punto A4 pertenecerá al cluster cuyo centroide es c2. De la misma manera ha sido calculado a qué cluster pertenece A8, resultando perteneciente a A1.

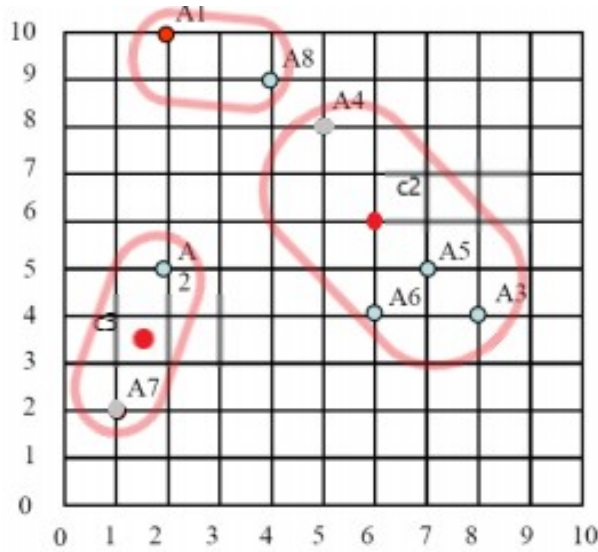


Figure 58: Segunda agrupación

La nueva relocalización de los centroides puede ser observada en la Figura 59. Como vemos, hemos tenido que añadir un nuevo centroide $c1$. La localización de estos nuevos centroides ha sido calculada de igual manera que antes, resultando ser: $c1(3, 9.5)$, $c2(6.5, 2.25)$ y $c3(1.5, 3.5)$.

Ahora, pasamos a calcular la distancia que existe entre los nuevos centroides y los puntos:

$$Dist(c2, A4) = 3.13$$

$$Dist(c1, A4) = 2.5$$

Como vemos, al contrario que en el primer agrupamiento, el punto $A4$ pasaría a pertenecer al cluster cuyo centroide es $c1$, ya que existe una menor distancia

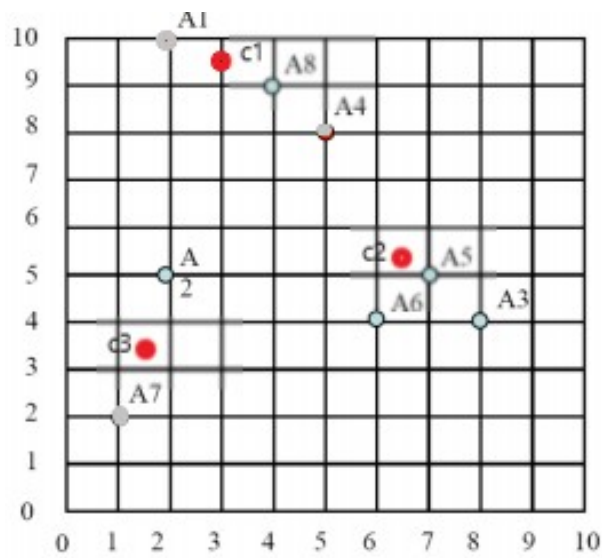


Figure 59: Segunda iteración

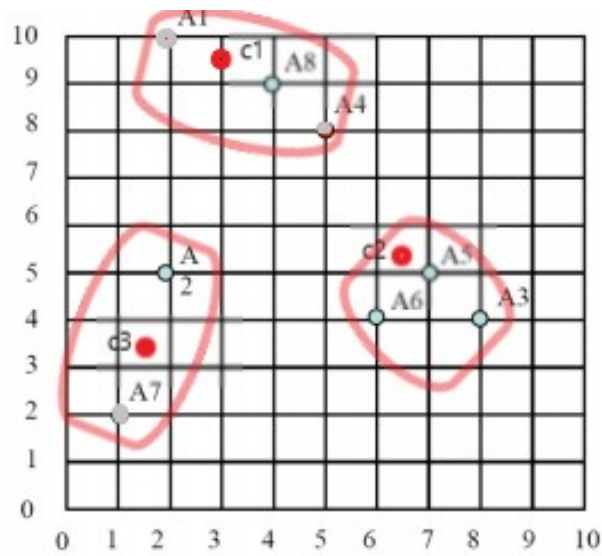


Figure 60: Segunda iteración

que con respecto a $c2$. El resultado de esta tercera agrupación es reflejada en la Figura 60.

Finalmente, volvemos a recalcular la localización de los cluster, siendo representada en la Figura 61. Como vemos, las coordenadas de los nuevos centroides serán: $c1(3.6, 9)$, $c2(7, 4.3)$ y $c3(1.5, 3.5)$. Como vemos, desde la segunda it-

eración, el centroide $c3$ no ha cambiado su posición.

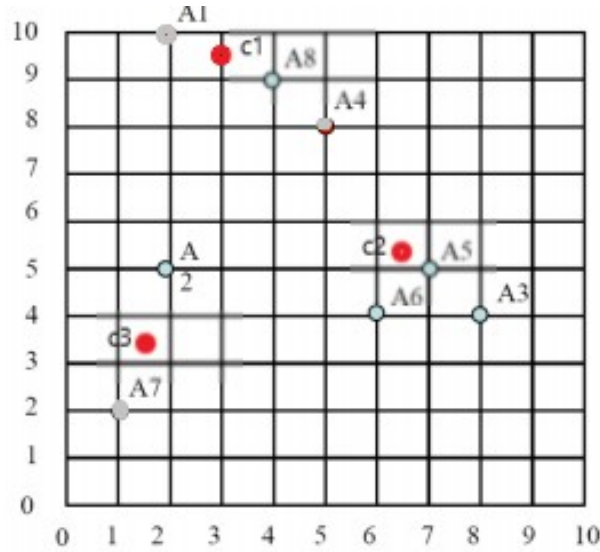


Figure 61: Tercera iteración

Como observamos en la Figura 62, al volver a realizar los calculos de las distancias, los clusters no se ven afectados, quedando exáctamente igual que en la Figura 60.

3.6 Ejercicio 6

Ejecute el algoritmo `HierarchicalClusterer` sobre su base de datos, usando la distancia Euclídea e indicando como parámetro que se corte el dendrograma en un número de clusters igual al número de clases de su problema, sin emplear la etiqueta de clase.

Para la realización de este ejercicio, los pasos a seguir son: primero, cargar la base de dato *Ecoli* en Weka. Segundo, ir al apartado *Cluster*. Una vez ahí, elegimos el algoritmo *HierarchicalClusterer*. Finalmente, como en todos los algoritmos, configuraremos éste tal y como el enunciado nos indica ($numClusters = 8$, $distanceFunction = EuclideanDistance$).

Seguidamente, pasaremos a analizar el porcentaje de instancias correctamente clasificadas dependiendo del *linkType* utilizado.

1. **SINGLE:** Según este tipo de enlace, las instancias incorrectamente clasificadas son 185, es decir un 55.06%.
2. **COMPLETE:** Para este enlace, las instancias incorrectamente clasificadas son 126, con lo que resulta ser un 37.5% del total.

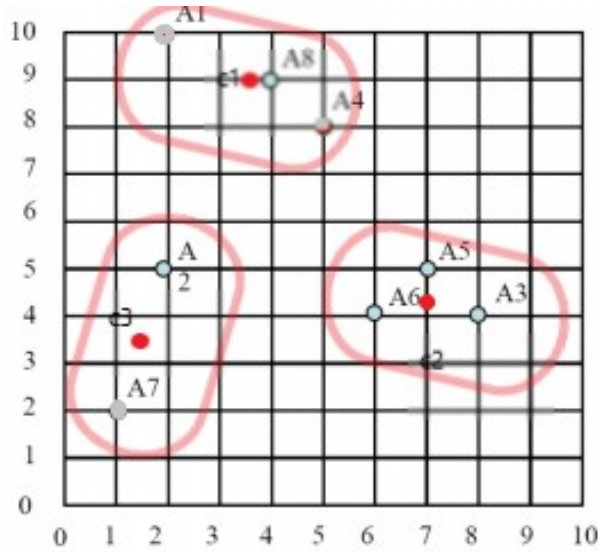


Figure 62: Tercera agrupación

3. **AVERAGE:** Aquí, las instancias incorrectamente clasificadas son 86, resultando ser un 25.6% de todas las instancias.
4. **CENTROID:** Para este último, el número de instancias incorrectamente clasificadas es 76, siendo éste un 22.62% del total.

Como observamos en la última enumeración, la mejor clasificación la obtenemos con el *linkType: CENTROID*, ya que de 336 instancias que constituyen el total de nuestra base de datos, sólo el 22.62% (76 instancias), han sido incorrectamente clasificadas. Este no es un buen resultado, no obstante, es mejor que el obtenido cuando usamos los *linkType: SINGLE, COMPLETE o AVERAGE*.

3.7 Ejercicio 7

Utilice el algoritmo de clustering jerárquico aglomerativo para agrupar los datos descritos por la siguiente matriz de distancias.

Para este ejercicio, vamos a dividirlo en dos partes según el tipo de enlazamiento utilizado, simple o complete.

1. Simple linkage

$$\begin{bmatrix} & A & B & C & D \\ A & 0 & 1 & 4 & 5 \\ B & & 0 & 2 & 6 \\ C & & & 0 & 3 \\ D & & & & 0 \end{bmatrix}$$

Como podemos observar en la matriz inicial, la menos distancia existente es entre A y B, luego con esto dos crearemos un cluster y posteriormente, recalcularemos la matriz de distancias.

$$\text{dist}((A, B), C) = \min(\text{dist}(A, C), \text{dist}(B, C)) = \min(4, 2) = 2$$

$$\text{dist}((A, B), D) = \min(\text{dist}(A, D), \text{dist}(B, D)) = \min(5, 6) = 5$$

Como vemos, las distancias entre cluster han cambiado, dejando como resultado la siguiente matriz de distancias:

$$\begin{bmatrix} & A, B & C & D \\ A, B & 0 & 2 & 5 \\ C & & 0 & 3 \\ D & & & 0 \end{bmatrix}$$

Seguidamente, como vemos en la anterior matriz de distancias, la menor distancia existente es entre (A,B) y C, con lo que crearíamos un nuevo cluster con ellos y posteriormente recalcularemos la matriz de distancias:

$$\text{dis}(((A, B), C), D) = \min(\text{dist}((A, B), D), \text{dist}(C, D)) = \min(5, 3) = 3$$

Con este cálculo, volvemos a realizar la matriz de distancias como sigue:

$$\begin{bmatrix} & A, B, C & D \\ A, B, C & 0 & 3 \\ D & & 0 \end{bmatrix}$$

Según la matriz de distancias anterior, la jerarquía obtenida por *Simple linkage* sería: $((A, B), C), D$. El dendograma obtenido con esta jerarquía se muestra en la Figura 63.

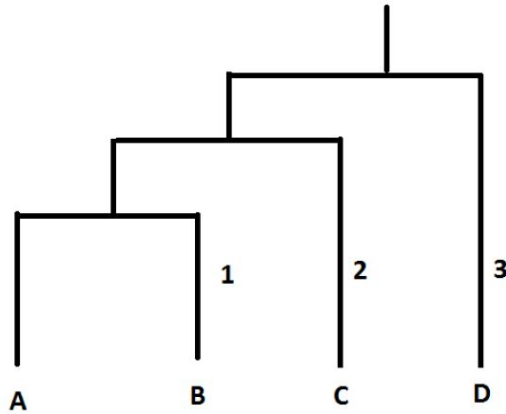


Figure 63: Dendograma obtenido usando *Simple linkage*

2. Complete linkage

$$\begin{bmatrix} & A & B & C & D \\ A & 0 & 1 & 4 & 5 \\ B & & 0 & 2 & 6 \\ C & & & 0 & 3 \\ D & & & & 0 \end{bmatrix}$$

A diferencia del apartado anterior, esta vez, cuando recalculemos las distancias, en vez de escoger el mínimo entre dos distancias, utilizaremos el máximo existente entre éstas.

Como vemos en la matriz de distancias inicial, la distancia menor existente es entre A y B, con lo que entre ellos crearíamos un nuevo cluster y recalcularemos las nuevas distancias:

$$dist((A, B), C) = \max(dist(A, C), dist(B, C)) = \max(4, 2) = 4$$

$$dist((A, B), D) = \max(dist(A, D), dist(B, D)) = \max(5, 6) = 6$$

Con los calculos ya obtenidos, procedemos a realizar nuestra nueva matriz de distancias:

$$\begin{bmatrix} & A, B & C & D \\ A, B & 0 & 4 & 6 \\ C & & 0 & 3 \\ D & & & 0 \end{bmatrix}$$

Como vemos en la matriz de distancias anterior, la menor distancia existente es entre C y D, lo que significa que crearíamos un nuevo cluster con estos dos y seguidamente recalculamos la matriz de distancias con el resultado obtenido.

$$dist((A, B), (C, D)) = \max(dist(A, C), dist(A, D), dist(B, C), dist(B, D)) =$$

$$\max(4, 5, 2, 6) = 6$$

Una vez recalculados, obtenemos nuestra matriz de distancias final, resultando ser la siguiente:

$$\begin{bmatrix} & A, B & D, C \\ A, B & 0 & 6 \\ D, C & & 0 \end{bmatrix}$$

Como vemos, la jerarquía resultante sería: $((A, B), (C, D))$. Dando como resultado el dendograma observable en la Figura 64.

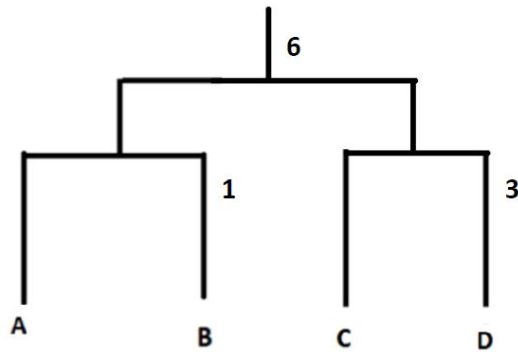


Figure 64: Dendrograma obtenido usando *Complete linkage*

3.8 Ejercicio 8

Ejecute el algoritmo `HierarchicalClusterer` con tipo de link completo y métrica de distancia euclídea, y visualice las gráficas de los puntos agrupados. Comparando en el eje X `instance.number` y el eje Y vaya variando y mostrando cada uno de los atributos (`sepal.length`, `sepal.width`, `petal.length`, `petal.width`). ¿Alguno de ellos produce unos grupos bien diferenciados y con fronteras claras?

Para la realización de este ejercicio, tenemos que cargar la base de datos *Iris*. Una vez cargada, nos vamos a la sección *Cluster*, ahí, seleccionamos el algoritmo *HierarchicalClusterer* y lo configuramos como nos dice el enunciado. Los resultados de éste pueden ser observados en las Figuras 65, 66, 67 y 68.

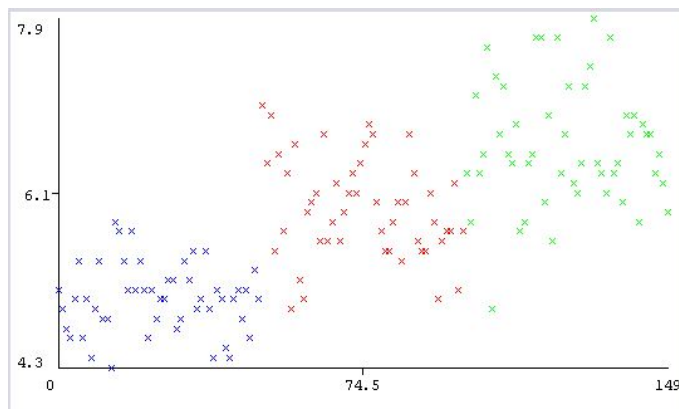


Figure 65: Gráfica obtenida comparando *Instance.number* y *Sepal.Length*

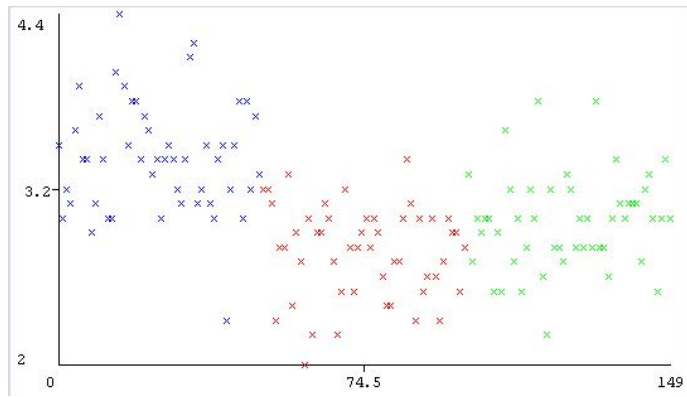


Figure 66: Gráfica obtenida comparando *Instance_number* y *SepalWidth*

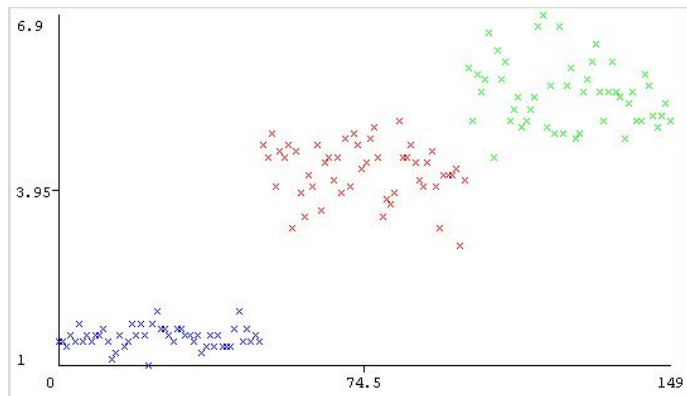


Figure 67: Gráfica obtenida comparando *Instance_number* y *PetalLength*

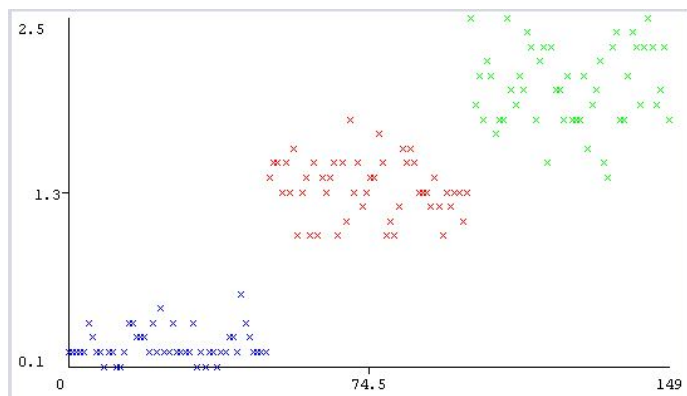


Figure 68: Gráfica obtenida comparando *Instance_number* y *PetalWidth*

Como podemos observar en las anteriores figuras, cuando lo comparamos con los atributos *PetalLength* y *PetalWidth*, los clusters son bien diferenciados ya que, como vemos, no hay patrones pertenecientes a una clase que estén dentro de otra, como se puede observar comparándolos con los otros dos atributos *SepalLength* y *SepalWidth*.

3.9 Ejercicio 9

Ejecute el algoritmo DBScan sin emplear la etiqueta de clase (“Classes to clusters evaluation”).

1. Realice una batería de pruebas cambiando los valores de los parámetros epsilon y minPoints. Se recomienda dejar uno fijo e ir variando el otro para ver la influencia. Use la distancia Euclídea como medida para encontrar vecinos en el radio Epsilon. Para la realización de este ejercicio vamos a comparar los resultados según el número de instancias agrupadas, instancias no agrupadas, instancias incorrectamente agrupadas y el número de clusters. Para ello, inicialmente vamos a ejecutar el algoritmo pero con la configuración por defecto.
 - (a) **Epsilon=0.9, minPoints=6:** 335 instancias agrupadas, 1 instancia no agrupada, 187 incorrectamente agrupadas y 2 clusters.
 - (b) **Epsilon=0.1, minPoints=5:** 71 instancias agrupadas, 265 instancias no agrupadas, 29 incorrectamente agrupadas y 4 clusters.
 - (c) **Epsilon=0.1, minPoints=6:** 53 instancias agrupadas, 283 instancias no agrupadas, 23 incorrectamente agrupadas y 4 clusters.

Como podemos observar, los resultados no son muy buenos, ya que cuando conseguimos más de 2 clusters, las instancias son incorrectamente agrupadas o directamente no son agrupadas. Estos resultados no se ven afectados aunque normalicemos estos valores.

Con respecto a la precisión de los clusters, podemos decir que con nuestra mejor configuración, es decir la (b), no conseguimos muy buena clasificación. Esto puede ser observado en la Figura 69, que representa la matriz de confianza obtenida en la segunda configuración.

```

      0  1  2  3  <-- assigned to cluster
35 23  4  0  | cp
  0  0  0  6  | im
Cluster 0 <-- cp      1  0  1  0  | pp
Cluster 1 <-- No class 0  0  0  1  | imU
Cluster 2 <-- pp      0  0  0  0  | om
Cluster 3 <-- im      0  0  0  0  | omL
                    0  0  0  0  | imL
                    0  0  0  0  | imS

```

Figure 69: Matriz de confianza obtenida con *Epsilo* = 0.1 y *minPoint* = 5

2. **Realice otra batería de pruebas usando otra función de distancia diferente a la distancia Euclídea, y compare los resultados de la mejor configuración obtenida en esta con la mejor de la batería anterior.**

Para la realizar este ejercicio, vamos a coger nuestra mejor configuración, es decir, $\epsilon = 0.1$ y $\text{minPoints} = 6$. Para ver cuál es la mejor configuración, vamos a ejecutar el algoritmo con las distancias, Euclídea, Minkowski y Chebyshev. Los parámetros en que nos basaremos serán los mismo que antes, instancias agrupadas, instancias no agrupadas, instancias incorrectamente agrupadas y número de clusters.

- (a) **Euclídea:** 71 instancias agrupadas, 265 no agrupadas, 29 incorrectamente agrupadas y 4 clusters.
- (b) **Minkowski:** 71 instancias agrupadas, 265 no agrupadas, 29 incorrectamente agrupadas y 4 clusters.
- (c) **Chebyshev:** 236 instancias agrupadas, 100 no agrupadas, 62 incorrectamente agrupadas y 3 clusters.

Como observamos, podemos observar que el número de instancias agrupadas es mayor utilizando la distancia de Chebyshev, con lo que podríamos decir que esta sería la mejor configuración. No obstante, es la que más instancias incorrectamente agrupadas tiene, con lo que no se podría decir que es la mejor configuración. En definitiva, podemos afirmar que cambiando el tipo de distancia utilizada, no hace que nuestro algoritmo clasifique mejor nuestras instancias.

4 Práctica 3: Árboles de Decisión y Redes Neuronales

4.1 Ejercicio 1

Cargue su base de datos y ejecute el algoritmo C4.5 usando un 75% para entrenar y un 25% para generalizar.

1. **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.**

Una vez cargada la base de datos *Ecoli*, nos tenemos que ir a la pestaña *Classify* de Weka. Una vez ahí, debemos de seleccionar el algoritmo J48, que es el correspondiente al algoritmo C4.5 en Weka. Dejaremos la configuración de éste por defecto, y para usar el 75% para entrenar, deberemos de seleccionar la opción *Percentage split* e igualarla a 75. Una vez hecho esto, procedemos a ejecutar este algoritmo.

Como se puede observar en la Figura 70, el nodo principal, o raíz, es el

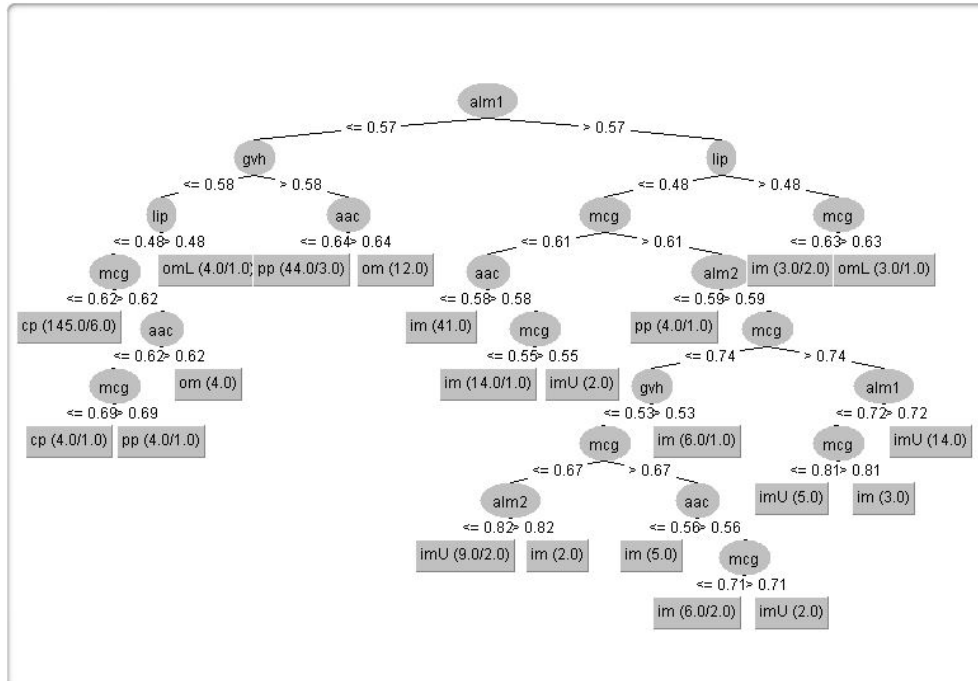


Figure 70: Árbol resultante al aplicar el algoritmo C4.5

atributo *aml1*. Además, el árbol cuenta con un total de 22 hojas y 21 nodos. La profundidad de éste es de 9. Junto con esta información, cabe destacar que hay dos variables omitidas: *imL* y *imS*, estando presente el resto de variables.

Con respecto a las métricas, hay que decir que son reflejadas sólo el 25% de éstas, es decir 84 instancias, de las cuales 69(82.14%) son clasificadas correctamente mientras que el resto, 15(17.86%) son clasificadas incorrectamente, siendo este un valor bastante bueno. Con respecto al parámetro Kappa, que representa la concordancia esperada utilizando el azar, es bastante alto, 0.7627, siendo éste un parámetro que siempre se intenta maximizar, con lo que podríamos decir que hemos obtenido una buena clasificación. Finalmente, los errores obtenidos son bastante bajos, aproximadamente 0.2 en todos, siendo estos parámetros unos que se intentan minimizar. El único error el cuál se sale de esta búsqueda es *Root relative squared error*, siendo este 64.36%.

2. **Analice y muestre cómo cambia el árbol (nodo principal, tamaño, número de hojas, variables y omitidas), al modificar los siguientes parámetros. Comente también los resultados de las métricas obtenidas**

Para la realización de este ejercicio, se van a ir variando los parámetros:

(a) **Unpruned=True, confidenceFactor=0.1, minNumObj=0, subtreeRaising=True:** Resultado observable en Figura 71 y 72.



```

Correctly Classified Instances      67          79.7619 %
Incorrectly Classified Instances   17          20.2381 %
Kappa statistic                    0.7352
Mean absolute error                0.0523
Root mean squared error            0.2239
Relative absolute error            27.8411 %
Root relative squared error        71.6366 %
Total Number of Instances         84

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,967    0,056    0,906     0,967    0,935     0,899    0,952    0,877    cp
      0,882    0,090    0,714     0,882    0,789     0,736    0,920    0,678    im
      0,611    0,015    0,917     0,611    0,733     0,699    0,786    0,644    pp
      0,667    0,027    0,750     0,667    0,706     0,674    0,819    0,517    imU
      0,714    0,039    0,625     0,714    0,667     0,636    0,838    0,470    om
      0,500    0,000    1,000     0,500    0,667     0,703    0,750    0,512    omL
      0,000    0,012    0,000     0,000    0,000     -0,012    0,494    0,012    imL
      ?       0,012    0,000     ?       ?       ?       ?       ?       imS
Weighted Avg.  0,798    0,047    0,821     0,798    0,798     0,762    0,876    0,695

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  <-- classified as
29  0  0  0  1  0  0  0  | a = cp
 0 15  0  2  0  0  0  0  | b = im
 3  2 11  0  2  0  0  0  | c = pp
 0  3  0  6  0  0  0  0  | d = imU
 0  0  1  0  5  0  0  1  | e = om
 0  0  0  0  0  1  1  0  | f = omL
 0  1  0  0  0  0  0  0  | g = imL
 0  0  0  0  0  0  0  0  | h = imS

```

Figure 72: Métricas resultantes

- (b) **Unpruned=True, confidenceFactor=0.5, minNumObj=0, subtreeRaising=True:** Resultado observable en Figura 73 y 74.


```

Correctly Classified Instances      67          79.7619 %
Incorrectly Classified Instances   17          20.2381 %
Kappa statistic                    0.7352
Mean absolute error                0.0523
Root mean squared error            0.2239
Relative absolute error            27.8411 %
Root relative squared error        71.6366 %
Total Number of Instances         84

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,967    0,056    0,906     0,967    0,935     0,899    0,952    0,877    cp
      0,882    0,090    0,714     0,882    0,789     0,736    0,920    0,678    im
      0,611    0,015    0,917     0,611    0,733     0,699    0,786    0,644    pp
      0,667    0,027    0,750     0,667    0,706     0,674    0,819    0,517    imU
      0,714    0,039    0,625     0,714    0,667     0,636    0,838    0,470    om
      0,500    0,000    1,000     0,500    0,667     0,703    0,750    0,512    omL
      0,000    0,012    0,000     0,000    0,000     -0,012    0,494    0,012    imL
      ?       0,012    0,000     ?        ?         ?        ?        ?        imS
Weighted Avg.  0,798    0,047    0,821     0,798    0,798     0,762    0,876    0,695

=== Confusion Matrix ===

 a  b  c  d  e  f  g  h  <-- classified as
29  0  0  0  1  0  0  0  | a = cp
 0 15  0  2  0  0  0  0  | b = im
 3  2 11  0  2  0  0  0  | c = pp
 0  3  0  6  0  0  0  0  | d = imU
 0  0  1  0  5  0  0  1  | e = om
 0  0  0  0  0  1  1  0  | f = omL
 0  1  0  0  0  0  0  0  | g = imL
 0  0  0  0  0  0  0  0  | h = imS

```

Figure 76: Métricas resultantes

- (d) **Unpruned=True, confidenceFactor=0.1, minNumObj=5, subtreeRaising=True:** Resultado observable en Figura 77 y 78.

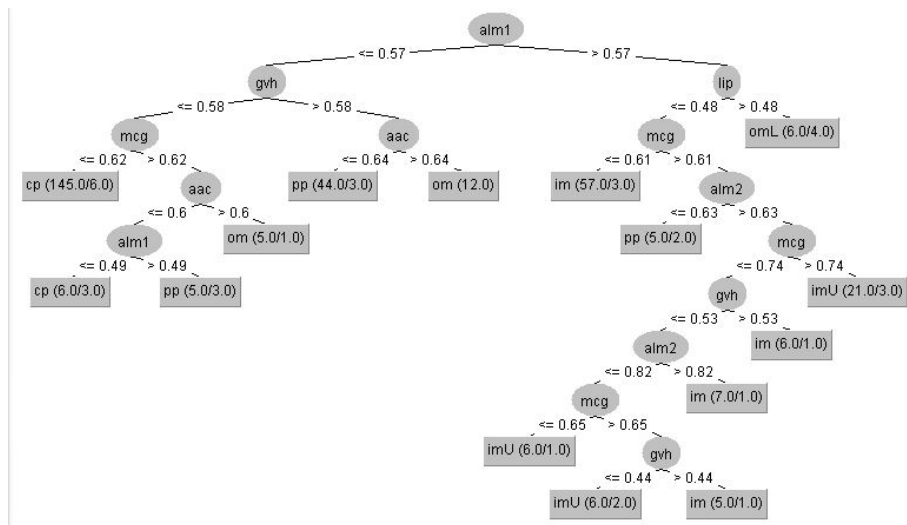


Figure 77: Árbol resultante

Correctly Classified Instances	70	83.3333 %
Incorrectly Classified Instances	14	16.6667 %
Kappa statistic	0.7798	
Mean absolute error	0.0542	
Root mean squared error	0.194	
Relative absolute error	28.8776 %	
Root relative squared error	62.0602 %	
Total Number of Instances	84	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,967	0,037	0,935	0,967	0,951	0,923	0,960	0,916	cp
	0,824	0,075	0,737	0,824	0,778	0,719	0,910	0,712	im
	0,833	0,061	0,789	0,833	0,811	0,758	0,919	0,819	pp
	0,667	0,027	0,750	0,667	0,706	0,674	0,822	0,558	imU
	0,857	0,013	0,857	0,857	0,857	0,844	0,978	0,856	om
	0,000	0,000	?	0,000	?	?	0,710	0,095	omL
	0,000	0,000	?	0,000	?	?	0,494	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,833	0,045	?	0,833	?	?	0,916	0,780	

=== Confusion Matrix ===

a	b	c	d	e	f	g	h	-- classified as
29	0	0	0	1	0	0	0	a = cp
0	14	1	2	0	0	0	0	b = im
2	1	15	0	0	0	0	0	c = pp
0	3	0	6	0	0	0	0	d = imU
0	0	1	0	6	0	0	0	e = om
0	0	2	0	0	0	0	0	f = omL
0	1	0	0	0	0	0	0	g = imL
0	0	0	0	0	0	0	0	h = imS

Figure 78: Métricas resultantes

(e) **Unpruned=True, confidenceFactor=0.1, minNumObj=10, subtreeRaising=True:** Resultado observable en Figura 79 y 80.

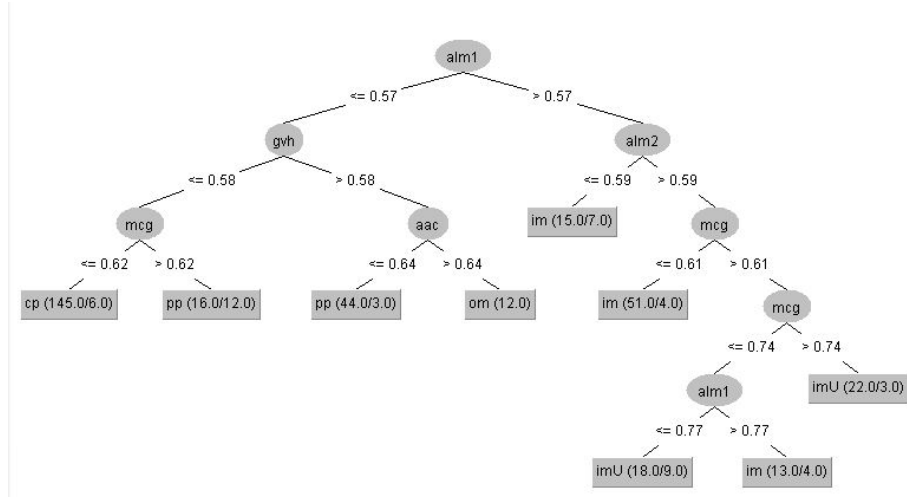


Figure 79: Árbol resultante

```

Correctly Classified Instances      67          79.7619 %
Incorrectly Classified Instances   17          20.2381 %
Kappa statistic                    0.7315
Mean absolute error                0.0594
Root mean squared error            0.1958
Relative absolute error            31.6165 %
Root relative squared error        62.6287 %
Total Number of Instances         84

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
0,967  0,037  0,935  0,967  0,951  0,923  0,980  0,933  cp
0,824  0,104  0,667  0,824  0,737  0,667  0,942  0,703  im
0,778  0,076  0,737  0,778  0,757  0,689  0,896  0,801  pp
0,667  0,040  0,667  0,667  0,667  0,627  0,823  0,598  imU
0,571  0,000  1,000  0,571  0,727  0,742  0,904  0,726  om
0,000  0,000  ?  0,000  ?  ?  0,963  0,250  omL
0,000  0,000  ?  0,000  ?  ?  0,476  0,012  imL
?  0,000  ?  ?  ?  ?  ?  ?  imS
Weighted Avg.  0,798  0,055  ?  0,798  ?  ?  0,925  0,778

=== Confusion Matrix ===

 a  b  c  d  e  f  g  h  <-- classified as
29  0  1  0  0  0  0  0  | a = cp
 0 14  0  3  0  0  0  0  | b = im
 2  2 14  0  0  0  0  0  | c = pp
 0  3  0  6  0  0  0  0  | d = imU
 0  0  3  0  4  0  0  0  | e = om
 0  1  1  0  0  0  0  0  | f = omL
 0  1  0  0  0  0  0  0  | g = imL
 0  0  0  0  0  0  0  0  | h = imS

```

Figure 80: Métricas resultantes

- (f) **Unpruned=True, confidenceFactor=0.1, minNumObj=0, subtreeRaising=False:** Resultado observable en Figura 81 y 82.

(g) **Unpruned=False, confidenceFactor=0.1, minNumObj=0, subtreeRaising=True:** Resultado observable en Figura 83 y 84.

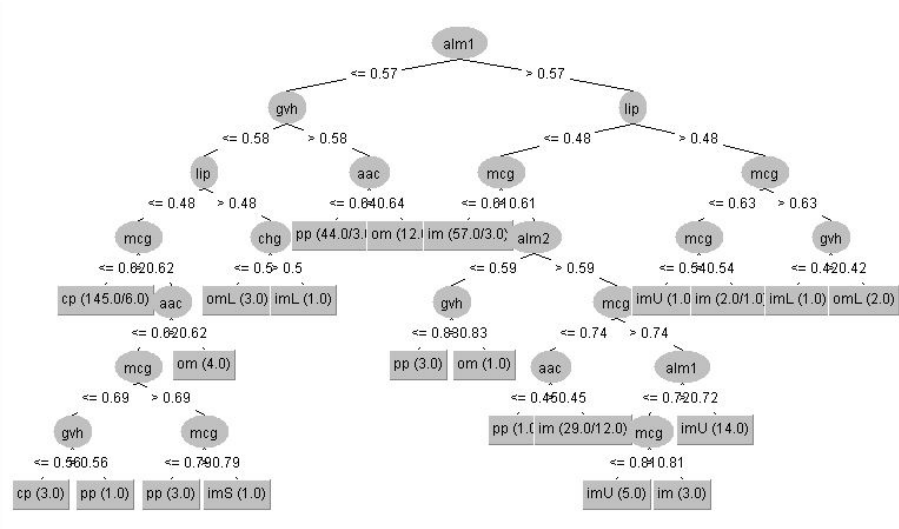


Figure 83: Árbol resultante

```

Correctly Classified Instances      68           80.9524 %
Incorrectly Classified Instances    16           19.0476 %
Kappa statistic                    0.7485
Mean absolute error                 0.0627
Root mean squared error             0.203
Relative absolute error             33.4085 %
Root relative squared error         64.9359 %
Total Number of Instances          84

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0,967    0,037    0,935     0,967    0,951     0,923    0,960     0,916    cp
      0,941    0,134    0,640     0,941    0,762     0,709    0,927     0,636    im
      0,778    0,015    0,933     0,778    0,848     0,817    0,908     0,799    pp
      0,444    0,013    0,800     0,444    0,571     0,564    0,916     0,522    imU
      0,714    0,013    0,833     0,714    0,769     0,753    0,872     0,648    om
      0,000    0,000    ?         0,000    ?         ?        0,604     0,032    omL
      0,000    0,012    0,000     0,000    0,000     -0,012    0,494     0,012    imL
      ?        0,012    0,000     ?         ?         ?        ?         ?        imS
Weighted Avg.    0,810    0,046    ?         0,810    ?         ?        0,916     0,738

=== Confusion Matrix ===

 a  b  c  d  e  f  g  h  <-- classified as
29  0  0  0  1  0  0  0  | a = cp
 0 16  0  1  0  0  0  0  | b = im
 2  2 14  0  0  0  0  0  | c = pp
 0  5  0  4  0  0  0  0  | d = imU
 0  0  1  0  5  0  0  1  | e = om
 0  1  0  0  0  0  1  0  | f = omL
 0  1  0  0  0  0  0  0  | g = imL
 0  0  0  0  0  0  0  0  | h = imS

```

Figure 84: Métricas resultantes

- (h) **Unpruned=False, confidenceFactor=0.1, minNumObj=5, subtreeRaising=True:** Resultado observable en Figura 85 y 86.

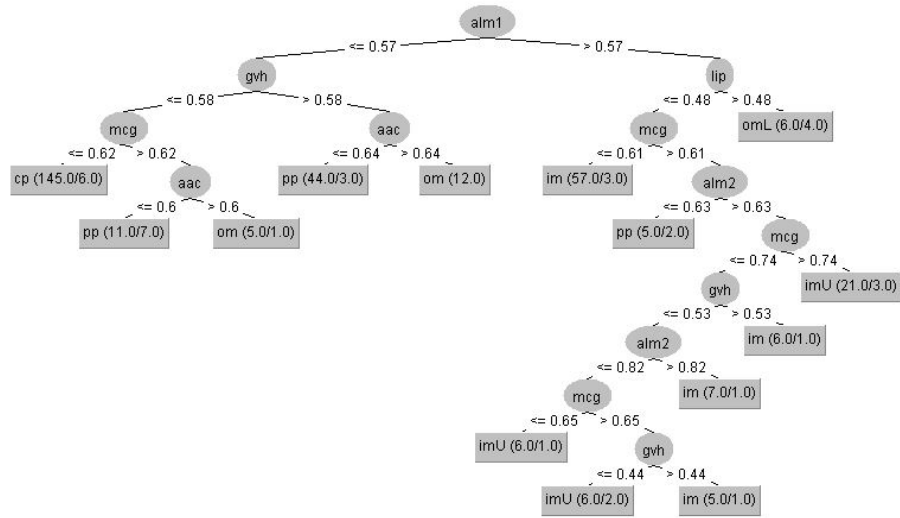


Figure 85: Árbol resultante

- (i) **Unpruned=False, confidenceFactor=0.1, minNumObj=10, subtreeRaising=True:** Resultado observable en Figura 87 y 88.

```

Correctly Classified Instances      69           82.1429 %
Incorrectly Classified Instances   15           17.8571 %
Kappa statistic                    0.7631
Mean absolute error                 0.0633
Root mean squared error             0.1951
Relative absolute error             33.7256 %
Root relative squared error         62.4348 %
Total Number of Instances          84

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
0,967    0,037    0,935    0,967    0,951    0,923    0,960    0,916    cp
0,941    0,134    0,640    0,941    0,762    0,709    0,927    0,636    im
0,778    0,030    0,875    0,778    0,824    0,781    0,905    0,797    pp
0,444    0,013    0,800    0,444    0,571    0,564    0,916    0,522    imU
0,857    0,013    0,857    0,857    0,857    0,844    0,983    0,862    om
0,000    0,000    ?        0,000    ?        ?        0,576    0,030    omL
0,000    0,000    ?        0,000    ?        ?        0,494    0,012    imL
?        0,000    ?        ?        ?        ?        ?        ?        imS
Weighted Avg.  0,821    0,049    ?        0,821    ?        ?        0,924    0,755

=== Confusion Matrix ===

 a  b  c  d  e  f  g  h  <-- classified as
29  0  0  0  1  0  0  0  | a = cp
 0 16  0  1  0  0  0  0  | b = im
 2  2 14  0  0  0  0  0  | c = pp
 0  5  0  4  0  0  0  0  | d = imU
 0  0  1  0  6  0  0  0  | e = om
 0  1  1  0  0  0  0  0  | f = omL
 0  1  0  0  0  0  0  0  | g = imL
 0  0  0  0  0  0  0  0  | h = imS

```

Figure 86: Métricas resultantes

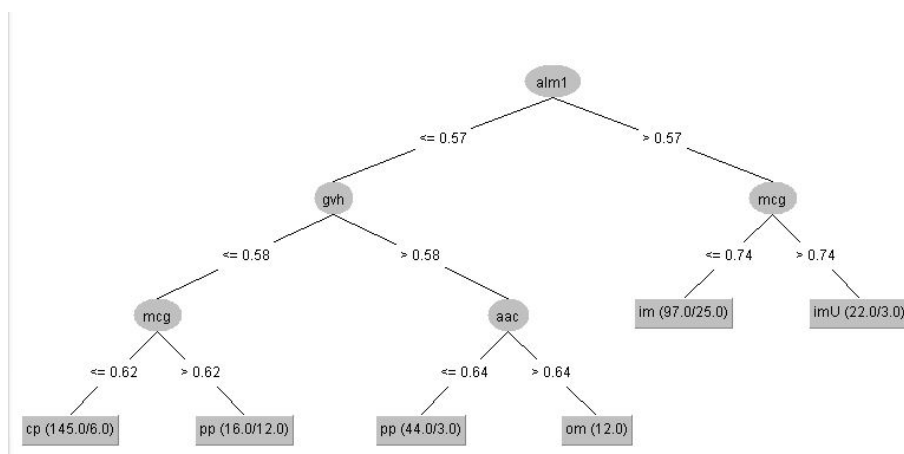


Figure 87: Árbol resultante

Como conclusión, podemos decir, observando el árbol y métricas resultantes, que la mejor configuración obtenida es con `unpruned=True`,

```

Correctly Classified Instances      67           79.7619 %
Incorrectly Classified Instances   17           20.2381 %
Kappa statistic                    0.7299
Mean absolute error                 0.0656
Root mean squared error             0.1958
Relative absolute error             34.9339 %
Root relative squared error         62.6288 %
Total Number of Instances         84

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
0,967    0,037    0,935      0,967    0,951      0,923    0,980    0,933    cp
0,941    0,134    0,640      0,941    0,762      0,709    0,927    0,636    im
0,778    0,076    0,737      0,778    0,757      0,689    0,881    0,788    pp
0,444    0,013    0,800      0,444    0,571      0,564    0,916    0,522    imU
0,571    0,000    1,000      0,571    0,727      0,742    0,900    0,798    om
0,000    0,000    ?          0,000    ?          ?        0,899    0,159    omL
0,000    0,000    ?          0,000    ?          ?        0,476    0,012    imL
?        0,000    ?          ?        ?          ?        ?        ?        imS
Weighted Avg.    0,798    0,058    ?          0,798    ?          ?        0,927    0,757

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  <-- classified as
29  0  1  0  0  0  0  0 | a = cp
 0 16  0  1  0  0  0  0 | b = im
 2  2 14  0  0  0  0  0 | c = pp
 0  5  0  4  0  0  0  0 | d = imU
 0  0  3  0  4  0  0  0 | e = om
 0  1  1  0  0  0  0  0 | f = omL
 0  1  0  0  0  0  0  0 | g = imL
 0  0  0  0  0  0  0  0 | h = imS

```

Figure 88: Métricas resultantes

minNumObj=5, subtreeRaising=True y confidenceFactor=0.1. Podemos decir esto ya que conseguimos un 83% de las instancias correctamente clasificadas, dejando esta configuración como la que más instancias correctamente clasificadas obtiene.

4.2 Ejercicio 2

Cargue su base de datos y ejecute el algoritmo LMT (con un 75/25 %) con los parámetros por defecto. Analice y comente los resultados obtenidos, los modelos logísticos obtenidos para cada clase y el árbol. En caso de obtener solo una hoja modifique el valor del parámetro numBoostingIterations.

Para ejecutar este algoritmo sobre nuestra base de datos, hemos dejado la configuración por defecto menos el parámetro numBoostingIterations, el cuál lo hemos igualado a 2. Como antes, para utilizar el 75/25%, hay que seleccionar el apartado *Porcentaje split* e igualarlo a 75. Una vez hecho esto, procedemos a analizar los modelos logísticos obtenidos como resultado. En las Figuras 89 y 90 podemos observar el resultado de dicho algoritmo. Lo que se ve, es el peso de cada atributo en cada clase, según el modelo en el que estén. Por ejemplo, podemos observar que el atributo *chg* tiene un gran peso en la clase *imL* dentro

del modelo *LM_1*.

```
LM_1:
Class cp :
10.71 +
[mcg] * -4.66 +
[gvh] * -7.34 +
[alml] * -8.8 +
[alm2] * 2.63

Class im :
-0.64 +
[mcg] * -5.48 +
[gvh] * 6.67 +
[aac] * -8.12 +
[alml] * 6

Class pp :
-2.03 +
[mcg] * 3.73 +
[gvh] * 0.04 +
[aac] * -1.21 +
[alml] * 2.24

Class imU :
1.38 +
[mcg] * 4.84 +
[aac] * -22.45 +
[alm2] * 3.92

Class om :
-25.12 +
[gvh] * 6.52 +
[aac] * 36.57

Class omL :
-17.3 +
[lip] * 17.63 +
[alml] * 9.12

Class imL :
-18.71 +
[lip] * 4.26 +
[chg] * 22.52

Class imS :
-29.82 +
[mcg] * 39.31

LM_2:
Class cp :
7.01 +
[mcg] * -4.66 +
[gvh] * 2.04 +
[alml] * -15.05

Class im :
-2.47 +
[mcg] * -5.18 +
[aac] * -1.38 +
[alml] * 6

Class pp :
0.28 +
[mcg] * 3.73 +
[gvh] * -2.41 +
[aac] * -4.87 +
[alml] * 10.26

Class imU :
-4.17 +
[mcg] * 4.84 +
[gvh] * 0.06 +
[aac] * -8.73 +
[alm2] * 3.92

Class om :
-22.82 +
[gvh] * 14.27 +
[aac] * 24.16

Class omL :
-12.56 +
[mcg] * 0 +
[gvh] * 0.01 +
[lip] * 14.85

Class imL :
-15.05 +
[mcg] * 0 +
[gvh] * 0.01 +
[lip] * 4.26 +
[chg] * 15.17

Class imS :
-11.63 +
[mcg] * 12.06 +
[gvh] * 0.05
```

Figure 89: Métricas y árbol resultante

<pre> LM_3: Class cp : 2.63 + [mcg] * -4.21 + [alm1] * -6.39 Class im : 0.05 + [mcg] * -6.35 + [gvh] * 1.59 + [alm1] * 6 + [alm2] * 1.33 Class pp : -3.85 + [mcg] * 3.06 + [gvh] * 11.16 + [aac] * -4.87 + [alm2] * -3.69 Class imU : -5.77 + [mcg] * 7.27 + [gvh] * -3.24 + [aac] * 2.11 + [alm2] * 3.92 Class om : -15.81 + [gvh] * 6.52 + [lip] * 3.25 + [aac] * 16.9 + [alm2] * -3.85 Class omL : -4.47 + [lip] * 7.82 + [alm2] * -6.64 Class imL : -18.89 + [lip] * 7.42 + [chg] * 7.19 + [alm1] * 9.9 Class imS : -6.25 + [mcg] * 1.41 + [aac] * -3.82 + [alm2] * 7.64 </pre>	<pre> LM_4: Class cp : 2.69 + [mcg] * -4.19 + [alm1] * -6.49 Class im : -11.4 + [mcg] * 1.01 + [gvh] * 1.59 + [alm1] * 16.03 Class pp : -4.88 + [mcg] * 0.09 + [gvh] * 11.03 + [aac] * -4.87 + [alm2] * -1.53 Class imU : 16.19 + [mcg] * -28.82 + [gvh] * -2.47 + [alm2] * 3.92 Class om : -26.39 + [gvh] * 6.52 + [lip] * 3.25 + [aac] * 30.8 Class omL : 0.9 + [lip] * 7.82 + [alm2] * -14.35 Class imL : -3.4 + [gvh] * -28.95 + [lip] * 7.42 + [chg] * 7.19 + [alm1] * 9.95 Class imS : -7.99 + [mcg] * 1.41 + [alm1] * -0.04 + [alm2] * 4.72 </pre>
--	--

Figure 90: Métricas y árbol resultante

Con respecto al árbol resultante, que puede verse en la Figura 92, vemos que es mucho más simple. Además, como antes, el nodo raíz es el atributo *alm1*, también, en cada hoja vemos el número de instancias que pertenecen a cada modelo(p.e. 161 en el modelo *LM_1*).

Finalmente, las métricas obtenidas son las siguientes:

1. 70 instancias correctamente clasificadas (83.33%).
2. 14 instancias incorrectamente clasificadas (16.66%).
3. 0.7796 en el estadístico Kappa.
4. 0.0549 *Mean absolute error*.
5. 0.1678 *Root mean squared error*.

6. 29.27% *Relative absolute error.*
7. 53.67% *Root relative squared error.*

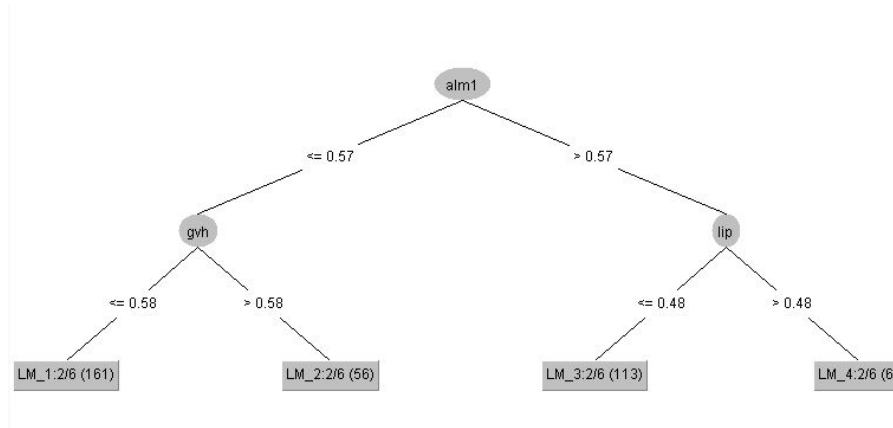


Figure 91: Árbol resultante

4.3 Ejercicio 3

Sobre el algoritmo **RandomForest** realice una labor de investigación.

1. **Explique con sus palabras en que se basa este árbol de decisión.**
Este árbol de decisión se basa en la media obtenida al combinar un conjunto de árboles que previamente son obtenidos variando las entradas de los mimos. Cada árbol tendrá un valor diferente, los cuales serán combinados y el resultado de este algoritmo será la media de dichos árboles combinados.
2. **Ejecútelo sobre su base de datos y exponga los resultados obtenidos.**

```

=== Summary ===

Correctly Classified Instances      72           85.7143 %
Incorrectly Classified Instances    12           14.2857 %
Kappa statistic                    0.8107
Mean absolute error                 0.0579
Root mean squared error             0.1701
Relative absolute error             30.8475 %
Root relative squared error         54.4092 %
Total Number of Instances          84

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              1,000    0,037    0,938      1,000    0,968      0,950    0,990    0,974    cp
              0,882    0,090    0,714      0,882    0,789      0,736    0,966    0,883    im
              0,833    0,030    0,882      0,833    0,857      0,820    0,962    0,915    pp
              0,556    0,027    0,714      0,556    0,625      0,592    0,971    0,776    imU
              0,857    0,000    1,000      0,857    0,923      0,920    0,998    0,968    om
              0,500    0,000    1,000      0,500    0,667      0,703    0,994    0,833    omL
              0,000    0,000    ?           0,000    ?           ?        0,440    0,012    imL
              ?        0,000    ?           ?         ?           ?        ?        ?        imS
Weighted Avg.    0,857    0,041    ?           0,857    ?           ?        0,971    0,907

=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  <-- classified as
30  0  0  0  0  0  0  0 | a = cp
 0 15  0  2  0  0  0  0 | b = im
 2  1 15  0  0  0  0  0 | c = pp
 0  4  0  5  0  0  0  0 | d = imU
 0  0  1  0  6  0  0  0 | e = om
 0  0  1  0  0  1  0  0 | f = omL
 0  1  0  0  0  0  0  0 | g = imL
 0  0  0  0  0  0  0  0 | h = imS

```

Figure 92: Métricas obtenidas al aplicar el algoritmo *RandomForest*

4.4 Ejercicio 4

Utilizando la base de datos “Iris” (Moodle) con un 75/25 %, ejecute el algoritmo MultilayerPerceptron con los parámetros por defecto.

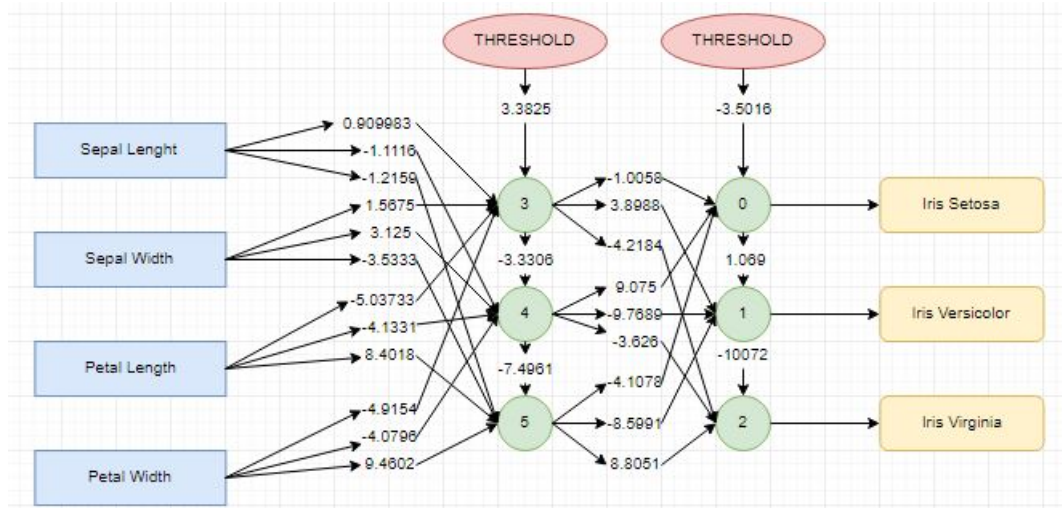


Figure 93: Diagrama de la red neural multicapa para la base de datos *Iris*

4.5 Ejercicio 5

Utilizando su base de datos con un 75/25% y el algoritmo MultilayerPerceptron.

1. ¿Cuál sería el valor por defecto para el atributo `hiddenLayers` en su base de datos?

Como ya sabemos, el número de capas por defecto corresponde a la siguiente fórmula:

$$a = \frac{\text{numAtributos} + \text{numClases}}{2}$$

Si sustituímos los datos de nuestra base de datos en la fórmula (8 atributos y 8 clases), nos daría un total 8 nodos por cada capa. Que, ejecutando el algoritmo sobre nuestra base de datos crearía dos capas, la primera con 7 nodos, la cual recibe los *inputs* de la red, y una segunda capa con 8 nodos, la cual decide a qué clase pertenece nuestra entrada(*outputs*).

2. Con los valores por defecto, ¿qué observa al ir modificando el `learningRate`? Use tablas para mostrar los resultados.

El parámetro *Learning Rate*, hace referencia a las variaciones que se van haciendo a los pesos dentro de nuestra red multicapa. Estos cambios se pueden comprobar con la siguiente fórmula:

$$\text{new_Weight} = \text{old_Weight} - \text{learningRate} * \text{gradient}$$

A continuación, podremos ver cómo varían las métricas resultantes de nuestra red cuando variamos el valor de *learningRate*.

LearningRate	CCI	Kappa	MAE	RMSE	RAE	RRSE
0.1	85.71%	0.8116	0.0575	0.1704	30.6004%	54.5154%
0.25	84.5238%	0.7956	0.054	0.1792	28.751%	57.3291%
0.5	80.9524%	0.7473	0.0516	0.1867	27.4912%	59.7467%
0.75	82.1429%	0.7631	0.0519	0.1855	27.6569%	59.3407%

Table 1: Métricas al aplicar MLP a la base de datos *Ecoli*

Como podemos observar en la Tabla 1, el mejor resultado lo obtenemos con el menor valor de Learning rate, 0.1, el cuál, como no vemos en la fórmula descrita anteriormente, en este caso, un valor muy pequeño para el ratio de aprendizaje, hará que nuestra neurona sea capaz de aprender con mayor facilidad. En el caso de tener un ratio de aprendizaje mayor, lo que hará es crear más ruido, haciendo que a nuestra neurona le resulte más complicado aprender.

3. Con los valores por defecto, ¿qué observa al ir modificando el momentum?. Use tablas para mostrar los resultados.

Como sabemos por la teoría, el parámetro *momentum*, se encarga de ajustar los enlaces entre los nodos de las capas de nuestra red neuronal. Esto ayuda a que nuestra red no se quede atascada en valores óptimos locales, sino que siga con el aprendizaje hasta cerciorarse de que el valor óptimo encontrado es global.

A continuación, como en el ejercicio anterior, vamos a mostrar una tabla en la cuál vamos a ir indicando las métricas obtenidas para distintos valores del parámetro *Momentum*.

Momentum	CCI	Kappa	MAE	RMSE	RAE	RRSE
0.1	85.7143%	0.8116	0.0601	0.17	32.0246%	54.4051%
0.25	85.7143%	0.8116	0.0564	0.1709	30.0568%	54.6695%
0.5	85.7143%	0.8116	0.0544	0.1751	28.9491%	56.035%
0.75	83.333%	0.7792	0.0525	0.187	27.9808%	59.833%

Table 2: Métricas al aplicar MLP a la base de datos *Ecoli*

A diferencia del ejercicio anterior, obtenemos mejores resultados cuando el parámetro *Momentum* no es mínimo. Esta vez, obtenemos mejores resultado con un valor de 0.25. Tampoco es que se diferencie mucho con el menor valor, que es el segundo que mejores resultados obtiene, pero aún así, podemos decir que el valor óptimo para el Momentum, con un ratio de aprendizaje de 0.1, es de 0.25.

4. Con los valores por defecto, realice una gráfica que muestre cómo cambia el valor de **Correctly Classified instances** al modificar el parámetro **trainingTime**.

Como podemos observar en la Figura 94, el porcentaje máximo de instancias correctamente clasificadas se alcanza con un **trainingTime** de 50, a partir de ahí, el porcentaje se mantiene hasta que, para un **trainin** de más de 2000, empieza a disminuir. Al principio, el aumento es muy significativo, esto es debido a que cuanto más tiempo tengamos para entrenar los datos, mejor clasificación habrá, pero llegado a un punto, esto es inútil y los datos bien clasificados empiezan a disminuir de manera menos significativa.

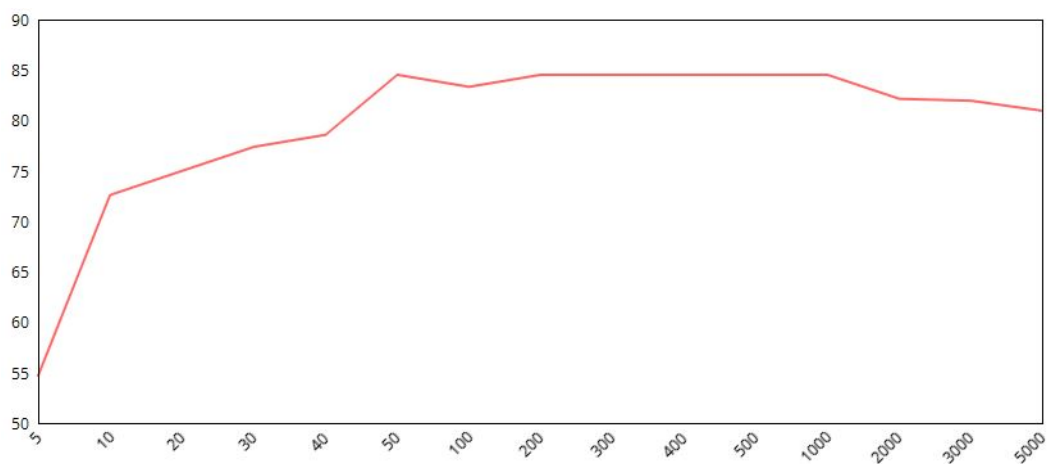


Figure 94: Gráfico comparativo entre CCI y trainingTime

5. Modifique como estime oportuno los parámetros por defecto del **MultilayerPerceptron** y comente los resultados de la mejor configuración obtenida.

Basándonos en los datos que hemos recogido hasta ahora, cabe destacar que hemos realizado el experimento cambiando los valores de *learning rate* y *momentum*. También, hemos declarado el training time a 500, ya que, como hemos visto en la figura 94, es cuando se estabiliza el porcentaje de instancias correctamente clasificadas. Una vez aclarado esto, y tras haber realizado varios cambios en los parámetros comentados anteriormente, nos damos cuenta que la mejor configuración se consigue con *learninRate* = 0.1 y *momentum* = 0.2. Con estos parámetros conseguimos las siguientes métricas:

- (a) Un CCI de 85.7143%.
- (b) Kappa statistic de 0.8116.
- (c) MAE de 0.0575.

- (d) RMSE de 0.1704.
- (e) RAE de 30.6004%.
- (f) RRSE de 54.5154%.

Como hemos comentado hasta ahora, hemos visto que a un mayor valor de learning rate, hace que nuestra red cree más ruido, con lo que, para ésta, es más difícil de aprender. También, realizando estos experimentos, nos hemos dado cuenta de que el parámetro momentum no debe de ser muy alto, pero tampoco bajo, ya que esto haría que nuestra red cayera en un óptimo local en vez de global. Por último, hemos visto que cuando cambiamos dichos parámetros, hay veces que algunas métricas bajan y otras suben, pero lo que nos interesa, es tener el mayor número de instancias correctamente clasificadas, pero a la vez, no tener un error muy alto, con lo que finalmente podemos decir que esta es la configuración más óptima.

5 Práctica 4: Preprocesamiento

Para la realización de esta práctica, primero hemos tenido que descargarnos los datos *train* y *test* de la plataforma Kaggle. Una vez descargados, con ayuda de un editor de texto, hemos completado los atributos que faltaban y reorganizado dicha base de datos. Una vez hecho esto, hemos cargado los archivos *.csv* en Weka y finalmente los hemos guardado como archivos *.arff*. Una vez completado esto, comenzaremos a procesar los datos.

Para poder obtener un buen resultado, primero hemos normalizado el archivo *test.arff* para poder tener todas las instancias en una misma escala (0,1). Seguidamente, con el archivo *train.arff*, haremos lo mismo, pero además, le aplicaremos más filtros aparte del filtro no supervisado *Normalize*.

Viendo las métricas de los datos, nos damos cuenta de que el atributo *MWD*, tiene un 100% de Missing values, con lo que, viendo que no influye, decidimos eliminarlo. Seguidamente, nos damos cuenta de que hay varios atributos cuyo porcentaje de Missing values es bastante elevado, un ejemplo es el atributo *ATMP*, con lo que decidimos aplicar el filtro no supervisado *ReplaceMissing-Values*, para que así dichos valores perdidos sean reemplazados por la moda y media de nuestro dataset. Una vez hecho este preprocesamiento, y viendo que ningún otro filtro mejoraría nuestro conjunto de datos, procedemos a la clasificación.

Habiendo realizado varios intentos, nuestro mejor F-Measure ha sido de 0.5629. Este valor lo hemos conseguido aplicando el algoritmo de clasificación RandomForest, en el cual hemos dejado los valores por defecto. Este valor ha sido intentado mejorar con otros algoritmos, como por ejemplo MLP con los valores por defecto y cambiándolos. Además, se ha intentado mejorar con el algoritmo C4.5, el cual nos ha dado uno de los peores valores. Finalmente, hemos

concluido con que el mejor algoritmo clasificador para este conjunto de datos es RandomForest.

6 Bibliografía:

1. Temario de la asignatura en Moodle (<http://moodle.uco.es/m1718/course/view.php?id=991>).
2. Documentación de Weka (https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf).
3. Información sobre randomForest (https://es.wikipedia.org/wiki/Random_forest).
4. ShareLatex documentation (<https://es.sharelatex.com/learn/>).
5. Base de datos Ecoli (<http://archive.ics.uci.edu/ml/datasets/Ecoli>).
6. Información dentro de la configuración de los distintos algoritmos usados en la realización de estas prácticas.
7. Base de datos para Kaggle e información de cómo organizarla. (<https://www.kaggle.com/c/competition-iaa-1819/data>).