



INTRODUCCIÓN AL APRENDIZAJE AUTOMÁTICO

Base de datos: Ecoli

3º Grado en Ingeniería Informática,

Segundo Cuatrimestre,

Curso 2017-2018

UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

Jose Luis Arcos García-Verdugo

45946278Y

i32argaj@uco.es

Córdoba, a 27 de junio de 2018

ÍNDICE

1. Práctica 1: Introducción a Weka.....	4
1.1. Ejercicio 1.....	4
1.2. Ejercicio 2.....	8
1.3. Ejercicio 3.	10
1.4. Ejercicio 4.	12
1.5. Ejercicio 5.	17
1.6. Ejercicio 6.	25
1.7. Ejercicio 7.	31
 2. Práctica 2-1: Regresión y clasificación con Weka.....	33
2.1. Ejercicio 1.	33
2.2. Ejercicio 2.	41
2.3. Ejercicio 3.	44
2.4. Ejercicio 4.	48
 3. Práctica 2-2: Clustering con Weka.....	53
3.1. Ejercicio 1.	53
3.2. Ejercicio 2.	60
3.3. Ejercicio 3.	62
3.4. Ejercicio 4.	62
3.5. Ejercicio 5.	64
3.6. Ejercicio 6.	68
3.7. Ejercicio 7.	70
3.8. Ejercicio 8.	75
3.9. Ejercicio 9.	78

4. Práctica 3: Árboles de decisión y redes neuronales.....	81
4.1. Ejercicio 1.	81
4.2. Ejercicio 2.	96
4.3. Ejercicio 3.	99
4.4. Ejercicio 4.	100
4.5. Ejercicio 5.	101
 5. Práctica 4: Competición con la plataforma Kaggle.....	107
 6. Referencias bibliográficas.....	108

1. Práctica 1: Introducción a Weka

1.1. Ejercicio 1: Cargue la base de datos Iris. Observe los atributos.

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

Tras cargar la base de datos Iris podemos observar la información que nos muestra Weka, en ella por ejemplo se detalla el número de atributos, que en este caso es de 5. Sin embargo, los atributos que caracterizan a las instancias serían 4, ya que el último es el atributo que define a qué clases pertenece cada instancia.

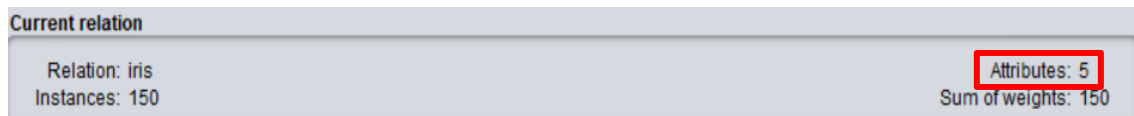


Figura 1.1: Atributos de la base de datos Iris

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

El último atributo del dataset es la clase a la que pertenece cada instancia luego estamos ante un problema de clasificación en la que, gracias a los otros atributos, podemos llegar a la conclusión de a qué clase pertenece cada instancia.

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

Todos estos valores se pueden encontrar en la sección “Statistic” de Weka que se refleja en la *figura 1.2*. El rango de valores para el atributo petalwidth toma los valores de [0.1, 2.5] mientras que la media y la desviación típica toman los valores 1.199 y 0.763 respectivamente.

Statistic	Value
Minimum	0.1
Maximum	2.5
Mean	1.199
StdDev	0.763

*Figura 1.2: Statistic para el atributo
petalwidth*

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

Las instancias “Distinct” y “Unique” se nos muestra en Weka tras cargar una base de datos y seleccionar un atributo. “Distinct” hace referencia al número de diferentes valores que toman las instancias en una base de datos mientras que “Unique” hace referencia al número de instancias en una base de datos que toman valores únicos.

Por ejemplo, supongamos que tenemos una base de datos con las siguientes instancias:

{ROJO, ROJO, ROJO, AZUL, AZUL, VERDE, AMARILLO, MORADO}

En este caso las instancias “Distinct” serán 5, que hace referencia a los distintos colores que tenemos. Y las instancias “Unique” serán 3, que hace referencia a los valores únicos (VERDE, AMARILLO y MORADO).

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

Para determinar que atributo discrimina linealmente se podría hacer de manera sencilla. Como se puede observar en la matriz de la *figura 1.3*, si intentamos trazar una línea por el eje X que discrimine la clase Iris-setosa, en azul oscuro, de las otras dos, solamente se podría obtener con dos atributos, que serían el ancho y el alto del sépalo (sepalwidth y sepallenght). Con líneas rojas he separado linealmente la Iris-setosa de las otras dos clases usando el pétalo como atributo.

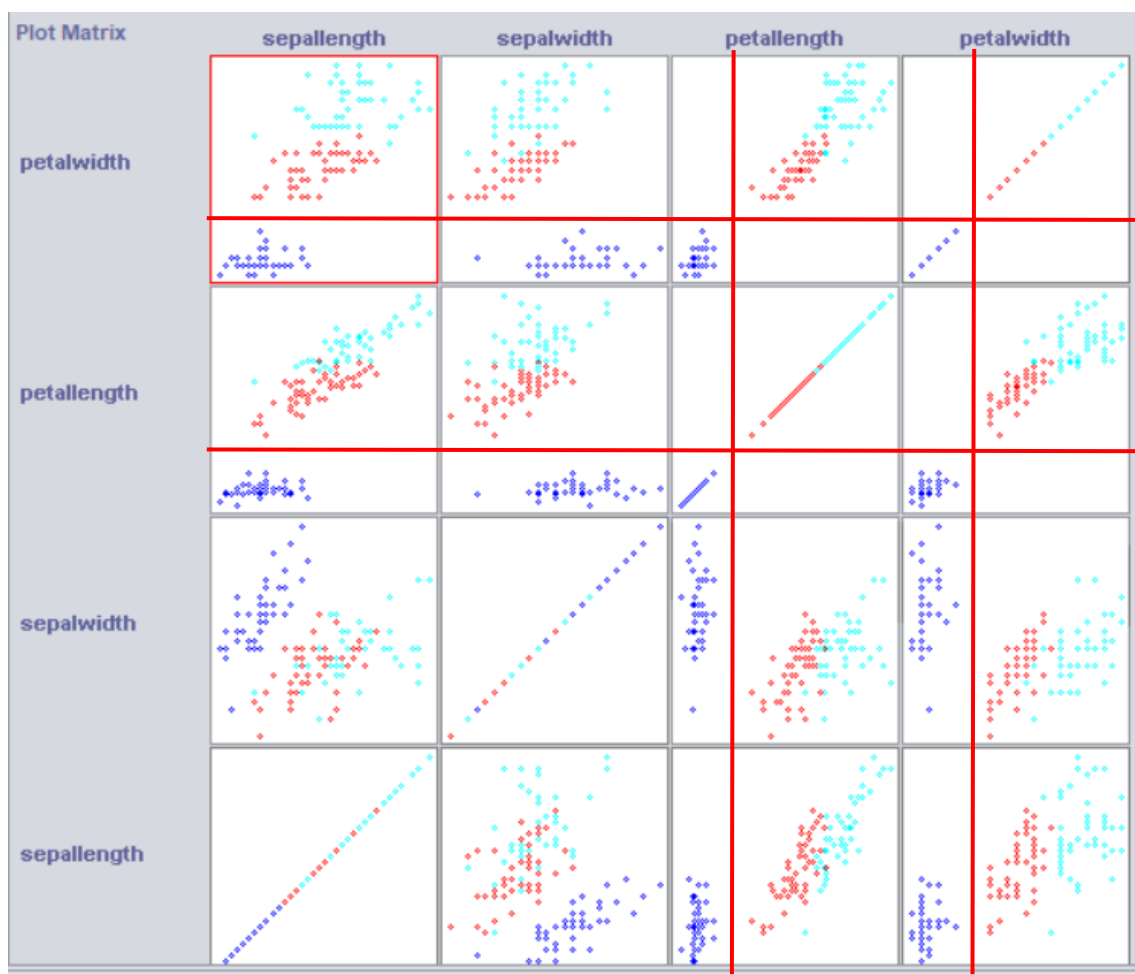


Figura 1.3: Plot Matrix con discriminación lineal de iris-setosa

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

No es posible separar ambas clases de manera lineal, ya que sus patrones están entremezclados.

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

Me quedaría con los atributos sepallength y petalwidth ya que son los que mejor diferencian las 3 clases como se observa en la *figura 1.4*.

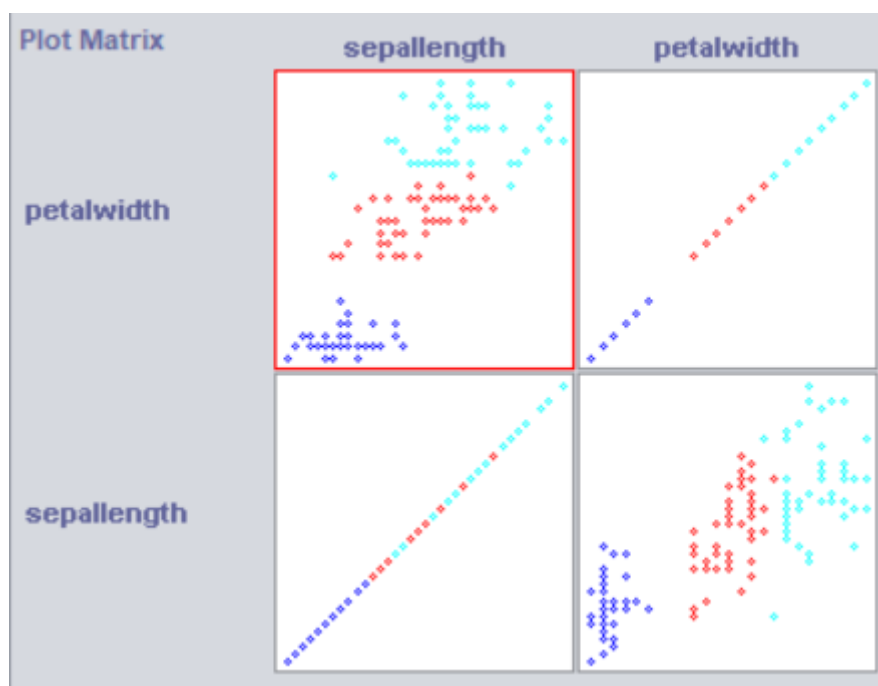
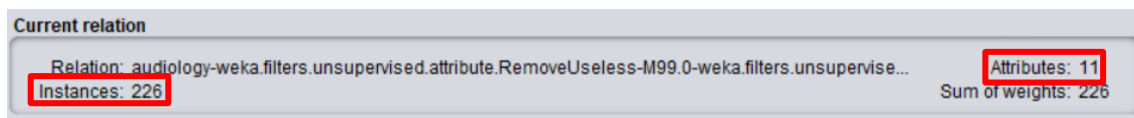


Figura 1.4: Plot Matrix para los atributos petalwidth y sepallength

1.2. Ejercicio 2: Cargue la base de datos audiology.

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

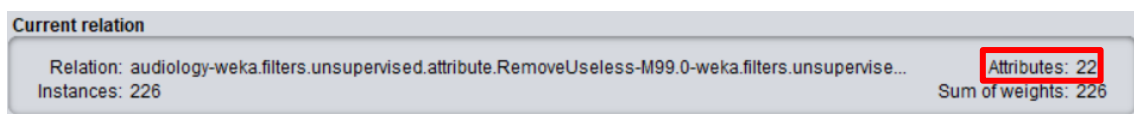
Una vez cargada la base de datos audiology podemos observar los atributos e instancias que contiene.



*Figura 1.5: Atributos e instancias del dataset
audiology*

Podemos ver que el número de Instancias es de 226 y de atributos es 11, todos ellos de tipo Nominal.

El filtro a aplicar es el “NominalToBinary” el cual como su nombre indica, convierte todos los atributos nominales en atributos binarios numéricos. Una vez aplicado, los atributos quedan así:



*Figura 1.6: Dataset audiology tras aplicar
filtro NominalToBinary*

Como se observa en la figura 6, el número de atributos se ha duplicado. Esto se debe a que atributos en los que se podía tomar por ejemplo 5 valores, como es el caso del atributo `air` (ver *figura 1.7.*), ahora cada valor forma a ser un atributo independiente con valor numérico 1 o 0.

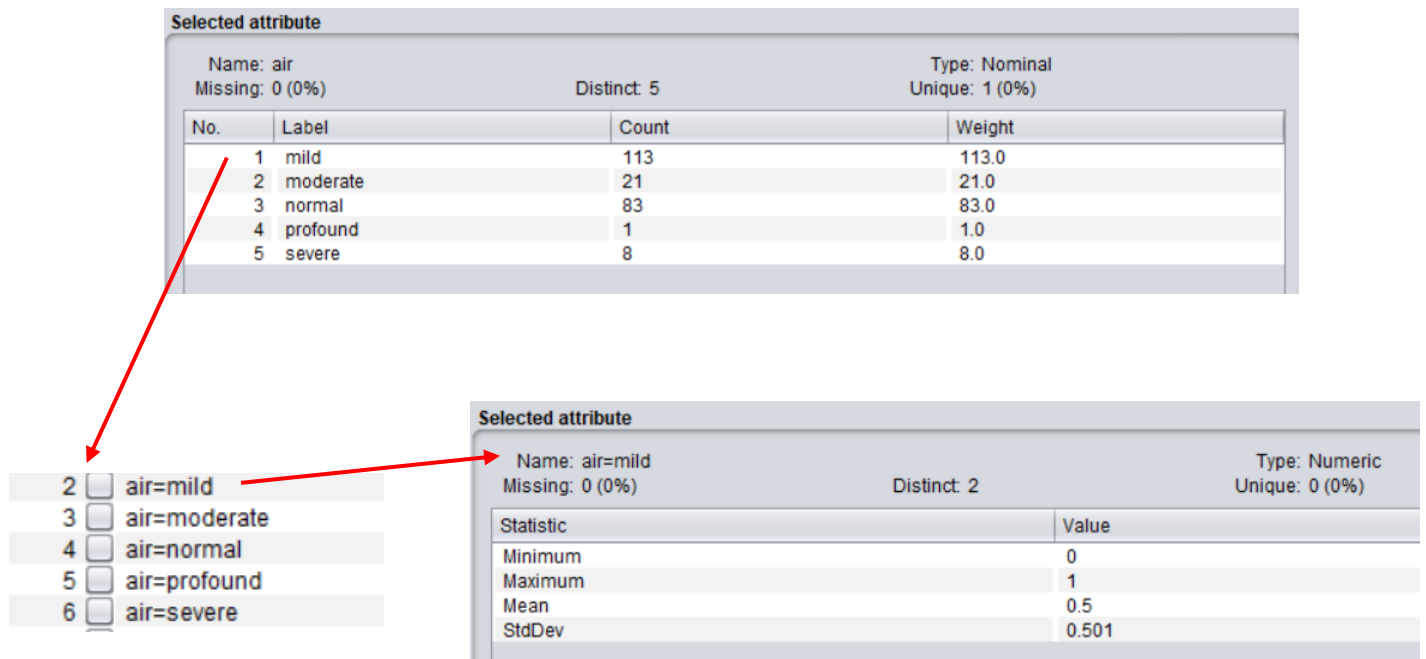


Figura 1.7: Atributo mild al detalle tras aplicar filtro NominalToBinary

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

Si, solo que habría que ir atributo a atributo mirando el número de valores que pueden tomar. Si el atributo puede tomar más de 2 valores nominales, se contarán como atributos adicionales. Por ejemplo, el atributo “bser” puede tomar dos valores nominales, “degraded” y “normal”, luego al transfórmalo a binario quedaría como un solo atributo, como antes de aplicar el filtro. Pero si el atributo tiene más de dos valores, como en el caso anterior de “air” cada valor será un nuevo atributo.

El problema esta que esta base de datos tiene pocos atributos y si es fácil saber con antelación el número de atributos finales antes de aplicar el filtro, si nos encontramos con una base de datos que tenga muchos más atributos, la labor sería mayor e impensable de realizar sin la aplicación del filtro.

1.3. Ejercicio 3: Particione la base de datos Iris usando filters/supervised/instance/StratifiedRemoveFolds

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

El filtro “StratifiedRemoveFolds” va a dividir la base de datos Iris de manera estratificada, es decir, manteniendo la proporción entre clases, para así preparar el modelo. El filtro va a dividir el dataset en un conjunto de “folds” o pliegues de manera que va a usar unos “folds” para entrenamiento y otros para generalización. Esto se hace para poder aplicar validación cruzada, en la que, en cada iteración, se va ir seleccionando unos “folds” diferentes para la validación, como se puede observar en la *figura 1.8*.

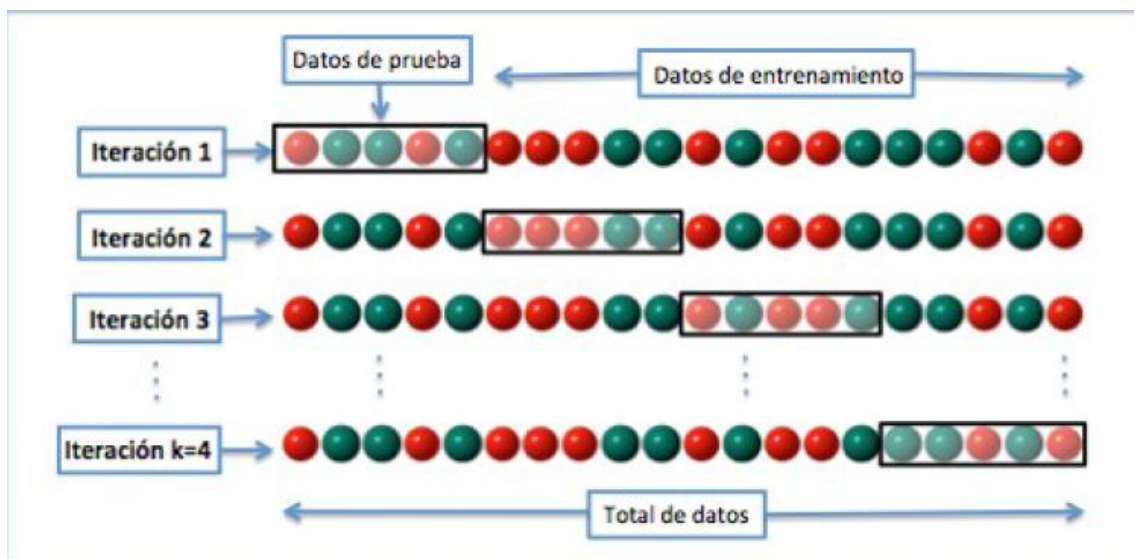
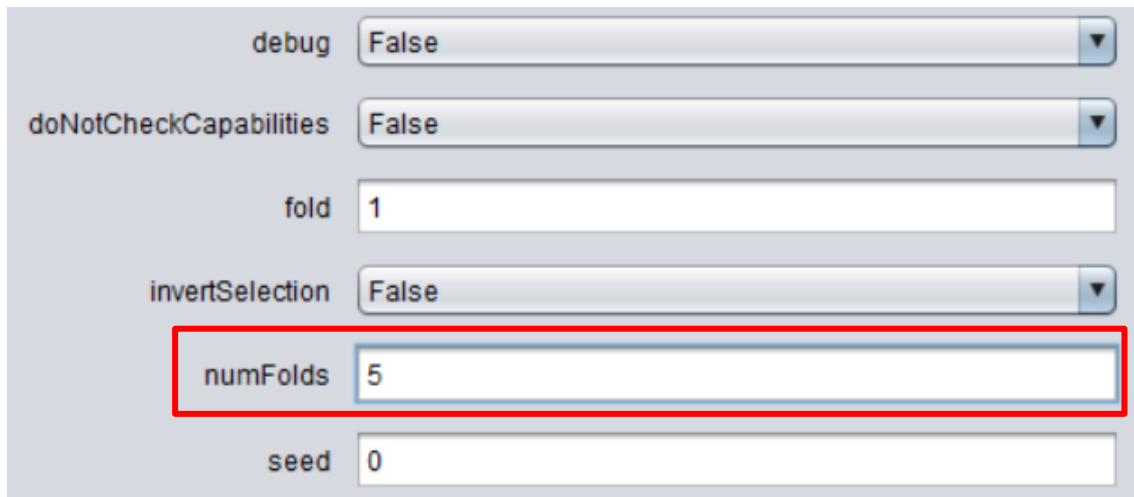


Figura 1.8: División en folds de un dataset

Una vez cargada la base de datos Iris, aplicamos el filtro teniendo en cuenta que el número de “folds” por iteración va a ser de 5.

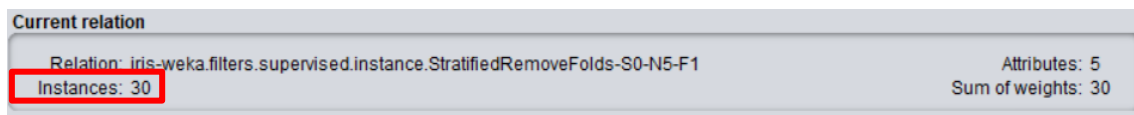


The image shows a configuration window for the 'StratifiedRemoveFolds' filter in Weka. The parameters are as follows:

Parameter	Value
debug	False
doNotCheckCapabilities	False
fold	1
invertSelection	False
numFolds	5
seed	0

*Figura 1.9: Parámetro numFolds del filtro
StratifiedRemoveFolds*

Una vez aplicado el filtro podemos observar cómo han cambiado los datos del dataset. Vemos como el número de instancias se ha reducido de 150 a 30. Esto se debe a lo que mencione antes, al dividir el dataset en 5 “folds” se han creado 5 “folds” con 30 instancias cada uno para así poder usarlos para la validación cruzada.



The image shows the 'Current relation' panel in Weka. The relation is 'iris-weka.filters.supervised.instance.StratifiedRemoveFolds-S0-N5-F1'. The number of instances is 30, which is highlighted with a red rectangle. The number of attributes is 5, and the sum of weights is 30.

Current relation
Relation: iris-weka.filters.supervised.instance.StratifiedRemoveFolds-S0-N5-F1
Instances: 30
Attributes: 5
Sum of weights: 30

*Figura 1.10: Instancias para el dataset Iris
tras aplicar filtro StratifiedRemoveFolds*

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

Con la base de datos cargada y el filtro StratifiedRemoveFolds seleccionado, nos dirigimos a cambiar sus parámetros.

Primero definimos el numFolds a 4 para así sacar 4 de los 5 holdOut que se piden. Para sacar el primero ponemos a 1 el parámetro fold indicando así que vamos a obtener la primera división del dataset en train y test. Por último, para distinguir entre Train y Test lo hacemos mediante el parámetro invertSelection, si está a False obtenemos el 25% de instancias para Test mientras que si lo ponemos a True obtenemos el 75% restante para Train.

Una vez hecho este proceso para cada uno de los 4 folds, el ultimo holdOut lo sacaríamos cambiando el parámetro seed por otro número, por ejemplo 1 y ya obtendría los 5.

1.4. Ejercicio 4: Ecoli dataset como base de datos para los guiones.

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

Tras acceder a la página web del UCI en la que se encuentra el dataset de Ecoli se nos muestra una gran variedad de información para el entendimiento de la base de datos antes de su procesamiento por Weka.

El dataset Ecoli contiene la ubicación de proteínas. Su creador y mantenedor es Kenta Nakai y los datos fueron donados por Pual Horton en septiembre de 1996. Las características de los atributos son reales y dispone de 8, 7

predictivos y 1 nominal sin faltarle ningún valor de atributo. Es una base de datos de clasificación que contiene 336 instancias.

En cuanto a los atributos, como he mencionado antes, tiene 8 que son:

- Sequence name: número de acceso para la base de datos SWISS-PROT.
- mcg: método de McGeoch para el reconocimiento de secuencia de señal.
- gvh: método de von Heijne para el reconocimiento de secuencia de señal.
- lip: puntuación de la secuencia consenso de “Signal Peptidase II” de von Heijne. (Atributo binario)
- chg: presencia de carga en el N-termino de las lipoproteínas pronosticadas. (Atributo binario)
- aac: puntuación del análisis discriminante del contenido de aminoácidos de la membrana externa y las proteínas periplásmicas.
- alm1: puntuación del programa de predicción de la región de membrana ALOM.
- alm2: puntuación del programa ALOM después de excluir las supuestas regiones de señal escindibles de la secuencia.

En cuanto a la distribución de clases tenemos que de las 336 instancias:

- 143 pertenecen a la clase citoplasmas (cp)
- 77 pertenecen a la clase membranas internas sin secuencia de señal (im)
- 52 pertenecen a la clase periplasmas (pp)
- 35 pertenecen a la clase membranas internas, secuencia de señal no divisible (imU)
- 20 pertenecen a la clase membranas externas (om)

- 5 pertenecen a la clase lipoproteínas de la membrana externa (omL)
- 2 pertenecen a la clase lipoproteínas de la membrana interna (imL)
- 2 pertenecen a la clase membranas internas, secuencia de señal divisible (imS)

1.4.2. 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 transformar la información descargada de la UCI en un archivo “.arff” valido he copiado el contenido de la información de la base de datos a un editor de texto he ido trasformando esa información para adecuarla a un archivo “.arff”.

Para ello primero he eliminado el primer atributo “Sequence Name” el cual no era descriptivo ya que no aportaba información útil para analizar. Después he ido adecuando la información al formato de archivo “.arff” como se puede observar en la *figura 1.12*.

1	AAT_ECOLI	0.49	0.29	0.48	0.50	0.56	0.24	0.35	cp
2	ACEA_ECOLI	0.07	0.40	0.48	0.50	0.54	0.35	0.44	cp
3	ACEK_ECOLI	0.56	0.40	0.48	0.50	0.49	0.37	0.46	cp
4	ACKA_ECOLI	0.59	0.49	0.48	0.50	0.52	0.45	0.36	cp
5	ADI_ECOLI	0.23	0.32	0.48	0.50	0.55	0.25	0.35	cp
6	ALKH_ECOLI	0.67	0.39	0.48	0.50	0.36	0.38	0.46	cp
7	AMPD_ECOLI	0.29	0.28	0.48	0.50	0.44	0.23	0.34	cp
8	AMY2_ECOLI	0.21	0.34	0.48	0.50	0.51	0.28	0.39	cp
9	APT_ECOLI	0.20	0.44	0.48	0.50	0.46	0.51	0.57	cp
10	ARAC_ECOLI	0.42	0.40	0.48	0.50	0.56	0.18	0.30	cp
11	ASG1_ECOLI	0.42	0.24	0.48	0.50	0.57	0.27	0.37	cp
12	BTUR_ECOLI	0.25	0.48	0.48	0.50	0.44	0.17	0.29	cp
13	CAFA_ECOLI	0.39	0.32	0.48	0.50	0.46	0.24	0.35	cp
14	CAIB_ECOLI	0.51	0.50	0.48	0.50	0.46	0.32	0.35	cp
15	CFA_ECOLI	0.22	0.43	0.48	0.50	0.48	0.16	0.28	cp
16	CHEA_ECOLI	0.25	0.40	0.48	0.50	0.46	0.44	0.52	cp

Figura 1.11: Dataset Ecoli

```

1  @relation ecoli
2
3  @attribute mcg numeric
4  @attribute gyh numeric
5  @attribute lip numeric
6  @attribute chg numeric
7  @attribute aac numeric
8  @attribute alm1 numeric
9  @attribute alm2 numeric
10 @attribute Class {cp,im,pp,imU,om,omL,imL,imS}
11
12 @data
13 0.49,0.29,0.48,0.50,0.56,0.24,0.35,cp
14 0.07,0.40,0.48,0.50,0.54,0.35,0.44,cp
15 0.56,0.40,0.48,0.50,0.49,0.37,0.46,cp
16 0.59,0.49,0.48,0.50,0.52,0.45,0.36,cp
17 0.23,0.32,0.48,0.50,0.55,0.25,0.35,cp
18 0.67,0.39,0.48,0.50,0.36,0.38,0.46,cp
19 0.29,0.28,0.48,0.50,0.44,0.23,0.34,cp
20 0.21,0.34,0.48,0.50,0.51,0.28,0.39,cp
21 0.20,0.44,0.48,0.50,0.46,0.51,0.57,cp

```

Figura 1.12: Dataset Ecoli en formato arff

1.4.3. Abra el entorno Weka Explorer -> Preprocess de Weka, cargue la base de datos y describa de forma concienzuda tanto los atributos como las clases.

Tras cargar el dataset “ecoli.arff” en el entorno Explorer de Weka, podemos ver una cantidad de información sobre la base de datos. Primero voy a comprobar que se han cargado bien los datos desde el fichero “.arff” y ver que se ha creado bien. El dataset Ecoli contenía 336 instancias con 8 atributos, cargando el archivo se puede ver que se han cargado correctamente las 336 instancias y los 8 atributos, quitando el primero que no era descriptivo y creando el atributo “Class”.

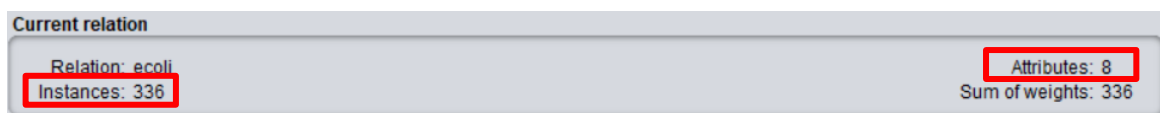


Figura 1.13: Instancias y atributos del dataset Ecoli

En cuanto a los atributos, ahora se puede hacer un análisis más exhaustivo de ellos. A continuación, voy a mostrar una tabla en la que se puede observar cada uno de los atributos con sus respectivos rangos, medias, desviaciones típicas y sus instancias Distinct y Unique:

Atributo	Rango	Media	Desv. Típica	Distinct	Unique
mcg	[0 , 0.89]	0.5	0.195	78	12
gvh	[0.16 , 1]	0.5	0.148	63	12
lip	[0.48 , 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

Tabla 1.1: Statistic de los atributos del dataset Ecoli

Podemos observar que todos los atributos son de tipo numérico y toman valores comprendidos en el rango [0 , 1]. Además, los atributos “lip” y “chg” solo toman dos valores diferentes nada más.

En cuanto a las clases, podemos ver su distribución ya que hemos creado un atributo que corresponde con la clase a la que pertenece cada instancia.

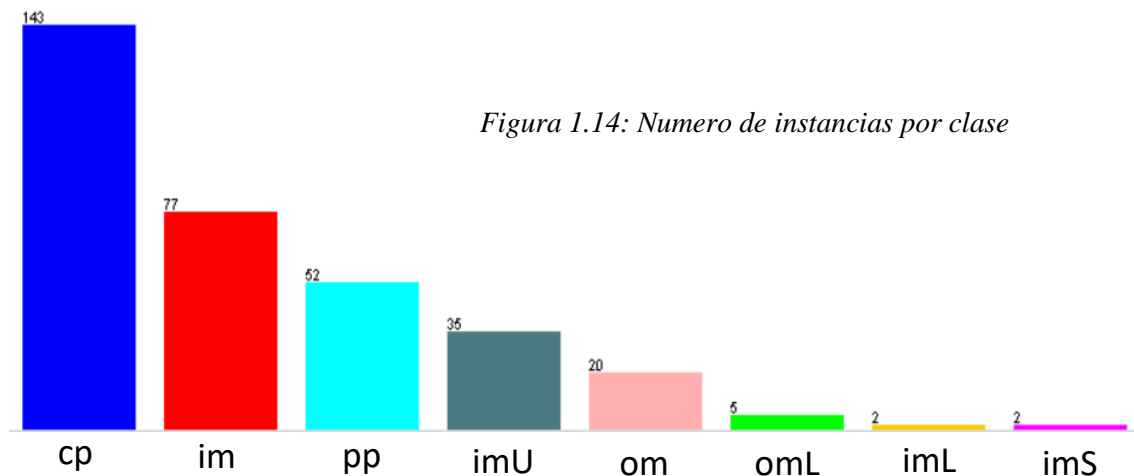


Figura 1.14: Numero de instancias por clase

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

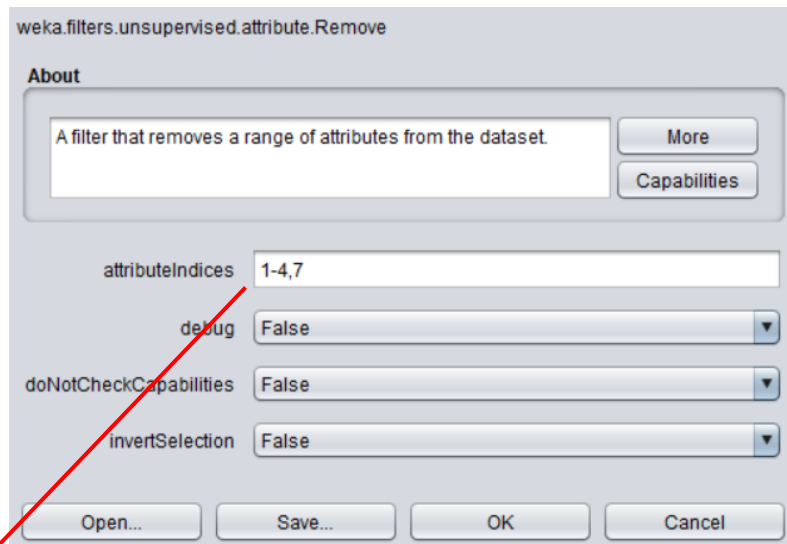
Un atributo identificador es un atributo que solamente define a un individuo, o en nuestro caso, a una instancia. En el caso del dataset ecoli, y como he mencionado anteriormente, el atributo “Sequence Name” fue eliminado del archivo “.arff” ya que era eso, un atributo identificador y no aportaba información útil para el análisis del dataset. Al contrario que los otros atributos que serían atributos descriptivos.

1.5. Ejercicio 5: Describa con sus propias palabras los siguientes filtros No Supervisados y acto seguido describa cómo quedan los datos al aplicar los filtros sobre la base de datos de sus prácticas.

1.5.1. filters/unsupervised/attribute/Remove

El filtro “Remove” es un filtro de atributos no supervisado, es decir, no tiene en cuenta el ultimo atributo del dataset el cual toma como clase o valor numérico de salida para regresión. “Remove” básicamente elimina atributos del dataset en un rango de atributos que se le pasa. Ese rango se puede pasar de varias maneras, si queremos eliminar atributos de manera específica simplemente indicamos el número de atributo que queremos eliminar y aplicamos el filtro, podemos especificar un rango o usar una combinación de ambos como se observa en la figura N.

A continuación, en la *figura 1.15* muestro un pequeño ejemplo del funcionamiento de este filtro:



1-4,7

Elimina el atributo número 7

Elimina los atributos del 1 al 4

Figura 1.15: Funcionamiento del filtro

Remove

Tras ejecutar el filtro Remove con esos parámetros, el dataset quedaría así:

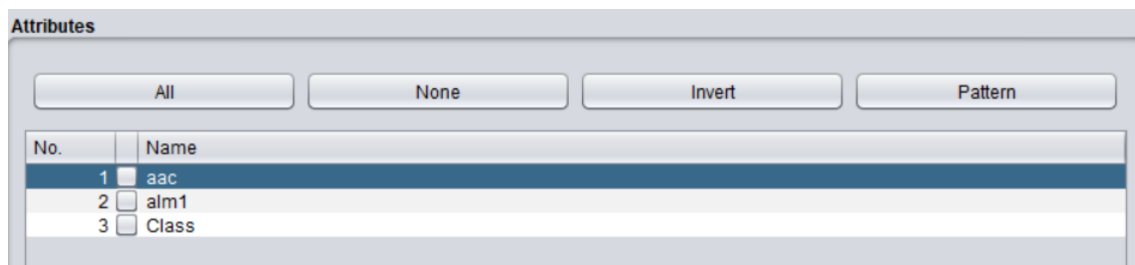


Figura 1.16: Atributos tras aplicar el filtro

Remove

Se puede ver como solamente han quedado los atributos que tenían como identificador los números 5, 6 y 8.

Este filtro no tiene sentido dejarlo aplicado en nuestro dataset ya que todos los atributos son necesarios y útiles para su análisis así que el quitar atributos no ayudaría.

1.5.2. filters/unsupervised/attributes/RemoveUseless

El filtro RemoveUseless elimina los atributos cuyos valores varían muy poco o que varían mucho. El variar mucho o poco viene definido por el parámetro “-M” que define la máxima varianza permitida antes de que un atributo sea eliminado.

Este filtro no puede ser aplicado ya que solo es aplicable a dataset con atributos nominales mientras que el dataset ecoli cuenta nada más que con atributos numéricos.

1.5.3. filters/unsupervised/attribute/Normalize

El filtro “Normalize”, como su nombre indica, normaliza los valores que pueden tomar los atributos de un dataset en un rango que se puede definir al gusto. Como valor por defecto se suele normalizar los valores en el rango [0,1]. Este filtro normaliza todos los atributos menos el último tributo que representa la clase.

Tras aplicar el filtro con 1 como valor de escala para la normalización, un atributo el cual antes tomaba como rango de valores [0,0.89] ahora pasa a tomar valores en el rango de [0,1].

A continuación, muestro un ejemplo de cómo quedaría un atributo tras aplicar este filtro.

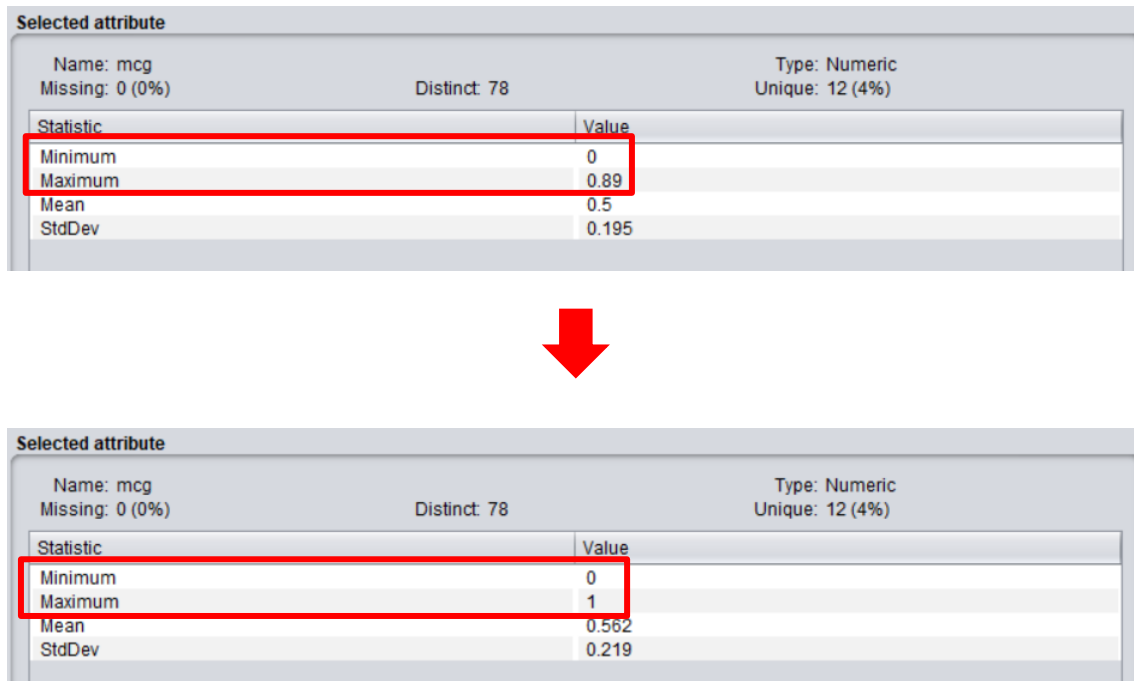


Figura 1.17: Atributo mcg tras aplicar el filtro Normalize

Normalizando los valores se consigue un mejor análisis ya que todos los atributos trabajarían en un rango de valores similar. En el caso del dataset ecoli no haría falta aplicar este filtro ya que los atributos ya toman valores entre [0,1].

1.5.4. filters/unsupervised/attribute/ReplaceMissingValues

Normalmente cuando se trabaja con bases de datos en normal encontrarse que en los atributos hay valores que son 0 cuando no deben serlo o toman valores corruptos los cuales causaran problemas en los análisis. Para poder solventar este problema podemos usar muchas opciones, una de ella es el uso del filtro “ReplaceMissingValues” el cual coge esos

valores que están fuera de lo normal y los transforma en valores que pueden computar en los análisis como por ejemplo la media de todos los valores.

En el caso de la base de datos ecoli está hecha de manera que no cuenta con “Missing values” con lo que este filtro no podrá ser aplicable.

1.5.5. filters/unsupervised/attributes/NominalToBinary

El filtro “NominalToBinary” transforma los atributos nominales de un dataset en atributos binarios. El funcionamiento de este filtro se ha podido comprobar en la pregunta 1.2.1 de esta práctica.

Con el dataset ecoli no se puede aplicar este filtro ya que sus atributos son todos de tipo numérico y tienen que ser de tipo nominal para poder ser aplicado.

Como ejemplo se puede ver la pregunta 1.2.1 como he comentado anteriormente.

1.5.6. filters/unsupervised/instance/RemovePercentage

Ahora pasamos a analizar los filtros que se aplican a las instancias y no a los atributos. El primer filtro a aplicar es el “RemovePercentage” el cual básicamente recibe como parámetro el porcentaje de instancias a eliminar y una vez aplicado, elimina ese porcentaje de instancias. Este filtro se podría usar para reducir un dataset extenso y poder analizarlo de manera más simple.

En el dataset ecoli, una ejecución de este filtro con un porcentaje del 50% produciría estos cambios:

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



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

Figura 1.18: Atributo Class tras ejecutar el filtro RemovePercentage

He mostrado el atributo “Class” en el que se muestran todas las clases con el número de instancias que pertenecen a esa clase. Como se puede observar el número de instancias se han reducido a la mitad, han pasado de 336 a 168. El problema está en que este filtro quita el porcentaje de las instancias de manera no proporcional, es decir, en el caso de este dataset las clases están ordenadas y las primeras 143 instancias son de la clase “cp” así que si se van a quitar instancias 168 instancias se van a empezar a quitar en orden con lo que la clase “cp” se queda sin instancias. Así que no se debería aplicar este filtro a el dataset ecoli ya que dejaría sin instancias en clases que tienen pocas instancias.

1.5.7. filters/unsupervised/instance/RemoveDuplicates

“RemoveDuplicates” es un filtro para instancias el cual elimina las instancias que sean iguales, es decir, que sus atributos tomen los mismos valores de manera que se pueda eliminar datos redundantes para un análisis posterior. En el caso del dataset ecoli no hay ninguna instancia duplicada luego no tendría sentido aplicar este filtro.

Para poner un breve ejemplo, he duplicado dos instancias de la base de datos ecoli para hacer una prueba y ver qué sucede cuando ejecuto el filtro “RemoveDuplicates”. Ahora el dataset contiene 338 instancias, dos más que antes, al aplicar el filtro se aprecia como el número de instancias ha vuelto a su valor normal sin duplicados.

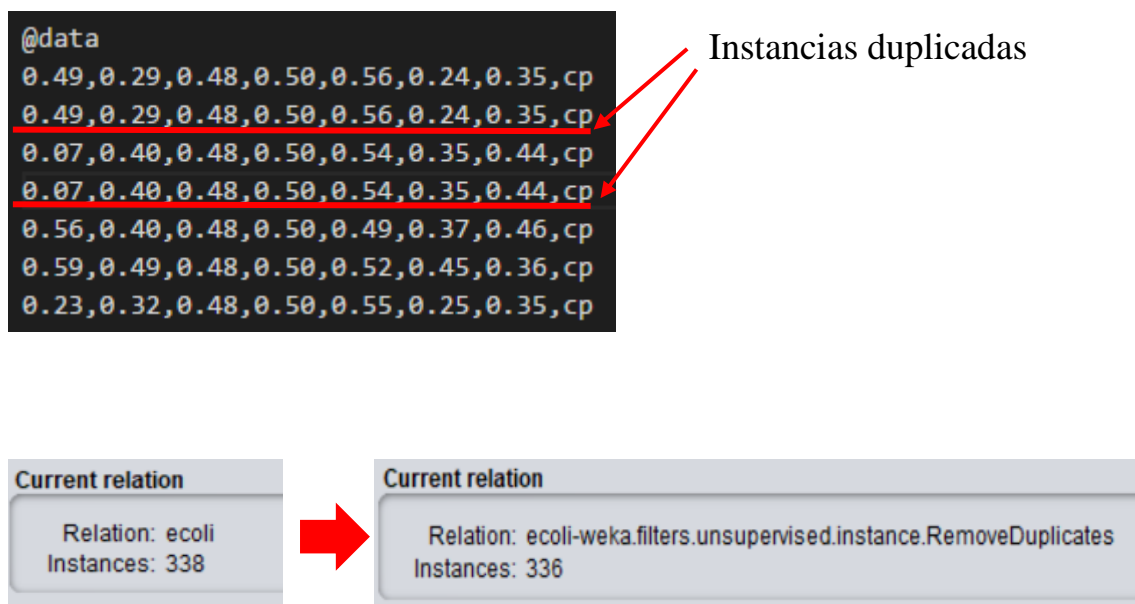


Figura 1.19: Ejemplo de ejecución del filtro
RemoveDuplicates

1.5.8. filters/unsupervised/instance/Resample

El filtro Resample genera un dataset diferente al original, con el que se intenta igualar un poco la diferencia de instancias entre las clases.

Los parámetros que podemos ajustar son los siguientes:

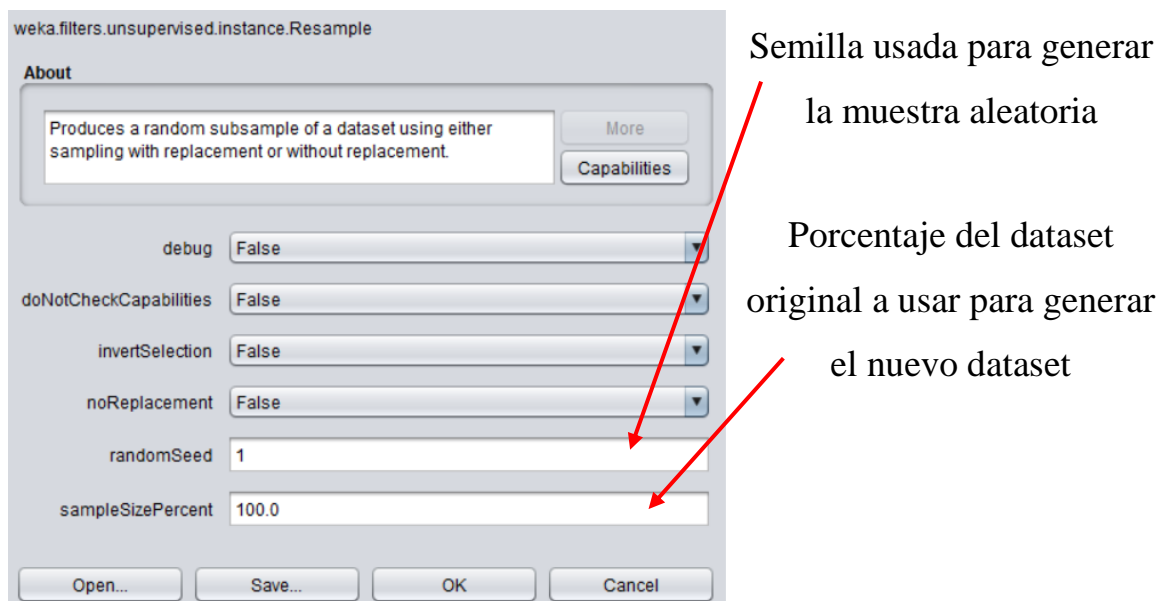


Figura 1.20: Parámetros de configuración del filtro Resample

Tras ejecutar el filtro, podemos observar como se ha creado otro dataset en el que se han añadido patrones nuevos a clases como “imS”, que tenía originalmente 2 patrones y ahora tiene 3, o omL”, que tenía 5 patrones y ahora tiene 6, y se han quitado a otros como es el caso de “imL” que ahora solo tiene 1 cuando tenía 2.

Luego este filtro no solventa el problema de desbalanceo que tenemos en el dataset.

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.

1.6.1. filters/supervised/attribute/Discretize

Paso ahora a analizar los filtros supervisados, los cuales, a diferencia de los no supervisados, necesitan usar el ultimo atributo del dataset a la hora de hacer un tratamiento sobre los datos. Este último atributo es la clase a la que pertenece cada instancia al ser un problema de clasificación.

El filtro “Discretize” trasforma los atributos seleccionados en atributos de tipo nominal con una serie de etiquetas que resultan de dividir el rango total del atributo en intervalos. Este filtro tiene varias opciones, las más útiles a mi parecer son el parámetro “precisión” el cual dicta con que cantidad de decimales se va separar el rango de un atributo y el parámetro “makeBinary” el cual transforma todos los atributos en atributos nominales binarios.

Para aplicar este filtro voy a dejar los valores por defecto, voy a transformar todos los atributos, con una precisión en decimales de 6 y sin hacerlos binarios como se puede ver en la *figura 1.21*.

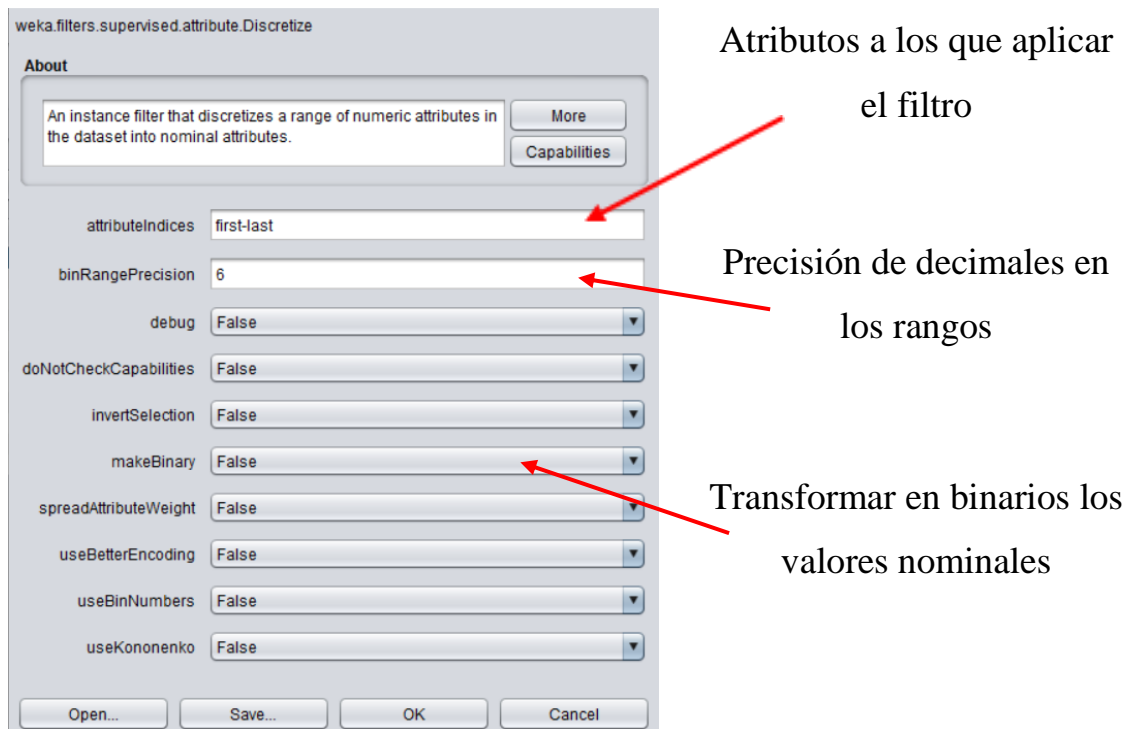


Figura 1.21: Parámetros de configuración del filtro Discretize

Tras la ejecución de este filtro paso a mostrar cómo quedaría un atributo (mcg) cuyo rango de valores es de [0,0.89] y que toma 78 valores distintos en ese rango.

Name: mcg Missing: 0 (0%)		Distinct: 78	Type: Numeric Unique: 12 (4%)
Statistic	Value		
Minimum	0		
Maximum	0.89		
Mean	0.5		
StdDev	0.195		

Name: mcg Missing: 0 (0%)		Distinct: 3	Type: Nominal Unique: 0 (0%)
No.	Label	Count	Weight
1	'(-inf-0.555]'	187	187.0
2	'(0.555-0.755]'	123	123.0
3	'(0.755-inf)'	26	26.0

Figura 1.22: Atributo mcg tras la ejecución del filtro Discretize

Hemos pasado de un atributo numérico a uno nominal en el que antes se podían tomar 78 valores numéricos diferentes en un rango de $[0,0.89]$ y ahora solo hay 3 etiquetas que reflejan tres rangos en los que se ubican los valores. Con este filtro podemos saber cuántas instancias toman que valores de un atributo seleccionado. Por ejemplo, en el caso del atributo “mcg” mostrado anteriormente. 187 instancias toman valores de entre $-\text{inf}$ a 0.555, 123 instancias toman valores de entre 0.555 a 0.755 y los 26 restantes toman valores de entre 0.755 a inf . Con esto conseguimos un análisis más claro del dataset.

Ya que todavía no podemos ver la pestaña Classify de Weka para saber si el aplicar este filtro mejora el análisis del dataset no lo voy a dejar aplicado, sin embargo, para poder usar los siguientes filtros voy a dejarlo aplicado para ver cómo se comporta el dataset.

1.6.2. filters/supervised/attribute/NominalToBinary

Como he mencionado anteriormente voy a dejar el filtro “Discretize” para poder aplicar así el filtro “NominalToBinary” al dataset Ecoli. El funcionamiento de este filtro es igual al aplicado en la pregunta 1.5.5. Tras ejecutarlo podemos ver el siguiente resultado en los atributos:

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

Figura 1.23: Atributos tras la ejecución del filtro NominalToBinary

Pienso que este filtro no va a mejorar el análisis del dataset así que no lo dejare aplicado ya que hace que los resultados parezcan menos interpretativos.

1.6.3. filters/supervised/attribute/SpreadSubsample

“SpreadSubsample” al igual que el filtro “Resample” genera un dataset aleatorio a partir del dataset original. La diferencia de “SpreadSubsample” es que le puedes indicar el número máximo de separación entre la clase menos común y las más común, consiguiendo así igualar el número de instancias entre clases. Si en el argumento que define la distribución ponemos el valor 0 no produciría ningún cambio, pero si ponemos 1 conseguiremos una distribución uniforme ya que la separación máxima entre clases sería de 1.

Un ejemplo claro se ve al aplicar este filtro con valor de “spread” máximo de 1:

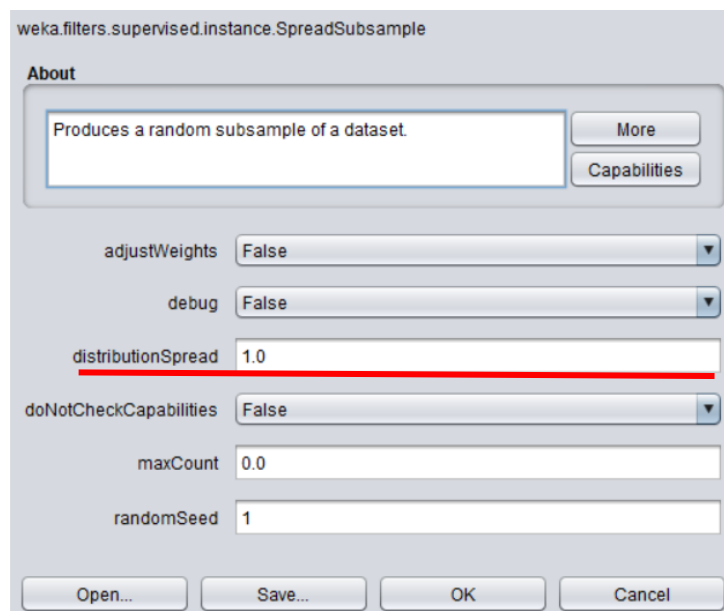


Figura 1.24: Configuración de parámetros para el filtro SpreadSubsample

Current relation	
Relation: <code>ecoli-weka.filters.supervised.attribute.Discretize-Rfirst-last-precision6-weka.filters.supervised.in...</code>	Attributes: 8
Instances: 16	Sum of weights: 16

Figura 1.25: Número de instancias tras aplicar el filtro SpreadSubsample

Vemos como ahora el número de instancias es de 16, es decir, 1 instancia por cada valor que puede tomar los atributos, recordando que estamos trabajando con la base de datos ecoli discretizada.

No aplicaré este filtro ya que no aporta nada relevante para un análisis posterior del dataset.

1.6.4. filters/supervised/attribute/ClassBalancer

Con este filtro conseguimos obtener un dataset totalmente balanceado en el que hay el mismo número de instancias en cada clase. Al aplicar este filtro se consigue que el dataset tenga 42 instancias en cada clase ya que 42 es el número de instancias que se obtiene al dividir todas las instancias entre el total de las clases.

Name: Class		Distinct: 8	Type: Nominal
Missing: 0 (0%)			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

Figura 1.26: Número de instancias por clase tras aplicar filtro ClassBalancer

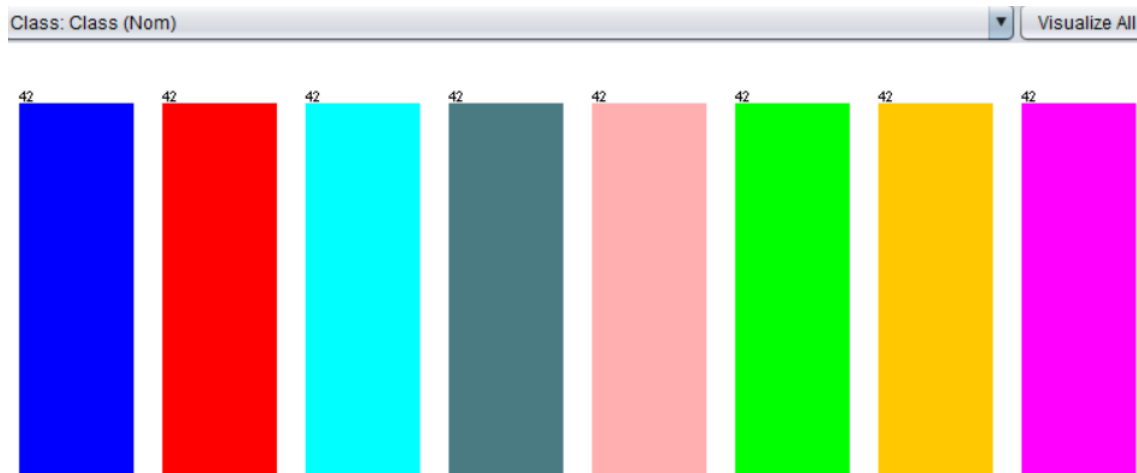


Figura 1.27: Número de instancias por clase tras aplicar filtro ClassBalancer

Este filtro sería muy útil para datasets en los que hay mucha diferencia entre clases en cuanto a número de instancias. En el caso de ecoli se da ya que la clase “cp” tiene 143 instancias y la clase “imS” 2 así que la diferencia es notable pero no demasiado grande como para aplicarlo. Luego este filtro podría ser útil ya que consigue que todas las clases tengan el mismo peso en el dataset, aun así, las instancias que se generan nuevas podrían dar problemas e inconsistencia a la base de datos así que no la aplicaré.

1.6.5. filters/supervised/attribute/Resample

El filtro Resample, como ya hemos visto en el punto 5.8, genera un dataset aleatorio. La única diferencia es que, al ser un filtro supervisado, va a tener en cuenta el ultimo tributo que hace referencia a la clase a las que pertenecen las instancias, luego va a mantener la proporción de estas.

Por ejemplo, si aplico el filtro y pongo como valor de parámetro para el tamaño de la muestra del 10%, el filtro mantendrá la proporción entre clases dando el siguiente resultado:

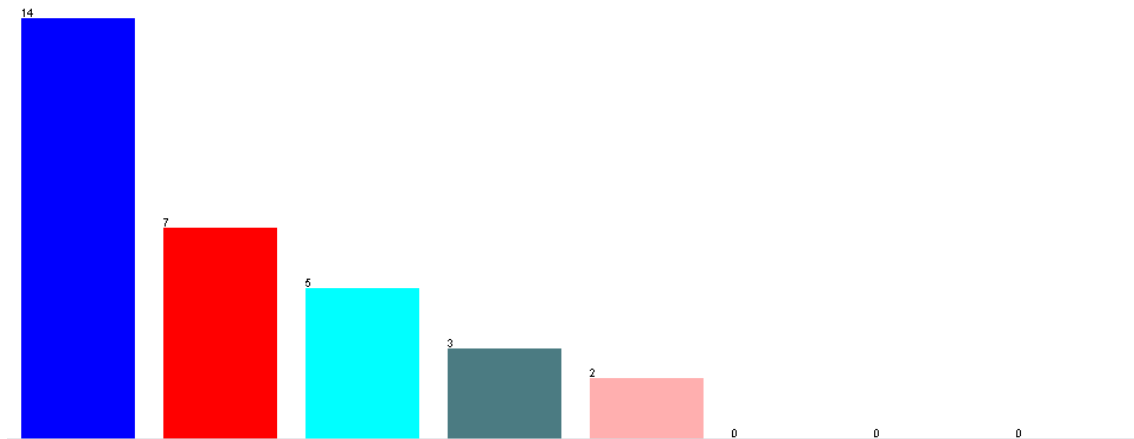


Figura 1.27: Número de instancias por clase tras aplicar filtro Resample

Se puede apreciar como el filtro ha mantenido la proporción, dando lugar a que las clases con menor número de instancias se han quedado a cero. Luego este filtro no aportaría nada significativo para un posterior análisis del dataset.

1.7. Ejercicio 7: Haga un breve resumen de los filtros aplicados de manera permanente a su base de datos final. Si no ha aplicado filtros porque no tengan sentido para esta base de datos con los conocimientos actuales, indique que se queda sin cambios.

Tras analizar uno a uno cada uno de los filtros, he llegado a la conclusión de que por ahora no voy a dejar aplicado ninguno de ellos. Al no poder usar la pestaña “Classify” de Weka todavía, no se ha podido analizar el rendimiento de esos filtros con el dataset. El problema de este dataset es que tiene pocas instancias y las clases están muy desbalanceadas. Las mayoritarias tienen muchas más instancias que las

minoritarias, que son ínfimas, y si quiero hacer un buen análisis del dataset debo tener en cuenta cada una de las instancias de las clases.

Ninguno de los filtros a aplicar resuelve ese problema con los conocimientos que tengo.

2. Práctica 2-1: Regresión y clasificación 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 (métricas, gráficas y matriz de confusión), interpretando los resultados. Tenga en cuenta si se clasifican bien todas las clases de su problema (TP Rate por clase), es decir, tenga en cuenta el balanceo de su base de datos.

Una vez cargada nuestra base de datos Ecoli en el entorno Explorer de Weka procedo a aplicar el algoritmo de clasificación IB1. Este algoritmo clasifica siguiendo un algoritmo de vecinos más cercanos el cual se basa en los N patrones más cercanos a uno dado para etiquetarlo en una clase. El IB1 usa la distancia Euclídea normalizada para encontrar la instancia de train más cercana a la instancia test dada y así predecir la misma clase a la que pertenece la instancia de train.

La configuración del algoritmo se puede ver en las siguientes figuras:

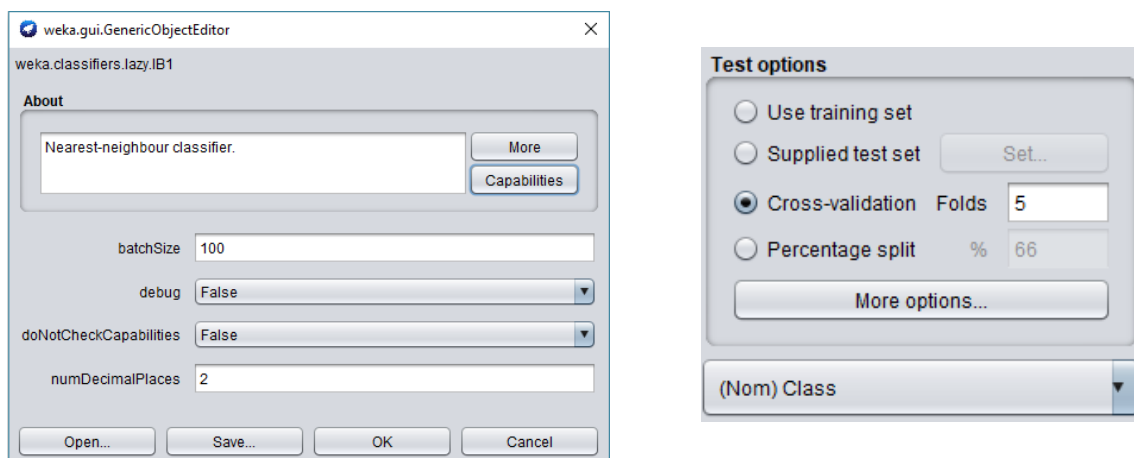


Figura 2.1: Configuración del algoritmo IB1

Al usar 5 como número de folds en la Cross-validation, queremos indicar al algoritmo que va a usar 5 subconjuntos del dataset como datos de test en cada iteración. Ejecutando esta configuración obtenemos la siguiente salida por pantalla.

Primero vemos cierta información sobre la base de datos que se ha usado y el algoritmo que se ha aplicado.

```

=== Run information ===

Scheme:      weka.classifiers.lazy.IB1
Relation:    ecoli
Instances:   336
Attributes:  8
             mcg
             gvh
             lip
             chg
             aac
             alml
             alm2
             Class
Test mode:   5-fold cross-validation

```

*Figura 2.2: Información sobre el algoritmo
IB1 ejecutado*

Seguidamente tenemos el resumen de la clasificación realizada la cual procederé a comentar:

Correctly Classified Instances	272	80.9524 %
Incorrectly Classified Instances	64	19.0476 %
Kappa statistic	0.7372	
Mean absolute error	0.0476	
Root mean squared error	0.2182	
Relative absolute error	26.0201 %	
Root relative squared error	72.3392 %	
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	0,946	0,903	cp
	0,740	0,077	0,740	0,740	0,740	0,663	0,832	0,608	im
	0,827	0,046	0,768	0,827	0,796	0,758	0,891	0,662	pp
	0,486	0,050	0,531	0,486	0,507	0,454	0,718	0,312	imU
	0,750	0,006	0,882	0,750	0,811	0,803	0,872	0,677	om
	1,000	0,003	0,833	1,000	0,909	0,911	0,998	0,833	omL
	0,000	0,006	0,000	0,000	0,000	-0,006	0,497	0,006	imL
	0,000	0,003	0,000	0,000	0,000	-0,004	0,499	0,006	imS
Weighted Avg.	0,810	0,052	0,805	0,810	0,807	0,757	0,879	0,711	


```

=== 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

*Figura 2.3: Resumen de la clasificación
realizada por el algoritmo IB1*

La primera métrica que podemos observar es el **CCR** (el cual he marcado en rojo). El CCR nos indica el número de instancias bien clasificadas por el algoritmo. Vemos que he obtenido un 80% de patrones bien clasificados lo cual es un buen resultado, de 336 se ha clasificado correctamente 272. Seguidamente podemos ver la métrica contraria al CCR, es decir, el número de instancias mal clasificadas por el algoritmo, que son el 19%.

La siguiente métrica a tener en cuenta es el estadístico **KAPPA** (el cual he marcado en azul). El estadístico KAPPA compara la concordancia observada en un conjunto de datos por un modelo respecto a la que podría ocurrir por mero azar. Puede tomar valores en el rango $[-1,1]$ y se tiene que conseguir maximizar. En el modelo hemos obtenido un KAPPA de 0.73, valor cercano al 1 luego nos quiere indicar que nuestro modelo tiene una mejor concordancia que la que se esperaría por puro azar.

A continuación, nos encontramos con los **errores** cometidos mediante diferentes métricas (marcadas en verde):

- Mean Absolute Error (MAE): la media del error absoluto es la diferencia entre el valor real y el valor que se ha obtenido, es decir, la diferencia que hay entre la clase predicha y la clase real. El resultado obtenido es de 0.04 luego es un error muy bajo, indicativo de que el modelo ha clasificado bien.
- Root Mean Squared Error (RMSE): otra medida de error que se básicamente calcula lo mismo, pero mediante otra fórmula. La diferencia reside en para que se va a calcular el error.

- Relative Absolute Error (RAE): el error relativo absoluto es el cociente entre el error absoluto y la media de los valores reales. Este porcentaje nos indica cuanto se diferencia el valor real de su valor propio medio. En mi caso el resultado es de un 26.02%.
- Root Relative Squared Error (RRSE): otra medida de error que se básicamente calcula lo mismo, pero mediante otra fórmula. La diferencia reside en para que se va a calcular el error.

Por último, tenemos las **métricas** para la clasificación multiclase (en rosa) y la **matriz de confusión** (en naranja) que voy a analizar detenidamente a continuación.

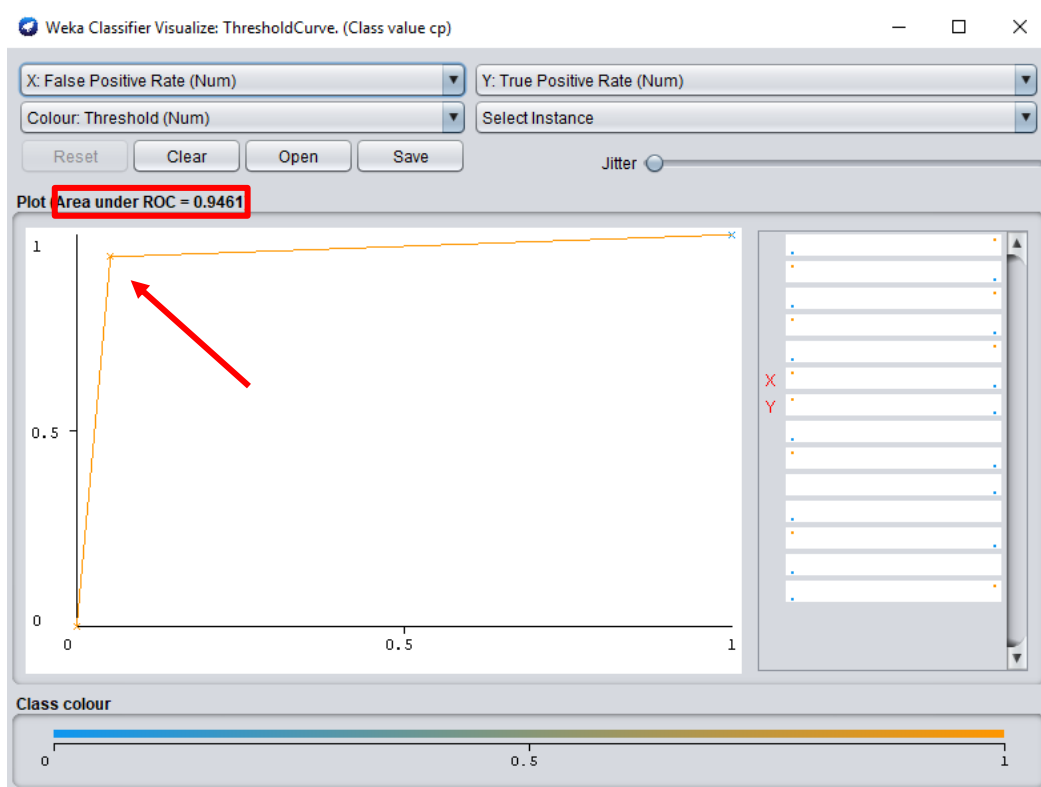
Al estar ante un problema de clasificación multiclase las métricas obtenidas se obtienen considerando la clase a analizar como positiva. Las métricas obtenidas son las siguientes:

- TP Rate: indica el porcentaje de patrones positivos predichos como positivos, es decir, el número de patrones que nuestro modelo ha predicho correctamente. Los valores que se pueden destacar por clase son que la clase mayoritaria “cp” tiene un TPRate de 0.94 luego el modelo ha clasificado muy bien los patrones de esta clase. A tener en cuenta también las dos clases minoritarias de tan solo 2 patrones cada una (imL y imS) las cuales han obtenido un TPRate de 0. Esto se debe a que al estar usando el algoritmo de vecinos más cercanos puede darse el caso de que al tener solo dos patrones las clases, estas muy separados entre sí y al estar usando el algoritmo IB1, la probabilidad de que el patrón que esté más cerca sea de esas clases es muy baja. El TPRate ponderado total es de un 81% de patrones bien clasificados, es decir, prácticamente el CCR obtenido, que es una buena cifra. Aun

así, podemos decir que nuestro modelo no clasifica bien todas las clases ya que el dataset está muy desbalanceado.

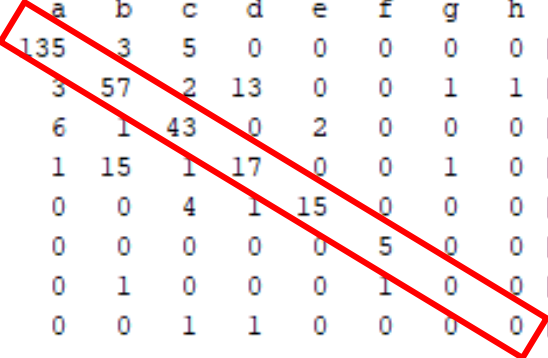
- **FP Rate:** es el porcentaje de patrones negativos predichos como positivo. En esta medida hay que conseguir buscar el menor valor posible ya que refleja que los patrones no están siendo bien clasificados. El valor medio ponderado total es de 0.052, un valor ínfimo. Esto quiere decir que nuestro modelo ha clasificado muy pocos patrones como positivos cuando eran negativos.
- **Precision:** porcentaje de patrones positivos predichos como positivos, frente al total de patrones predichos como positivos. Se calcula dividiendo el TPRate de una clase entre su TPRate más su FPRate. Básicamente con esta métrica obtenemos un valor más sólido que con los dos anteriores ya que está juntando ambas métricas anteriores. El valor medio ponderador total es de 0.805, valor muy similar al TPRate total ya que el FPRate total es muy bajo, luego ha influido muy poco.
- **F-Measure:** porcentaje que combina las métricas Recall (TPRate) y Precision. Otra métrica más que hay que maximizar la cual nos sigue indicando si nuestro modelo ha calificado bien usando el algoritmo aplicado o no. El valor obtenido es de 0.807.
- **MCC:** es el coeficiente de correlación de Matthews y se usa para medir la asociación entre dos variables binarias.
- **ROC Area:** el AUC o Area Under Curve es una, métrica que indica la probabilidad de que un clasificador ordene o puntué una instancia positiva elegida aleatoriamente más alta que una negativa. En nuestro

caso obtenemos un valor total ponderado de 0.879, valor que es muy similar a los otros obtenidos que nos estamos moviendo siempre en el rango del 80% de los patrones bien clasificados. Si queremos visualizar el AUC de una clase de manera gráfica, podemos hacerlo en Weka mediante la visualización de “Treshold Curve”. En la *figura 2.4* a continuación podemos ver la curva ROC de la clase cp que tiene un ROC Area de 0.946. El valor ideal de un AUC debe ser 1 o cercano a uno, esto daría lugar a un modelo perfecto el cual representaría un 100% de sensibilidad, ningún FN, y un 100% de especificidad, ningún FP.



*Figura 2.4: Curva ROC señalando el valor
optimo alcanzado*

Voy a analizar a continuación la matriz de confusión dada por nuestro modelo.



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

Figura 2.5: Matriz de confusión con las instancias bien clasificadas marcadas

En la matriz de confusión podemos ver que patrones nuestro modelo ha clasificado correctamente y cuales incorrectamente. La diagonal principal (señalada en rojo) de la matriz señala los patrones que han sido clasificados correctamente. Por ejemplo, si nos vamos a la clase “cp”, “a” en la matriz, vemos que de 143 patrones que tiene la clase “cp” en total, nuestro modelo ha clasificado 135 correctamente, 3 los ha clasificado como “im”, 6 como “pp” y 1 como “imU”. Con estos valores podríamos sacar el 81% de patrones de la clase “cp” bien clasificados que weka nos muestra.

La matriz de confusión es muy útil ya que nos permite hacer un análisis mas detallado de la lcasificacion que nuestro modelo ha hecho de cada clase. Podemos ver varios casos significativos como lo mal que ha clasificado los patrones de las clases “imL” e “imS”, comentados anteriormente, o como clasifica al 100% la clase “omL”.

Para un mejor análisis, voy a mostrar la clasificación que ha hecho nuestro modelo gráficamente. La *figura 2.6.* a continuación nos muestra la matriz de confusión, pero de una manera más gráfica en la que podemos ver como

nuestro modelo a agrupado los patrones de cada clase. He tenido que introducirle algo de ruido (jitter) para una mejor visualización:

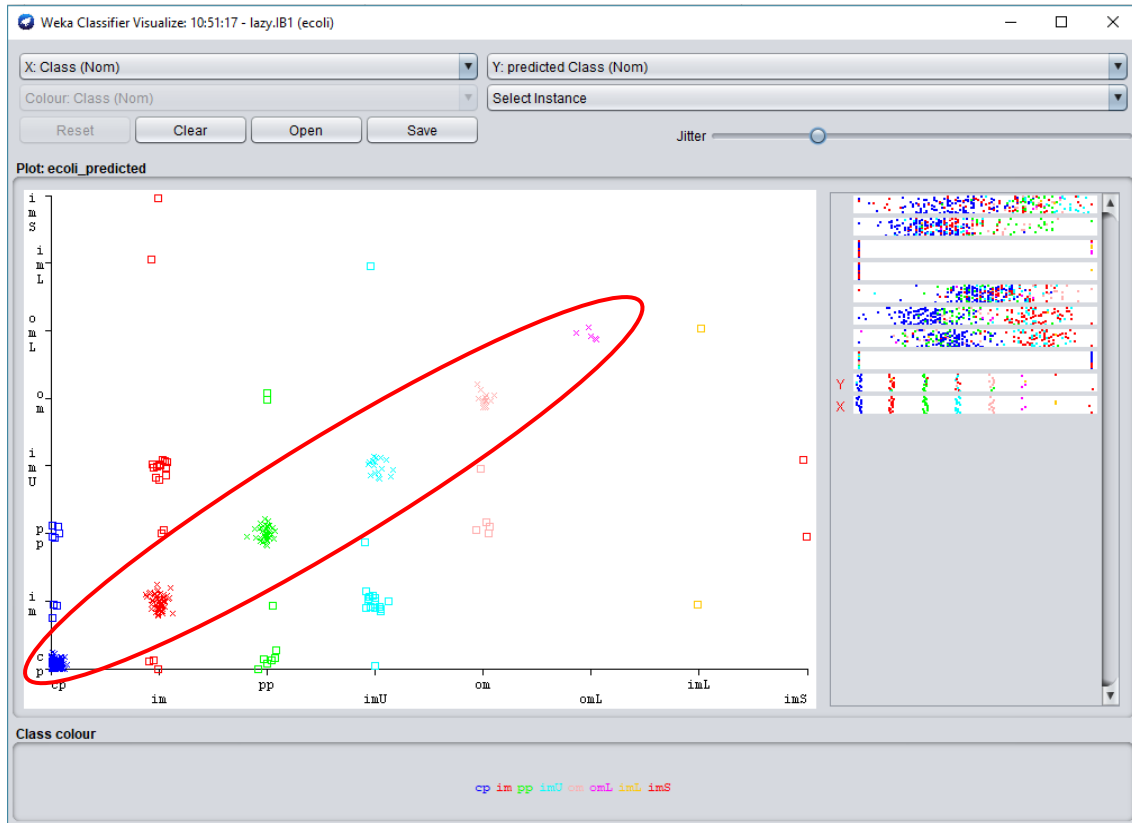


Figura 2.5: Matriz de confusión con las instancias bien clasificadas marcadas

En rojo he señalado lo que correspondería a los patrones bien clasificados de cada clase.

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.

2.2.1. 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 poder realizar esta parte primero hay que configurar el entorno Experimenter de Weka para la realización de las pruebas. En la *figura 2.6.* a continuación se puede apreciar la configuración:

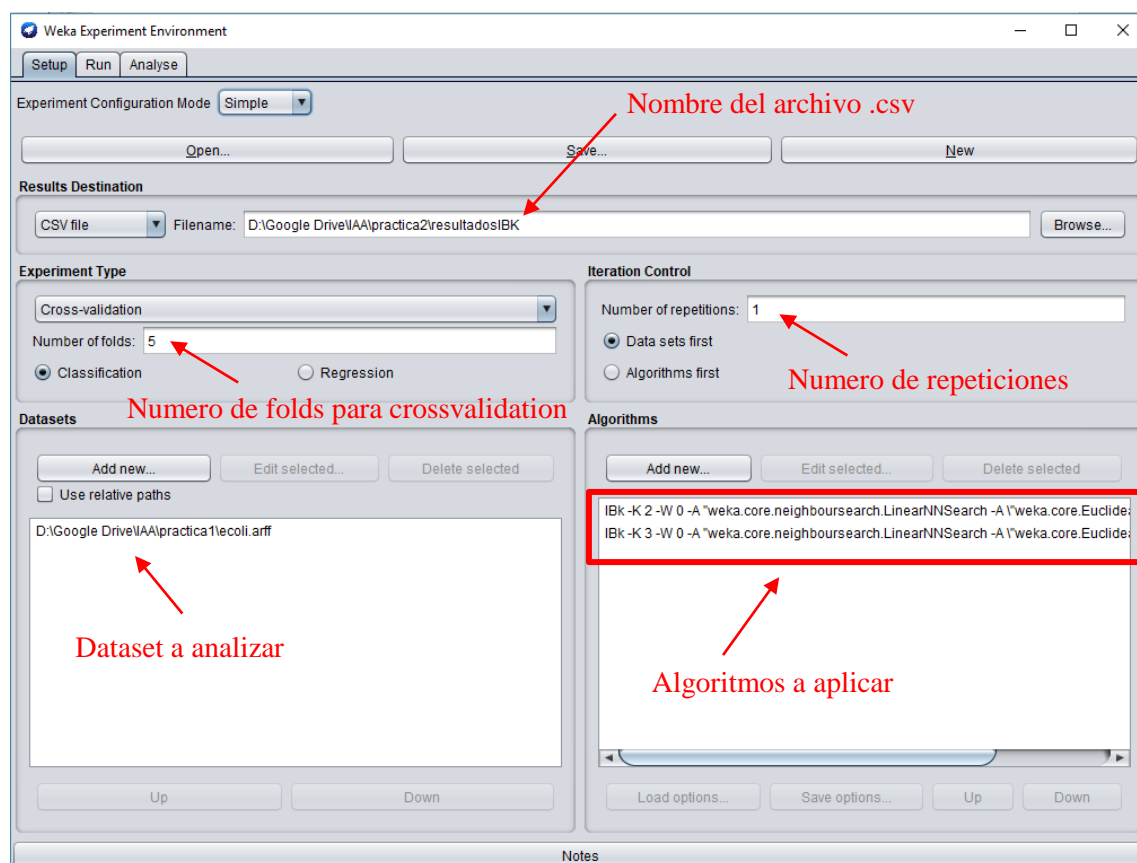


Figura 2.6: Configuración para ejecutar el algoritmo IBK

Una vez ejecutadas las pruebas, obtengo un fichero .csv con los resultados que Weka nos da. A continuación, voy a mostrar una tabla en la que se reflejara la media y desviación típica de las medidas: Accuracy o CCR, Kappa, RMSE y F-Measure.

2.2.2. 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.

K=2

	Accuracy	Kappa	RMSE	F-Measure
Media	83.05092	0.76107	0.18346	0.94675
Desv. Típica	4.79709	0.06885	0.02490	0.02145

Tabla 2.1: Métricas obtenidas para K=2

K=3

	Accuracy	Kappa	RMSE	F-Measure
Media	86.61545	0.81353	0.16992	0.96315
Desv. Típica	3.85605	0.05467	0.02076	0.02436

Tabla 2.2: Métricas obtenidas para K=3

Procedo ahora a analizar los resultados obtenidos teniendo en cuenta el valor de K:

- Accuracy: refleja el porcentaje de patrones bien clasificados. Se puede ver que, al aumentar el valor de K, es decir, al ampliar el vecindario, obtenemos un mayor número de patrones bien clasificados.

Seguramente al aumentar la K el algoritmo reflejará mejores resultados.

- Kappa: al igual que con la métrica Accuracy, con el estadístico Kappa ocurre lo mismo. Al aumentar el tamaño del vecindario aumenta el valor de Kappa. Hay que recordar que cuanto mayor sea este valor, mejor es el resultado ya que nos indica que hay una mayor concordancia respecto a lo que se esperaría por puro azar. Como pasaría con Accuracy, al ir aumentando el valor de K seguramente se obtendrán mejores resultados.
- RMSE: en cuanto al error medio cuadrático, no hay mucha diferencia entre una K y la otra. Aun así, se nota que el error es menor cuando la K es mayor, indicativo de que el modelo está clasificando mejor.
- F-Measure: esta métrica surge de la combinación de otras dos, Recall y Precision las cuales miden el porcentaje de patrones positivos clasificados como positivos. Se puede ver que obtenemos un valor algo más alto para la K=3 y ya que esta métrica hay que maximizarla, se considera que es mejor resultado que para K=2.

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.

2.3.1. 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).

Cargamos nuestra base de datos en Weka y ejecutamos el algoritmo Logistic con la siguiente configuración:

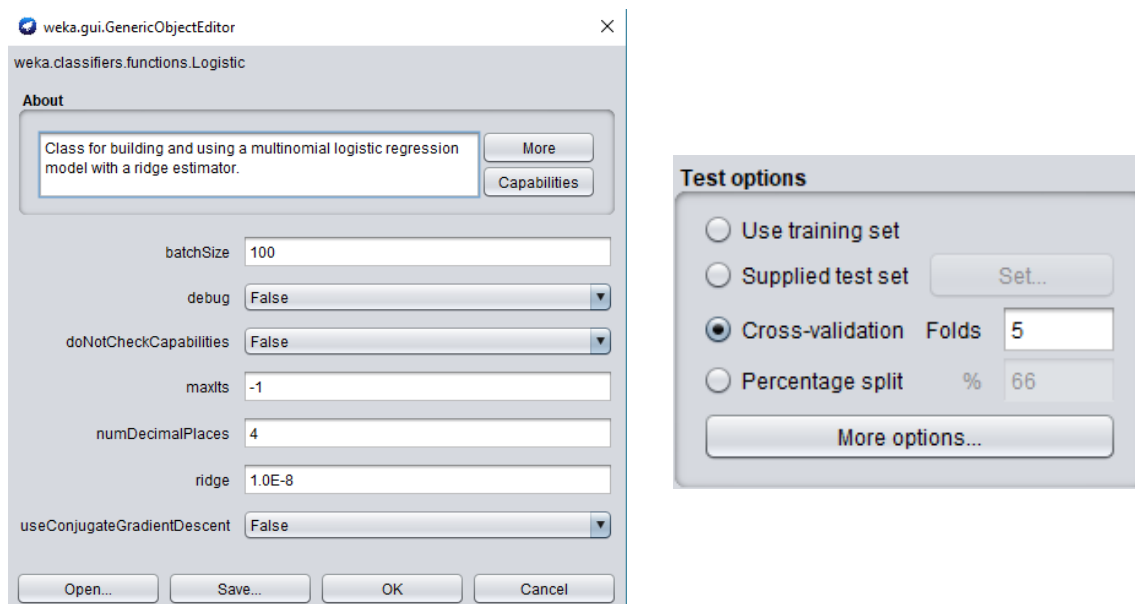


Figura 2.7: Configuración para ejecutar el algoritmo Logistic

Una vez ejecutado el algoritmo, el resultado podemos dividirlo en dos partes, en la primera tenemos los “Coefficients” y los “Odds Ratios” y en la segunda las métricas con su correspondiente matriz de confusión.

Primero vamos a analizar la primera parte de la regresión logística que hace referencia a la importancia de los atributos.

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	17.6508	-33.1551	21.6587	28.1532	91.3962	157.6524
chg	-44.8643	-25.0369	-17.3309	-10.8965	6.8927	68.2399	264.5702
aac	-2.6992	-3.5114	-2.1942	-3.9119	37.9813	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	4.519566394804493E13	1.2501170217589327E33
gvh	0.0252	6.5635	130609.2894	0.0427	243796853.5988	0	0
lip	0	46308026.8192	0	2548255005.257	1.6856182530970247E12	4.930254977866356E39	2.9346846815276627E68
chg	0	0	0	0	985.0365	4.3271527461844415E29	7.968821002813222E114
aac	0.0673	0.0299	0.1114	0.02	3.1265032360356044E16	25719626.1345	0
alm1	0	25188.1328	0.003	25.1183	0	7947611.628	0
alm2	6413.5211	130.746	0.0773	1315.1823	0	0	1180.9146

Figura 2.8: Coefficients y Odds Ratios para el modelo ejecutado

Lo primero que podemos destacar es que se nos muestra 7 clases en lugar de los 8 originales, esto es debido a que la última se calcula restando a 1 la suma de las probabilidades de que pertenezca al resto de clases. También podemos ver que tenemos dos tablas, una de “Coefficients” y otra de “Odds Ratios”.

Comenzaré analizando la tabla de “Coefficients”. Esta tabla representa el peso que tiene cada uno de los 7 atributos en el modelo obtenido antes de juntar todos. Esto quiere decir que, a mayor valor, mayor peso tendrá ese atributo a la hora de decidir si entra en una clase.

Por ejemplo, respecto a la clase “cp” podemos ver que el atributo que más peso tiene a la hora de decidir si un patrón pertenece a esta clase es el atributo “alm2” mientras que el atributo “chg” sería el menos importante.

La tabla “Odds Ratios” refleja como de grande es la influencia que supondría un cambio de un atributo en la predicción del modelo. Siguiendo con el ejemplo anterior de la clase “cp” podemos ver como el atributo “alm2” sigue siendo el más representativo ya que un cambio en ese atributo influiría mucho en la predicción. También cabe destacar que hay muchos atributos que tienen valor 0, esto quiere decir que son irrelevantes para las clases y que no van a influir en nada.

Por otra parte, tenemos las métricas y la matriz de confusión. Weka nos devuelve la siguiente información para su análisis:

```

=== 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

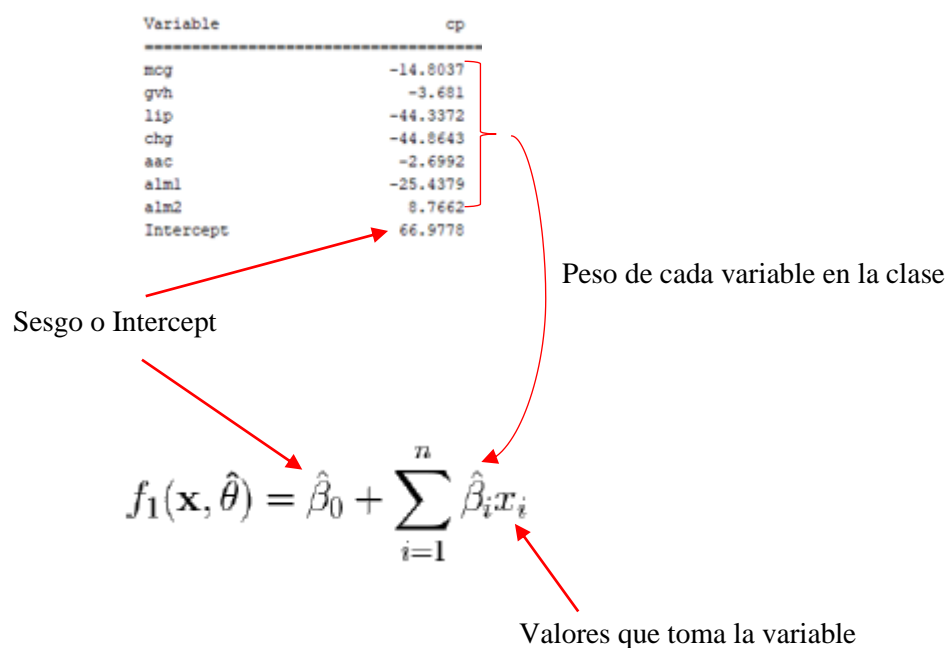
```

Figura 2.9: Métricas tras ejecutar el algoritmo Logistic

Para analizar las métricas primero voy a ejecutar el siguiente algoritmo, “SimpleLogistic”, para poder así hacer un mejor análisis.

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

Para la salida obtenida de coeficientes por el algoritmo Logistic, pasare a analizar cómo se calcularía la probabilidad de una clase (“cp”) con las fórmulas que muestro a continuación:



Con esto calculamos f_1 para poder aplicar la siguiente formula para calcular la probabilidad de la clase “cp”:

$$p_1(\mathbf{x}, \hat{\theta}) = \frac{e^{f_1(\mathbf{x}, \hat{\theta})}}{1 + \underbrace{\sum_{k=1}^{K-1} e^{f_k(\mathbf{x}, \hat{\theta})}}_{\text{Sumatorio de todos los } e^f \text{ de cada clase}}}$$

Red arrows indicate that $f_1(\mathbf{x}, \hat{\theta})$ in the numerator is the 'f1 calculado anteriormente para la clase “cp”' and the denominator's sum represents the 'Sumatorio de todos los e^f de cada clase'.

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.

2.4.1. 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).

Al igual que hicimos con el algoritmo Logistic, procedo a ejecutar SimpleLogistic con el dataset ecoli y 5 folds para el crossvalidation.

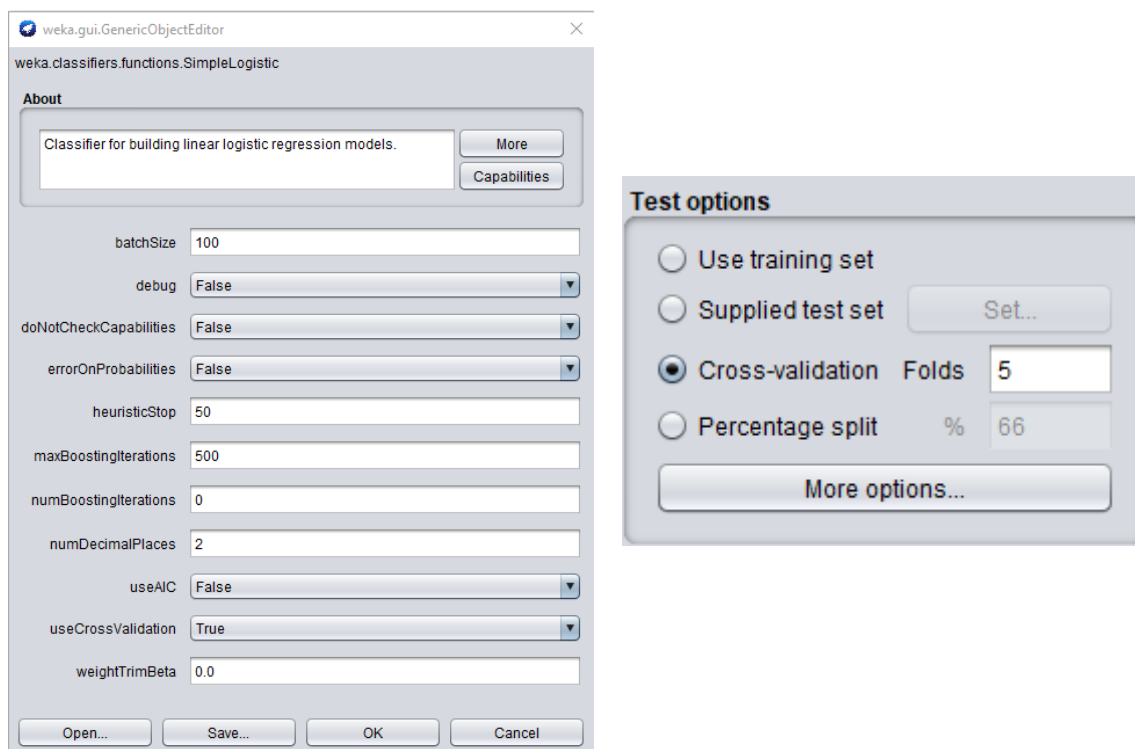


Figura 2.10: Configuración para ejecutar el algoritmo SimpleLogistic

Al ejecutar el algoritmo y como pasaba con Logistic, el resultado que nos muestra Weka podemos dividirlo en dos. En la primera parte tenemos el

análisis de los atributos por clase y seguidamente las metricas y matriz de confusión.

Lo primero que Weka nos muestra es como de influyentes son que atributos para cada clase y su peso en ella. Con ello se puede realizar un análisis mas exhaustivo de la relación de los atributos y las clases. A diferencia con el algoritmo Logistic, en este se muestran las 8 clases.

A continuación, muestro esa relación que he comentado anteriormente:

SimpleLogistic:

```
Class cp :  
11.47 +  
[mcg] * -6.7 +  
[gvh] * -4.41 +  
[alml] * -9.9
```

```
Class im :  
-3.37 +  
[mcg] * -5.31 +  
[alml] * 11.97
```

```
Class pp :  
-2.84 +  
[mcg] * 1.38 +  
[gvh] * 10.5 +  
[aac] * -4.87 +  
[alm2] * -1.8
```

```
Class imU :  
-6.52 +  
[mcg] * 4.84 +  
[gvh] * -2.61 +  
[alml] * 4.24 +  
[alm2] * 3.92
```

```
Class om :  
-11.77 +  
[gvh] * 6.52 +  
[aac] * 16.45 +  
[alm2] * -3.95
```

```
Class omL :  
-6.01 +  
[lip] * 10.81 +  
[alm2] * -5.84
```

```
Class imL :  
-14.32 +  
[lip] * 9.77 +  
[chg] * 7.19 +  
[alml] * 3.46
```

```
Class imS :  
-7.71 +  
[mcg] * 8.48
```

Como se puede observar, se va mostrando cada clase con los atributos que influyen para cada una de ellas. Cabe destacar que hay muchos atributos que ni se muestran ya que no aportan ningún tipo de información.

Hay varios casos a destacar como el atributo “alml” para la clase “im”, el atributo “gvh” para la clase “pp”, el atributo “acc” para la clase “om”, el atributo “lip” para la clase “omL” o el atributo “lip” para la clase “imL”.

A mayor numero, mayor será la influencia de este atributo para la predicción de la clase.

Remarcar también que el primer valor que aparece debajo de cada clase corresponde al sesgo.

Figura 2.11: Salida para SimpleLogistic en

Weka

Como dije en el apartado anterior, para analizar las métricas de ambos algoritmos voy a usar tablas que comparen las métricas de ambos y así ver mejor las diferencias entre ambos.

```

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

```

Figura 2.12: Métricas tras ejecutar el algoritmo SimpleLogistic

- CCR, Kappa, MAE, RMSE, RAE y RRSE

	CCR	Kappa	MAE	RMSE	RAE	RRSE
Logistic	86.607	0.815	0.047	0.164	25.899	54.361
SimpleLogistic	85.119	0.793	0.057	0.164	31.508	54.645

Tabla 2.3: Métricas obtenidas para el algoritmo Logistic y SimpleLogistic

- Métricas por clase
 - Logistic: **Amarillo**
 - SimpleLogistic: **Verde**

	cp		im		pp		imU		om		omL	
TP Rate	0.965	0.965	0.844	0.857	0.885	0.808	0.600	0.543	0.800	0.800	1.000	1.000
FP Rate	0.031	0.047	0.058	0,073	0.025	0.032	0.033	0,027	0.009	0,003	0.006	0,003
Precision	0.958	0.939	0.813	0,776	0.868	0.824	0.677	0,027	0.842	0,941	0.714	0,833
FMeasure	0.962	0.952	0.828	0,815	0.876	0.816	0.636	0,613	0.821	0,865	0.833	0,909
MCC	0.933	0.915	0.776	0,758	0.853	0.782	0.598	0,580	0.810	0,860	0.843	0,911
ROC Area	0.988	0.990	0.957	0,964	0.951	0.957	0.929	0,949	0.919	0,992	0.998	0,998
PRC Area	0.976	0.985	0.906	0,903	0.898	0.893	0.609	0,626	0.859	0,929	0.853	0,810

Tabla 2.4: Métricas por clase obtenidas para el algoritmo Logistic y SimpleLogistic

Como podemos observar en la primera tabla se ve que el algoritmo “Logistic” clasifica mejor el dataset ya que obtenemos un CCR o número de instancias bien clasificadas mayor que el que alcanzamos usando “SimpleLogistic”. Aunque la diferencia no es muy significativa ya que estamos hablando de que “Logistic” clasifica bien 291 de 336 instancias mientras que “SimpleLogistic” clasifica bien 286 de 336 instancias.

En cuanto al estadístico Kappa estamos en las mismas. Se obtiene un valor algo superior en Logistic, lo que indica mayor concordancia en el conjunto de datos que con SimpleLogistic.

En el error medio absoluto obtenemos un valor más bajo para el algoritmo Logistic mientras que en error medio cuadrático obtenemos prácticamente el mismo valor.

Los demás errores, expresados en porcentaje, obtenemos prácticamente los mismos valores, aun así, el algoritmo Logistic es el que tiene un error más bajo.

Luego se podría decir que usando una Regresión Logística con el método de máxima verosimilitud se obtendría un mejor modelo para el dataset ecoli que con el uso de la Regresión Logística Simple.

En la segunda tabla se puede ver un análisis más exhaustivo de las métricas por clase el cual sigue reflejando un mejor modelo alcanzado con Logistic.

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

Para la salida obtenida por el algoritmo SimpleLogistic, pasare a analizar cómo se calcularía la probabilidad de una clase (“cp”) con las fórmulas que muestro a continuación:

Class cp :
11.47 +
[mcg] * -6.7 +
[gvh] * -4.41 +
[a1ml] * -9.9

Sesgo

Peso de cada variable en la clase

Valores de la variable

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

Con esto calculamos f_1 para poder aplicar la siguiente fórmula para calcular la probabilidad de la clase “cp”:

f_1 calculado anteriormente para la clase “cp”

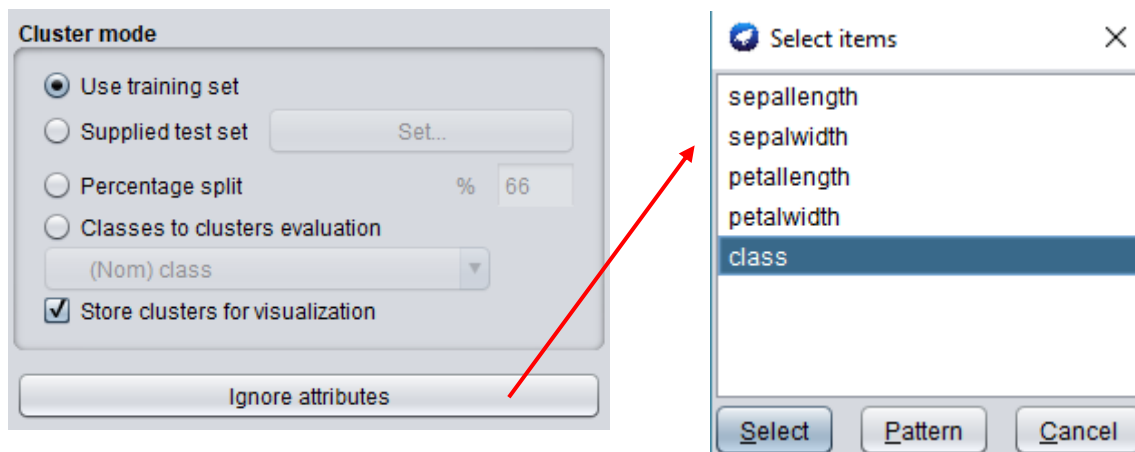
Sumatorio de todos los e^f de cada clase

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

3. Práctica 2-2: Clustering con Weka

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.

En el entorno Explorer, una vez cargada la base de datos Iris nos dirigimos a la pestaña Cluster. Una vez dentro de esta sección elegimos el algoritmo K-means que en Weka se llama SimpleKMeans. Pasamos seguidamente a elegir la opción “Use training mode” y a ignorar el atributo de clase como se muestra en la siguiente imagen:



*Figura 3.1: Configuración para el algoritmo
K-means ignorando la clase*

Paso ahora a lanzar las pruebas que consistirán en fijar el número de clústeres en 2, 3 y 4 y posteriormente analizar los resultados. Voy a lanzar primero las 3 pruebas mostrando los resultados que refleja Weka para posteriormente ir comparando cada una de las 3.

En rojo está marcado el SSE, en verde está marcado las coordenadas iniciales de los centroides, en azul está marcado las coordenadas finales de los centroides y en naranja el número de instancias por clúster.

- Número de clusters = 2

```

Within cluster sum of squared errors: 12.143688281579722

Initial starting points (random):

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

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute      Full Data      Cluster#
                (150.0)      (100.0)      (50.0)
=====
sepalwidth     5.8433         6.262        5.006
sepalwidth     3.054          2.872        3.418
petalwidth     3.7587         4.906        1.464
petalwidth     1.1987         1.676        0.244

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

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

Clustered Instances

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

```

*Figura 3.2: Salida para el algoritmo K-means
con n° clusters a 2*

- Número de clusters = 3

Within cluster sum of squared errors: 6.998114004826762

Initial starting points (random):

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

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Full Data (150.0)	Cluster#		
		0 (61.0)	1 (50.0)	2 (39.0)
sepalength	5.8433	5.8885	5.006	6.8462
sepalwidth	3.054	2.7377	3.418	3.0821
petallength	3.7587	4.3967	1.464	5.7026
petalwidth	1.1987	1.418	0.244	2.0795

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

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

Clustered Instances

0 61 (41%)
1 50 (33%)
2 39 (26%)

*Figura 3.3: Salida para el algoritmo K-means
con n° clusters a 3*

- Número de clusters = 4

Within cluster sum of squared errors: 5.532831003081898

Initial starting points (random):

```
Cluster 0: 6.1,2.9,4.7,1.4
Cluster 1: 6.2,2.9,4.3,1.3
Cluster 2: 6.9,3.1,5.1,2.3
Cluster 3: 5.5,4.2,1.4,0.2
```

Missing values globally replaced with mean/mode

Final cluster centroids:

Attribute	Cluster#				
	Full Data (150.0)	0 (42.0)	1 (29.0)	2 (29.0)	3 (50.0)
sepal.length	5.8433	6.25	5.5828	6.9586	5.006
sepal.width	3.054	2.9	2.569	3.1345	3.418
petal.length	3.7587	4.8738	4.0034	5.8552	1.464
petal.width	1.1987	1.6405	1.231	2.1724	0.244

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

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

Clustered Instances

```
0      42 ( 28%)
1      29 ( 19%)
2      29 ( 19%)
3      50 ( 33%)
```

*Figura 3.3: Salida para el algoritmo K-means
con n° clusters a 4*

La primera métrica a analizar sería el SSE o “Sum of Squared Error”. Es la suma total de distancias al cuadrado de los puntos asignados a su centroide y hay que intentar minimizarla. Como se puede ver conforme aumentamos el número de clúster, esta va minimizando. Podemos observar cómo se produce un cambio drástico al pasar de 2 a 3 clúster ya que se reduce en la

mitad ese SSE luego podríamos decir que un valor adecuado de clústeres es 3.

Pasemos ahora a ver las coordenadas de los centroides. Las coordenadas de los centroides iniciales son aleatorias ya que es el propio algoritmo el que las define. Por otro lado, tenemos las coordenadas finales de los centroides una vez finalizado el clustering. Estos clústeres se podrán ver con mejor detalle cuando los muestre gráficamente.

Por último, podemos ver la cantidad de instancias por clúster tras la ejecución del algoritmo K-means.

Ahora pasamos a visualizar los clusters resultantes de representar los atributos “petallength” y “petalwidth”.

- Número de clusters = 2

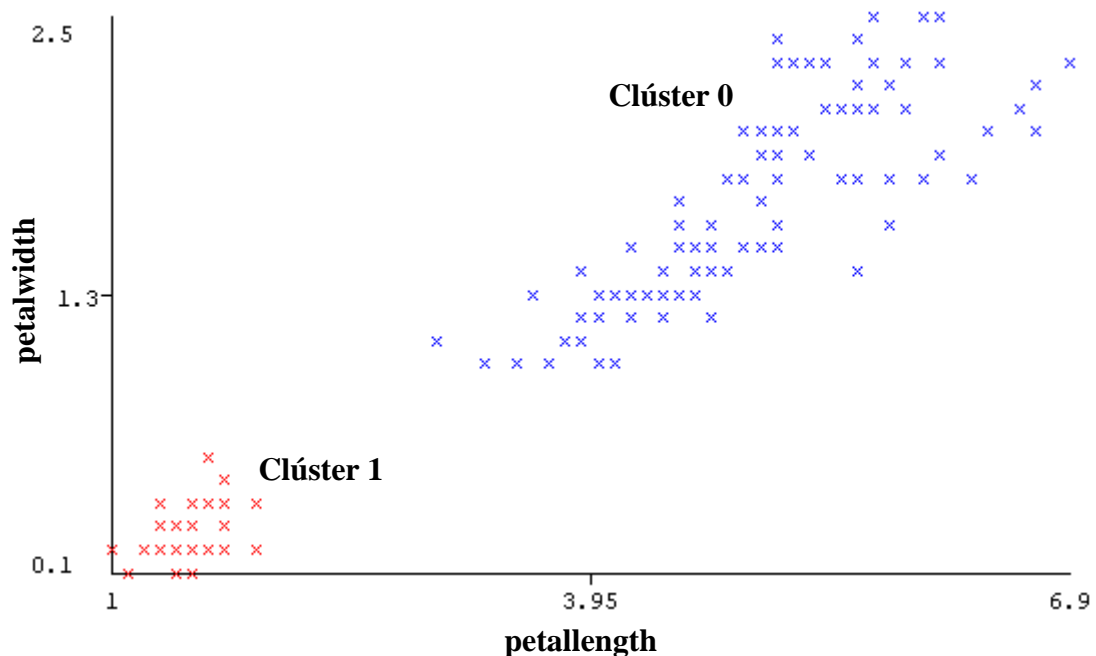


Figura 3.4: Representación gráfica de clusters para n° clusters a 2

Ahora podemos ver gráficamente los clústeres creados por nuestro modelo según los atributos “petallength” y “petalwidth”. Como hemos podido ver anteriormente en la figura para el numero de clústeres a 2, las coordenadas de los centroides finales ambos atributos son:

- Clúster 0: (4.9 , 1.6)
- Clúster 1: (1.4 , 0.2)

Lo cual se puede ver en la *figura 3.4*. ya que ambos centroides coinciden con las coordenadas de la imagen.

- Número de clusters = 3

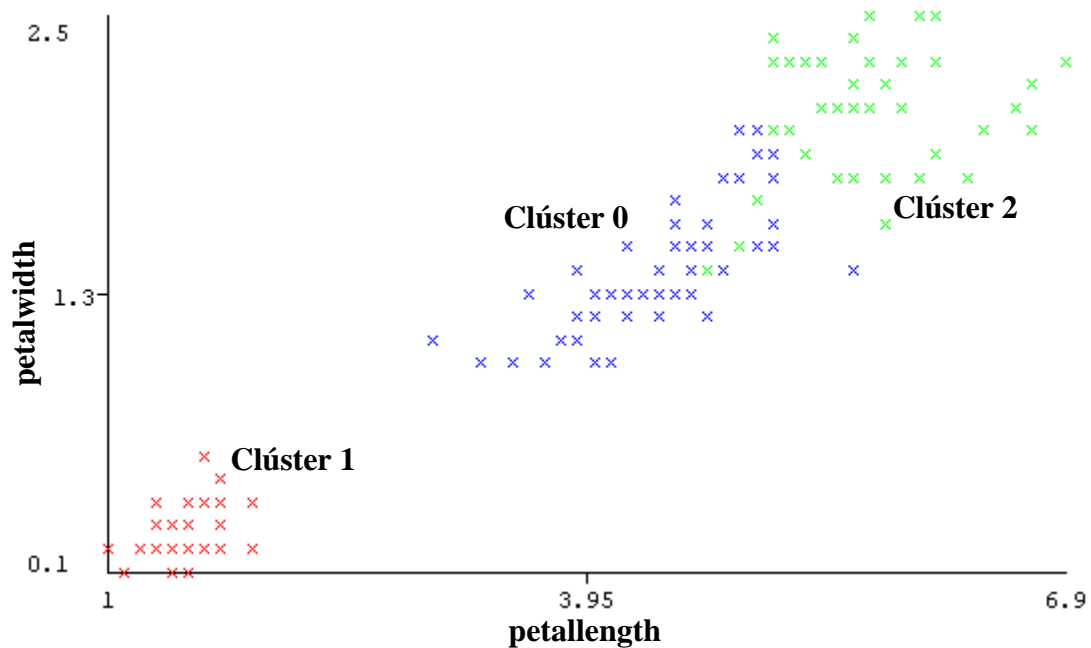


Figura 3.5: Representación gráfica de clusters para n° clusters a 3

Las coordenadas de los centroides de los clústeres en este caso serian:

- Clúster 0: (4.3 , 1.4)
- Clúster 1: (1.4 , 0.2)
- Clúster 2: (5.7 , 2.0)

- Número de clusters = 4

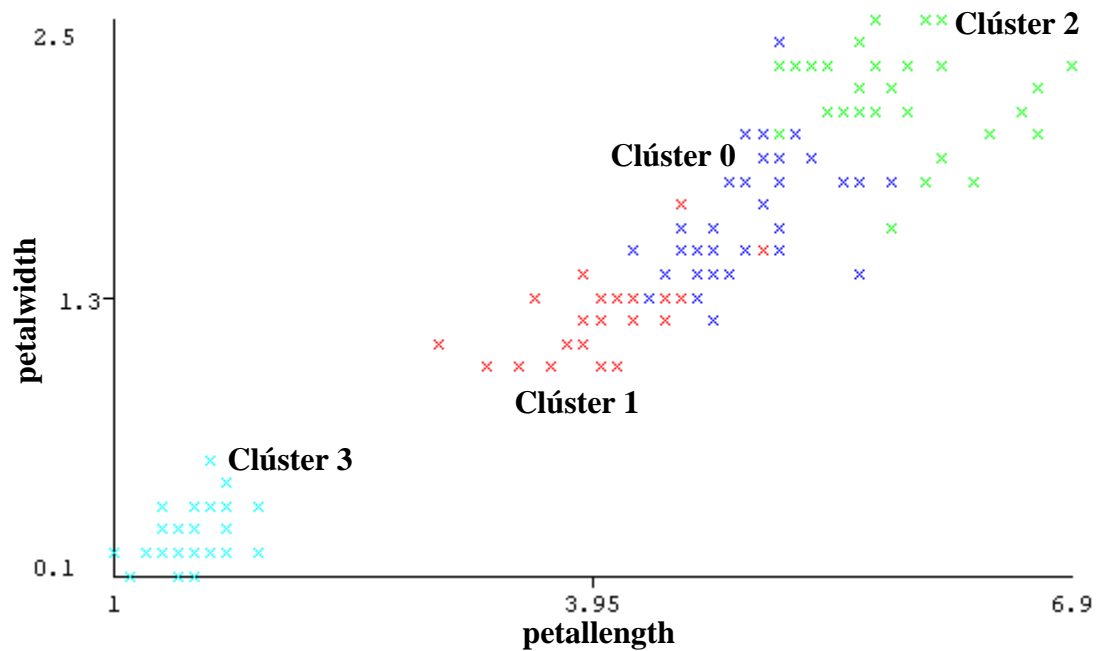


Figura 3.6: Representación gráfica de clusters para n° clusters a 4

Las coordenadas de los centroides de los clústeres en este caso serían:

- Clúster 0: (4.8 , 1.6)
- Clúster 1: (4.0 , 1.2)
- Clúster 2: (5.8 , 2.1)
- Clúster 3: (1.4 , 0.2)

Tras ver todas las imágenes, puedo afirmar que el mejor resultado más óptimo se obtendría con un número de clústeres de 3 ya que los clústeres están bien definidos y no se entremezclan tanto entre sí como es el caso de 4 clústeres.

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.

Una vez cargada la base de datos ecoli, procedo a la pestaña clúster para aplicar el algoritmo K-means con un numero de clúster igual al número de clases de nuestro problema el cual es 8.

Voy a ejecutar el algoritmo ignorando el atributo de clase y sin ignorarlo, los mostraré y después procederé a comentarlo. A el resultado le aplico un poco de ruido para que sean más apreciables los clústeres.

En la siguiente página se pueden apreciar dos gráficos de las dos ejecuciones del algoritmo, el primero teniendo en cuenta el atributo class y el segundo sin tenerlo. Los gráficos corresponden a la relación clase/clúster.

Se puede ver una clara diferencia entre los dos, en el que se ignora el atributo class, los clústeres obtenidos están muy dispersos. Esto se debe a que la clase a la que pertenece cada atributo tiene un peso considerable a la hora de realizar un clustering. Sin embargo esto no quiere decir que el modelo adecuado sea en el que se tiene en cuenta la clase, tal como veremos a continuación.

A parte del resultado gráfico, también se puede ver una diferencia analizando las métricas que Weka nos muestra. El SSE del modelo en el que se no se ignora la clase es de 48.26 mientras que el SSE del modelo en él no se tiene en cuenta es de 18.56 lo cual quiere decir que la similitud de los elementos de un clúster en el segundo modelo es mucho mayor que en del segundo con lo que se podría decir que el modelo en el que la clase es ignorada es mejor que en el que la clase no lo es.

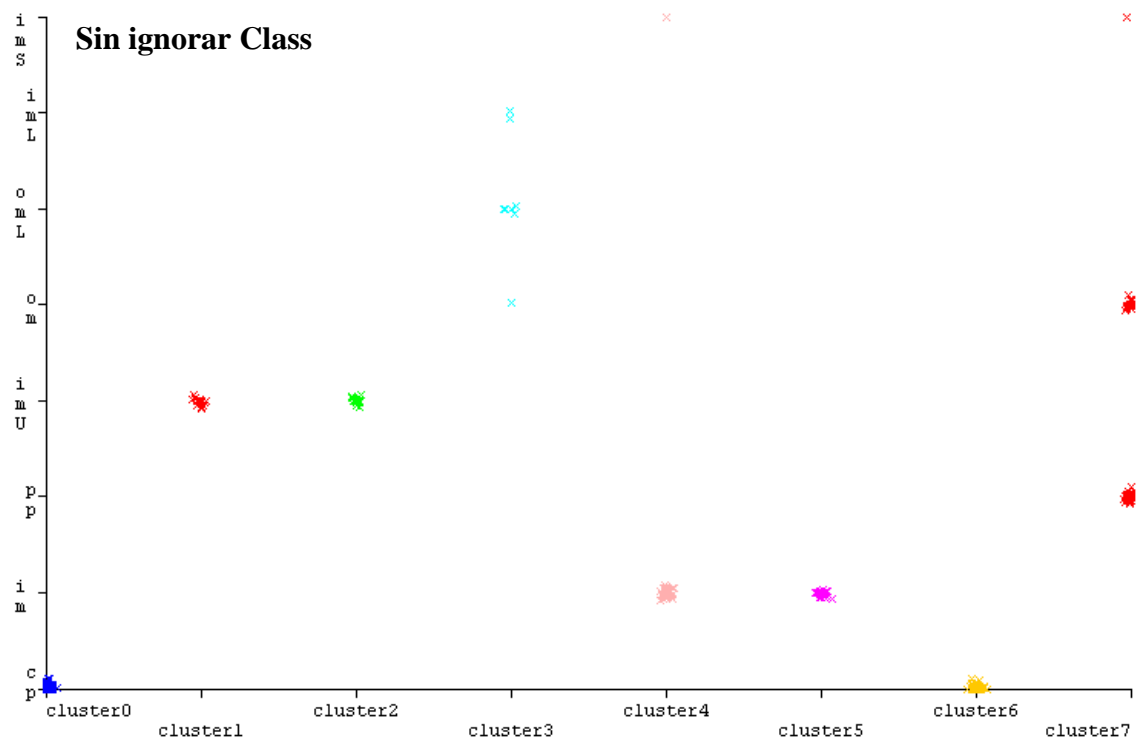


Figura 3.7: Representación gráfica de clusters para K-means sin ignorar class

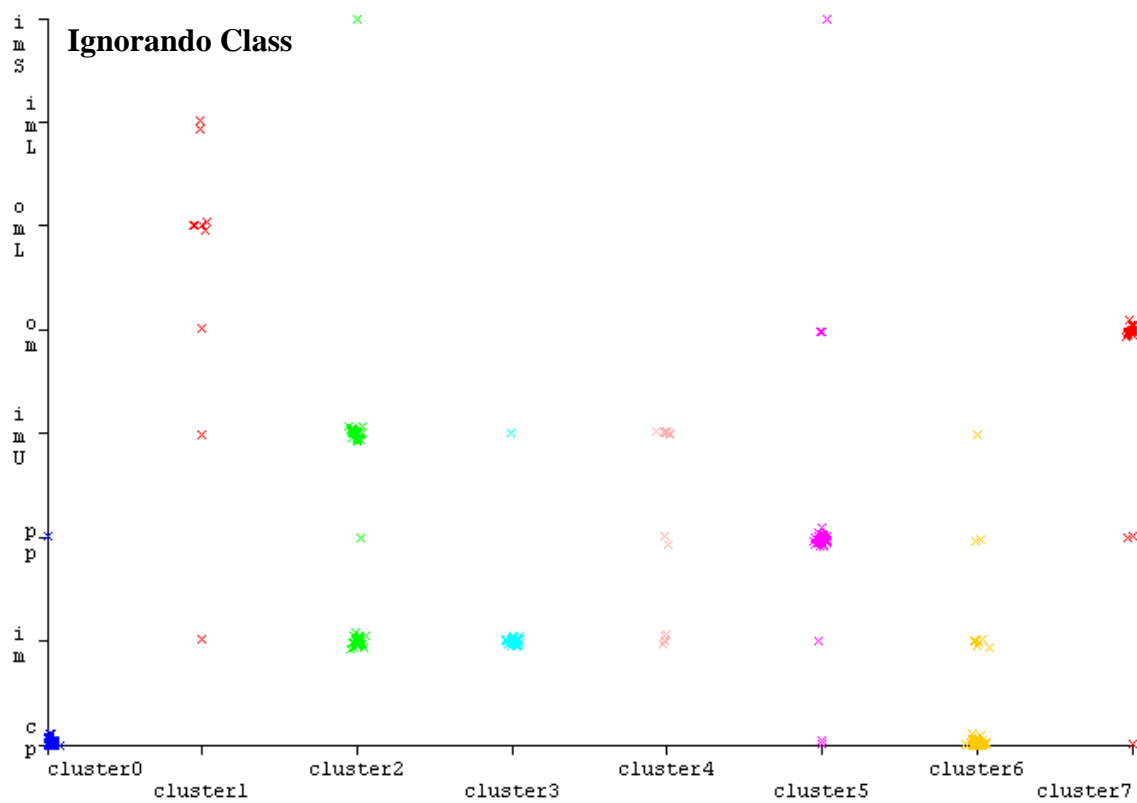


Figura 3.8: Representación gráfica de clusters para K-means ignorando class

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?

Al ser la inicialización de centroides de forma aleatoria se pueden dar que los centroides se inicialicen en los patrones con lo que cada instancia se agruparía individualmente en un clúster, es decir, que habría un clúster por cada patrón, dando lugar a un sobretratamiento.

Todo esto va a depender de la generación aleatoria de los centroides.

3.4. Ejercicio 4: Utilizando su base de datos, el algoritmo K-means, y seleccionando la variable de clase, ¿con qué número de clusters y con qué métrica de distancia obtiene mejores resultados?

La configuración para llevar a cabo las pruebas ha sido:

- Atributo Class ignorado ya que como he comprobado anteriormente, se obtiene un mejor SSE.
- Seeds con valores 10,20 y 30.
- El número de clústeres ha sido configurado para ver cómo se comporta el modelo cuando el número de clústeres es menor, igual y mayor que el número de clases.

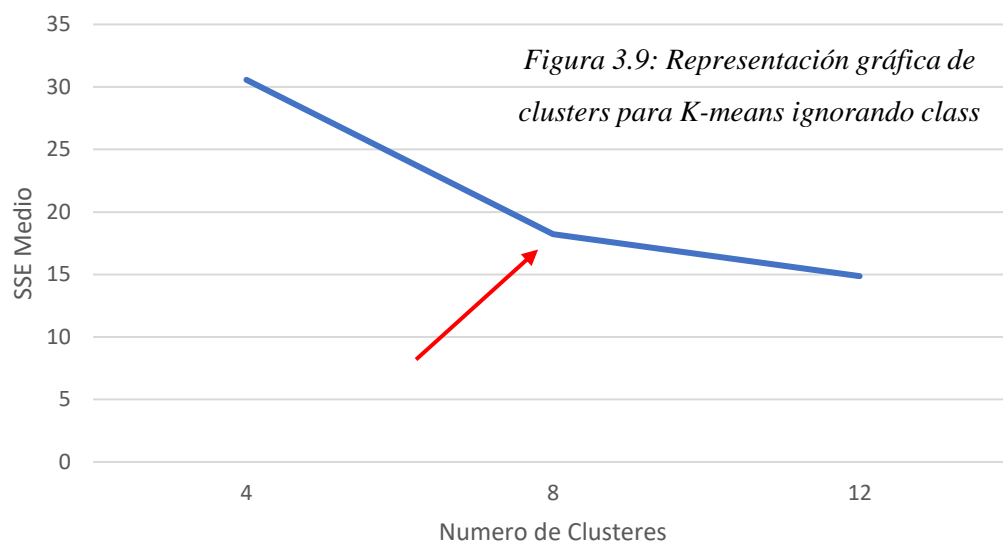
Numero de Clusters	Función de Distancia	SSE Medio
4	Euclídea	30.57161
4	Manhattan	155.49635
8	Euclídea	18.22665
8	Manhattan	132.87791
12	Euclídea	14.86375
12	Manhattan	119.98693

*Tabla 3.1: Resultados de las pruebas
variando el número de clusters*

Como se puede observar en la tabla, el mejor valor de SSE se obtiene con el mayor número de clústeres. Esto no quiere decir que 12 sea el mejor número de clústeres ya que el valor más adecuado viene dado cuando se produce un codo (marcado en rojo en la gráfica) en el valor del SSE y este codo se da al pasar de 4 a 8 clústeres ya que el valor del SSE baja drásticamente en comparación de como baja de 8 a 12.

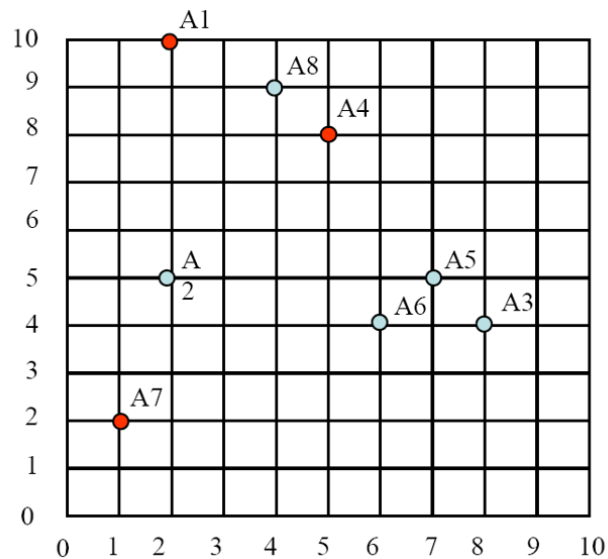
Luego puedo afirmar que el mejor valor se obtiene con un numero de clústeres de 8.

En cuanto a la métrica de distancia se puede ver claramente que el mejor valor de SSE se obtiene usando la distancia euclídea ya que esta es la menor distancia entre dos puntos.

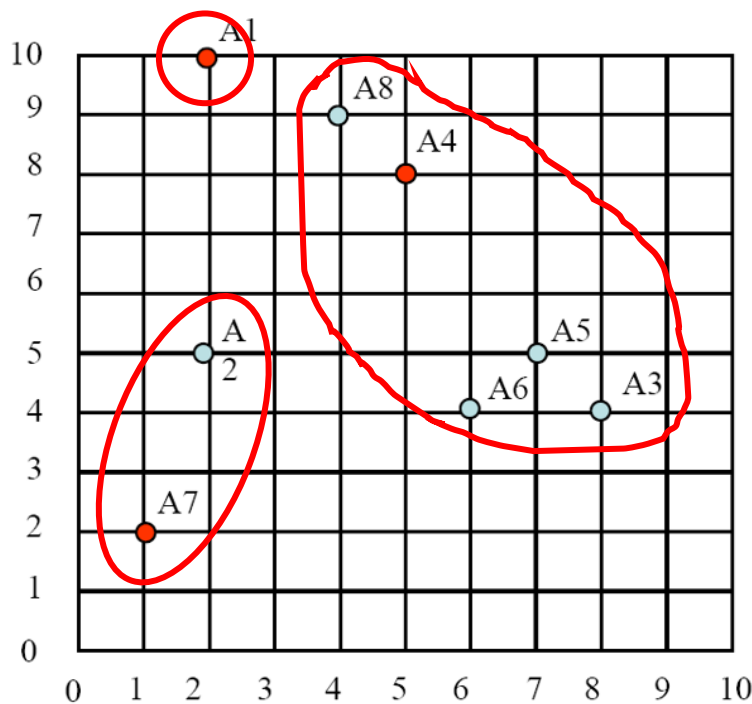


3.5. Ejercicio 5: Agrupe los 8 puntos de la figura en 3 clusters usando el algoritmo K-means. Los centroides iniciales se encuentran sobre los puntos A1, A4 y A7.

Se nos presentan el siguiente problema, en el que los centroides iniciales son marcados en rojo:



• **Primer agrupamiento**

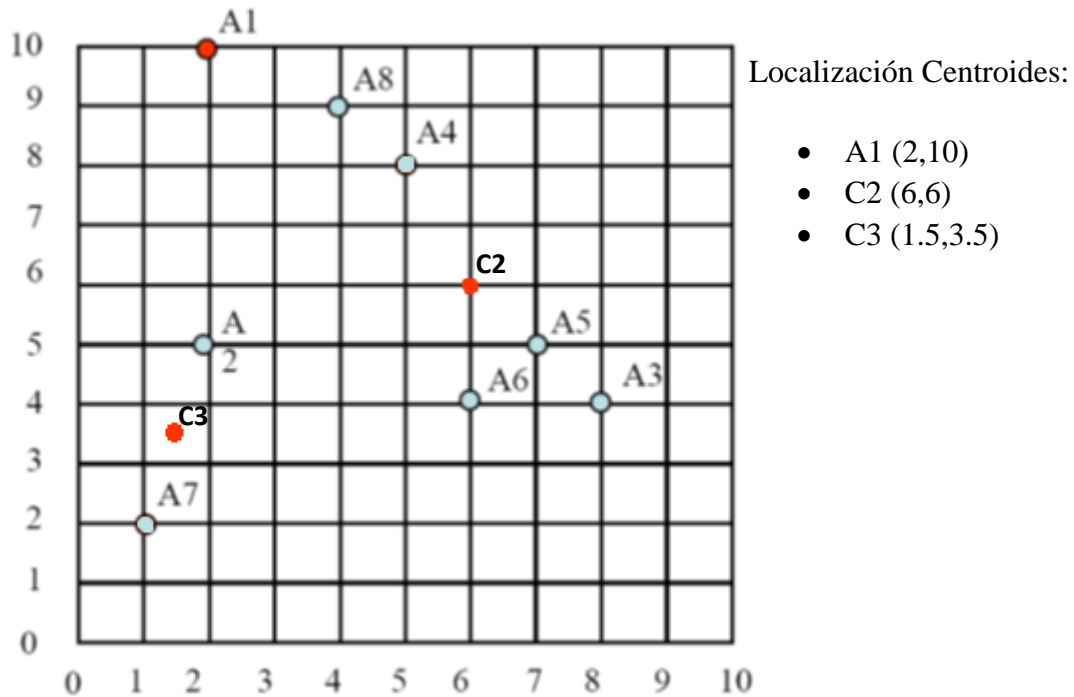


Localización Centroides:

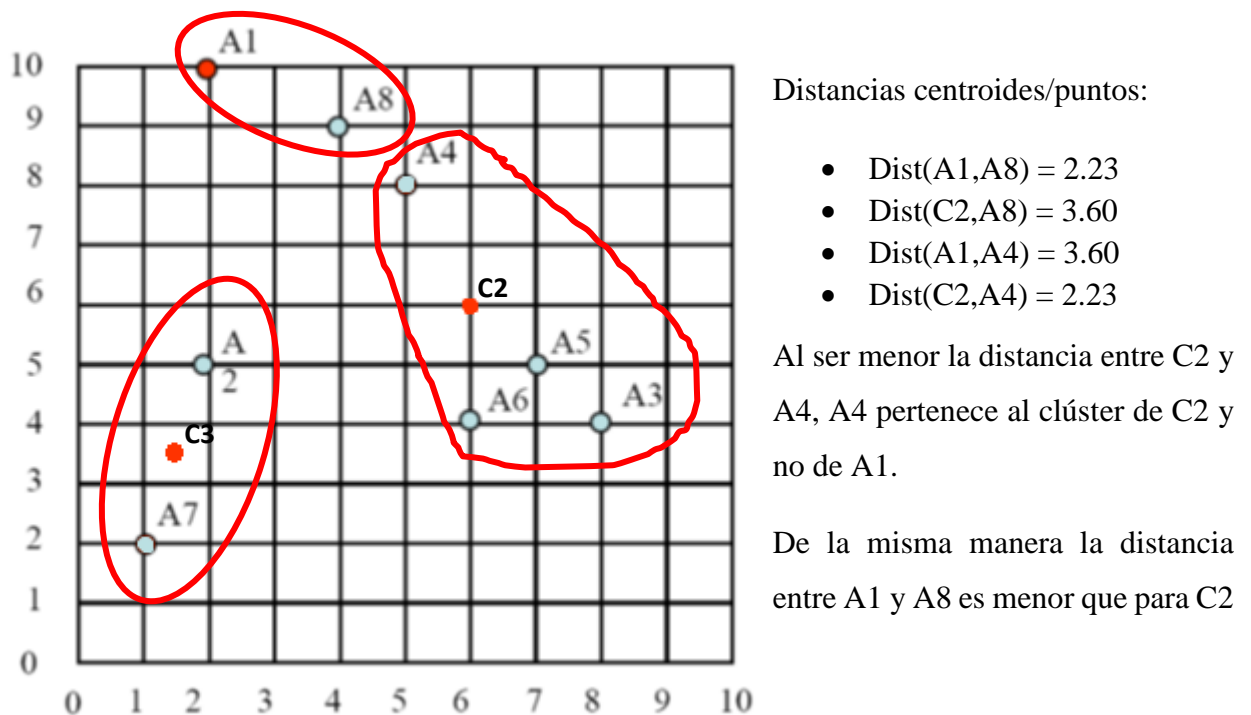
- A1 (2,10)
- A4 (5,8)
- A7 (1,2)

- **Primera iteración**

Se calcula la nueva localización de los centroides mediante el cálculo de la media de sus coordenadas:

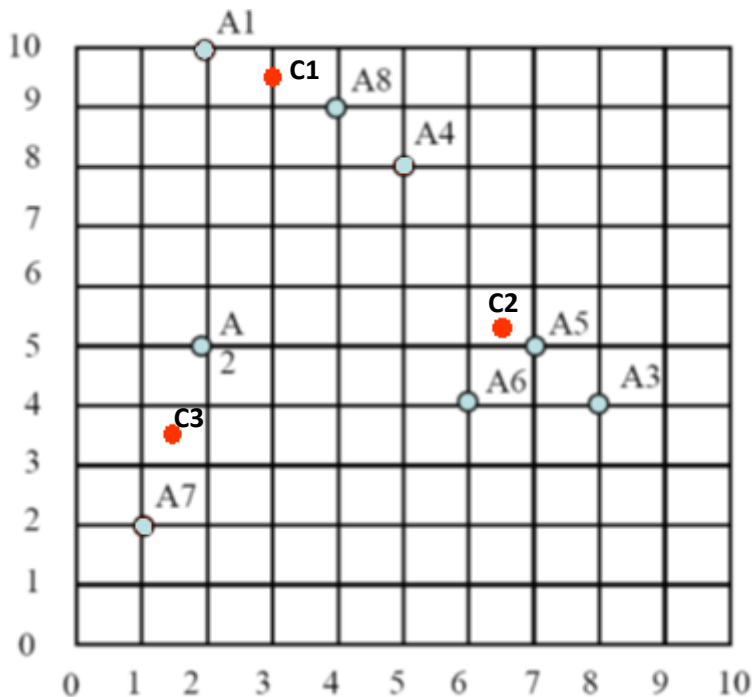


Una vez calculados los nuevos centroides, procedo a calcular las distancias euclídeas de los centroides a los puntos para agruparlos:



- **Segunda iteración**

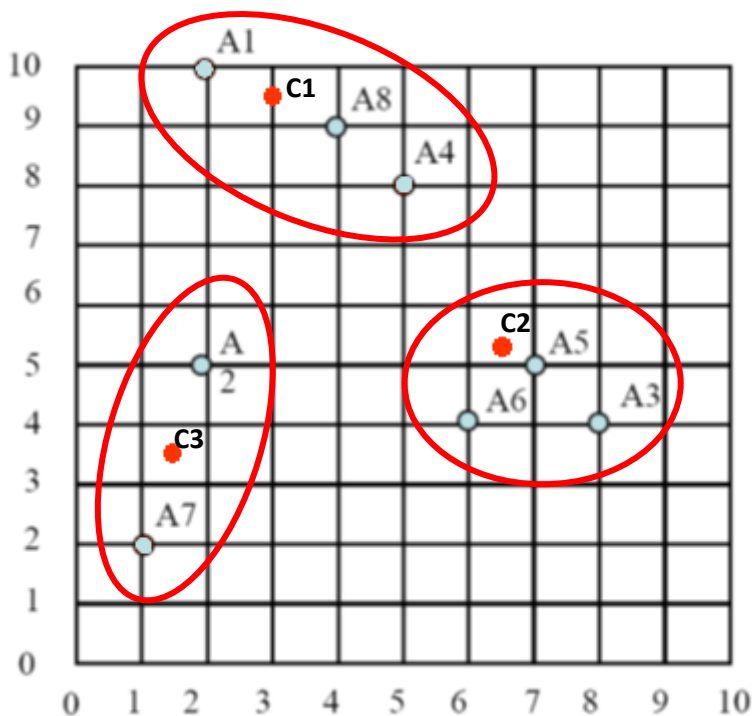
Se calcula la nueva localización de los centroides:



Localización Centroides:

- C1 (3,9.5)
- C2 (6.5,5.25)
- C3 (1.5,3.5)

Se vuelven a ajustar los centroides C2 y C1 menos el C3 ya que no tiene otro punto más cercano al cual añadir a su clúster así que se podría decir que el clúster 3 ya habría acabado. Sin embargo, tenemos que agrupar los nuevos centroides.



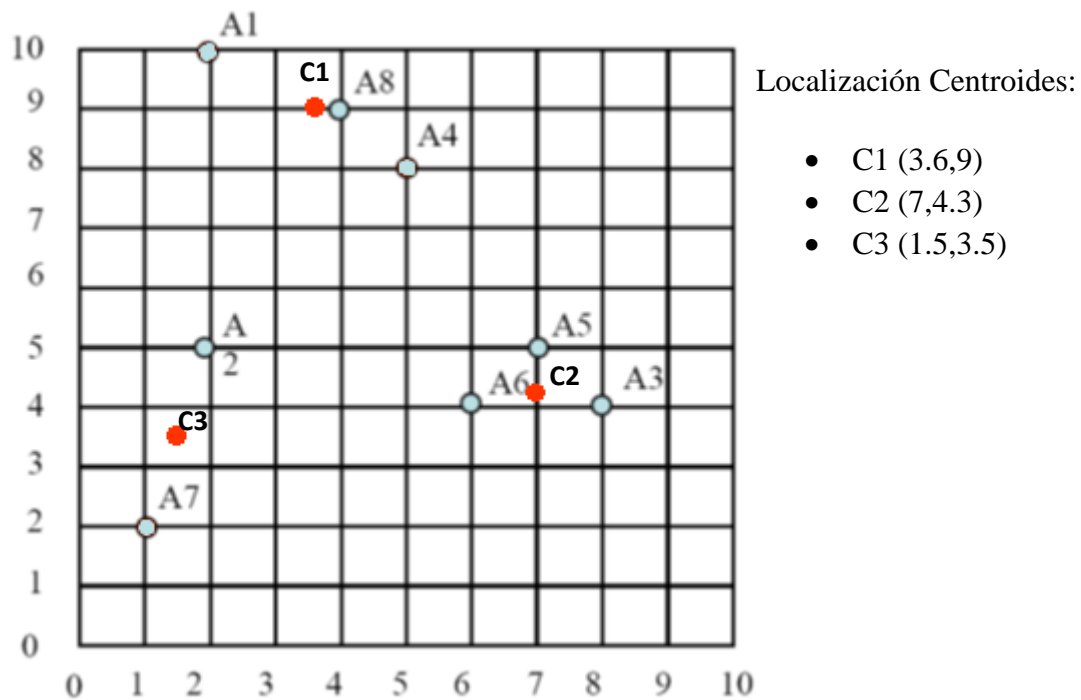
Distancias centroides/puntos:

- $\text{Dist}(C2, A4) = 3.13$
- $\text{Dist}(C1, A4) = 2.5$

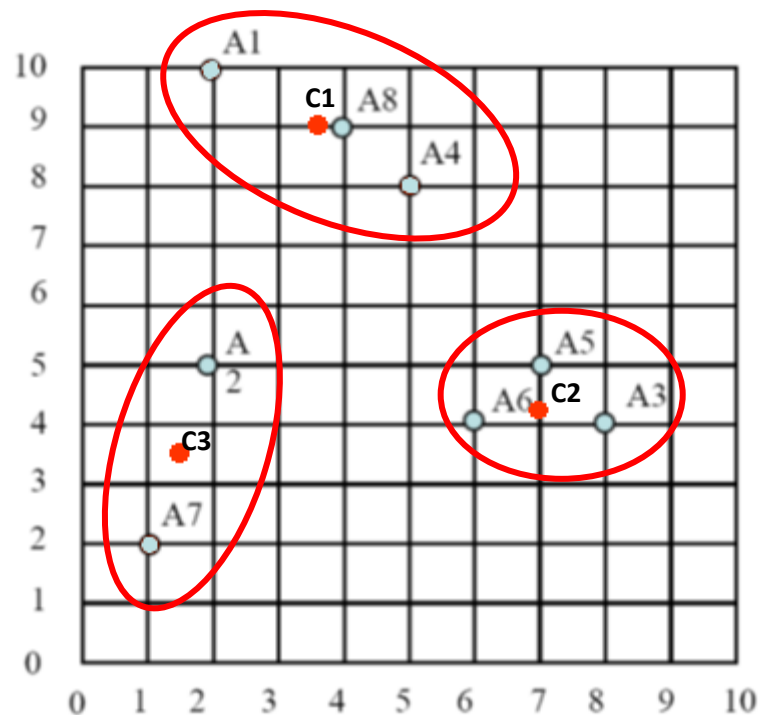
En este caso el punto A4 pasaría a pertenecer al clúster 1 ya que su distancia es menor que la que tiene con el centroide 2.

- Tercera iteración

Se calcula la nueva localización de los centroides:



Se vuelven a ajustar por última vez los centroides C1 y C2 ya que como se puede apreciar tras una siguiente iteración no se produciría ningún cambio quedando así los clústeres finales:



3.6. Ejercicio 6: Ejecute el algoritmo HierarchicalClusterer sobre su base de datos, usando la distancia Euclidea e indicando como parámetro que se corte el dendograma en un número de clusters igual al número de clases de su problema, sin emplear la etiqueta de clase.

Una vez cargada la base de datos Ecoli, pasamos a la pestaña Clúster y aplicamos el algoritmo jerárquico “HierarchicalClustered” con la configuración de la imagen cambiando los individuos seleccionados para el cálculo de la distancia o linkType:

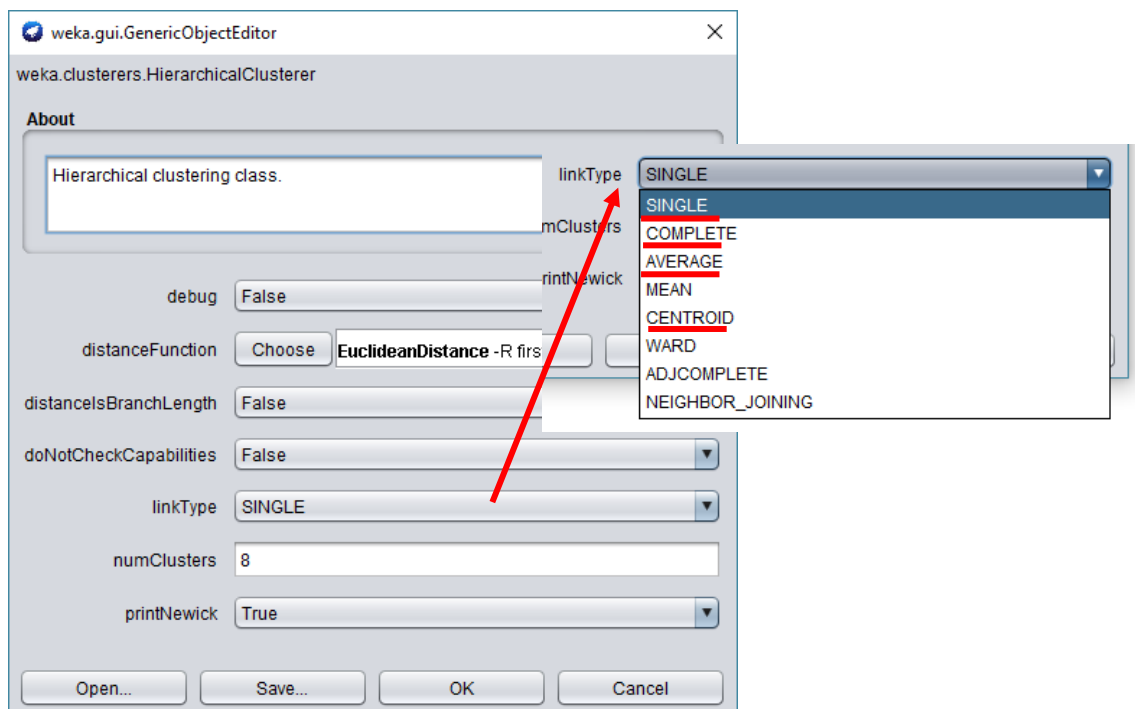


Figura 3.10: Configuración para hacer las pruebas de HierarchicalClustered

Una vez ejecutado el algoritmo variando el parámetro linkType a los valores subrayados en rojo, voy a mostrar en una tabla el porcentaje de instancias mal clasificadas para un mejor análisis:

	SINGLE	COMPLETE	AVERAGE	CENTROID
Instancias mal clasificadas	185 55.05%	126 37.50%	86 25.59%	76 22.61%

*Tabla 3.2: Resultados de las pruebas
variando el linkType*

Como podemos observar gracias a la tabla, el mejor resultado se obtiene para un linkType con método CENTROID ya que de 336 instancias solamente ha clasificado mal 76 que, comparándolo con los otros métodos, es el mejor resultado.

Ahora voy a mostrar la asignación de los clusters que se han creado dependiendo del tipo de linkType.

0	324 (96%)	0	93 (28%)	0	133 (40%)	0	153 (46%)
1	1 (0%)	1	60 (18%)	1	82 (24%)	1	70 (21%)
2	3 (1%)	2	74 (22%)	2	7 (2%)	2	100 (30%)
3	1 (0%)	3	97 (29%)	3	102 (30%)	3	1 (0%)
4	1 (0%)	4	4 (1%)	4	3 (1%)	4	3 (1%)
5	1 (0%)	5	2 (1%)	5	2 (1%)	5	2 (1%)
6	1 (0%)	6	1 (0%)	6	1 (0%)	6	1 (0%)
7	4 (1%)	7	5 (1%)	7	6 (2%)	7	6 (2%)
SINGLE		COMPLETE		AVERAGE		CENTROID	

*Figura 3.11: Asignaciones de clusters según
linkType*

3.7. Ejercicio 7: Utilice el algoritmo de clustering jerárquico aglomerativo para agrupar los datos descritos por la siguiente matriz de distancias. Resuelva el ejercicio con Simple linkage y Complete linkage para distancia entre clusters y compare los resultados.

- **Simple Linkage**

	A	B	C	D
A	0	1	4	5
B		0	2	6
C			0	3
D				0

Figura 3.12: Matriz de distancias del problema

En la primera iteración la distancia más pequeña entre objetos es la distancia entre A y B que es de 1 (señalado en rojo) luego agrupamos ambos objetos en un clúster y recalculamos la matriz de distancia con las distancias euclídeas:

- $\text{dist}((A,B),C) = \min(\text{dist}(A,C), \text{dist}(B,C)) = \min(4,2) = 2$
- $\text{dist}((A,B),D) = \min(\text{dist}(A,D), \text{dist}(B,D)) = \min(5,6) = 5$

	(A,B)	C	D
(A,B)	0	2	5
C		0	3
D			0

Figura 3.13: Matriz de distancias tras la primera iteración

En la segunda iteración, la distancia mínima entre objetos es la distancia entre (A,B) y C (señalado en rojo), luego se agrupa en un clúster y se vuelven a calcular las distancias entre objetos:

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

	((A,B),C)	D
((A,B),C)	0	3
D		0

Figura 3.14: Matriz de distancias tras la segunda iteración

La jerarquía alcanzada con Simple Linkage es:

((A,B),C),D)

El dendograma quedaría tal como se muestra en la *figura 3.15*:

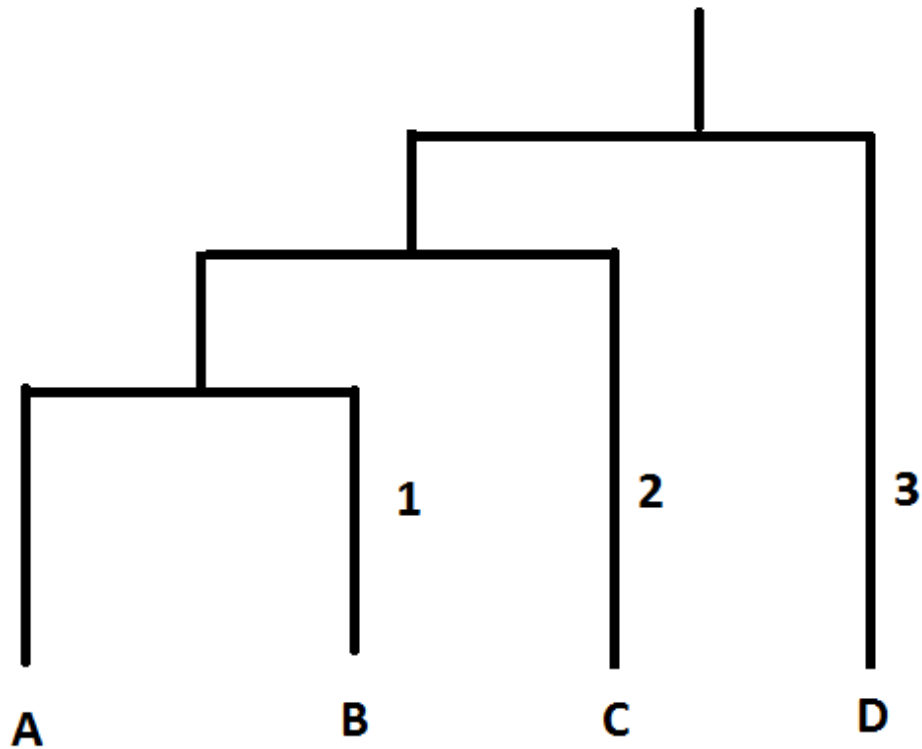


Figura 3.15: Dendograma de SimpleLinkage

- **Complete Linkage**

	A	B	C	D
A	0	1	4	5
B		0	2	6
C			0	3
D				0

Figura 3.16: Matriz de distancias del problema

Al contrario que con el Simple Linkage, ahora con el Complete Linkage se va a coger la distancia máxima entre dos objetos en vez de la mínima para recalcular la matriz de distancias. Aun así, seguimos cogiendo la distancia mínima entre objetos para agrupar (en rojo):

- $\text{dist}((A,B),D) = \max(\text{dist}(D,B), \text{dist}(D,A)) = \max(6,5) = 6$
- $\text{dist}((A,B),C) = \max(\text{dist}(C,A), \text{dist}(C,B)) = \max(4,2) = 4$

	(A,B)	C	D
(A,B)	0	4	6
C		0	3
D			0

Figura 3.17: Matriz de distancias tras la primera iteración

En la segunda iteración, la mínima distancia entre objetos es la que hay entre el objeto C y D luego los agrupamos en un nuevo clusters y recalculamos la matriz de distancias:

- $\text{dist}((A,B),(C,D)) = \max(\text{dist}(A,C), \text{dist}(A,D), \text{dist}(B,C), \text{dist}(B,D)) = \max(4,5,2,6) = 6$

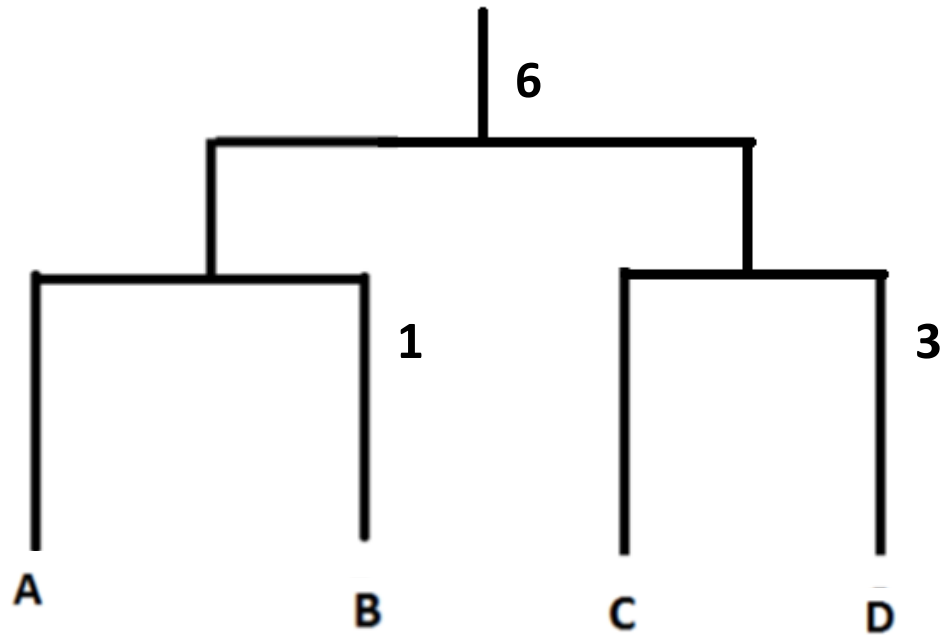
	(A,B)	(C,D)
(A,B)	0	6
(C,D)		0

Figura 3.18: Matriz de distancias tras la segunda iteración

La jerarquía alcanzada con Complete Linkage es:

$((A,B),(C,D))$

Y el dendograma quedaría así:



*Figura 3.19: Dendograma de
CompleteLinkage*

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 (sepallength, sepalwidth, petallength, petalwidth). ¿Alguno de ellos produce unos grupos bien diferenciados y con fronteras claras?

Los resultados obtenidos tras cargar la base de datos Iris y ejecutar el algoritmo HierarchicalClustered con la configuración dad en el enunciado son:

- Sepallength

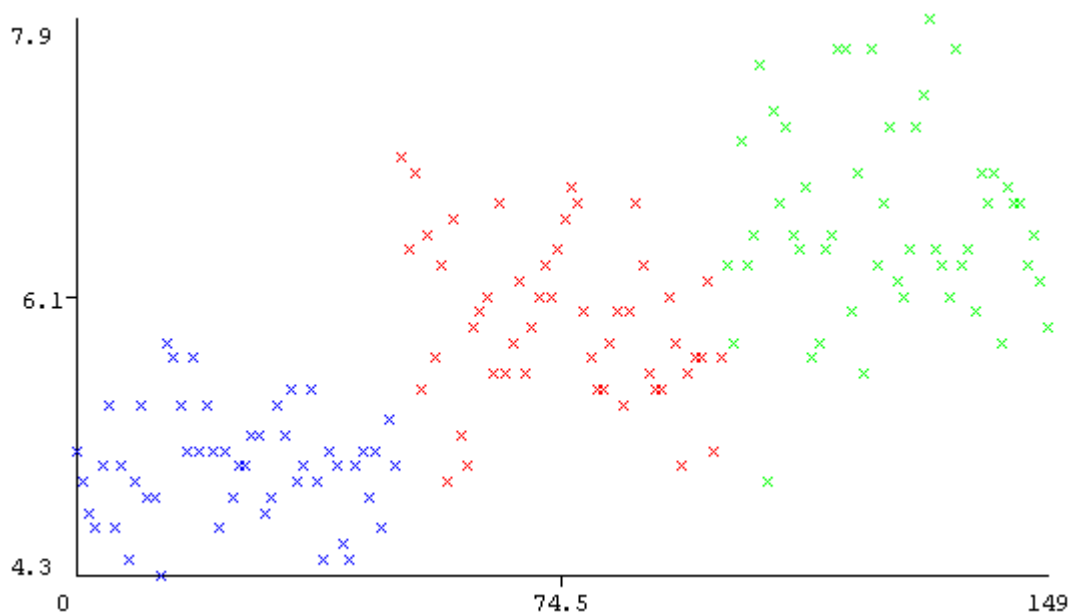
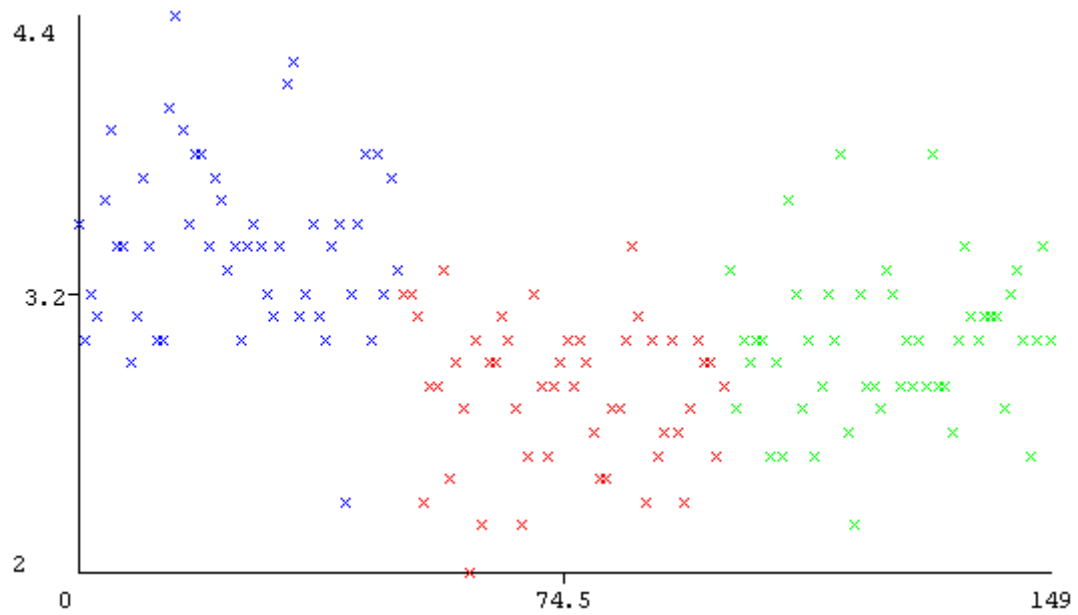


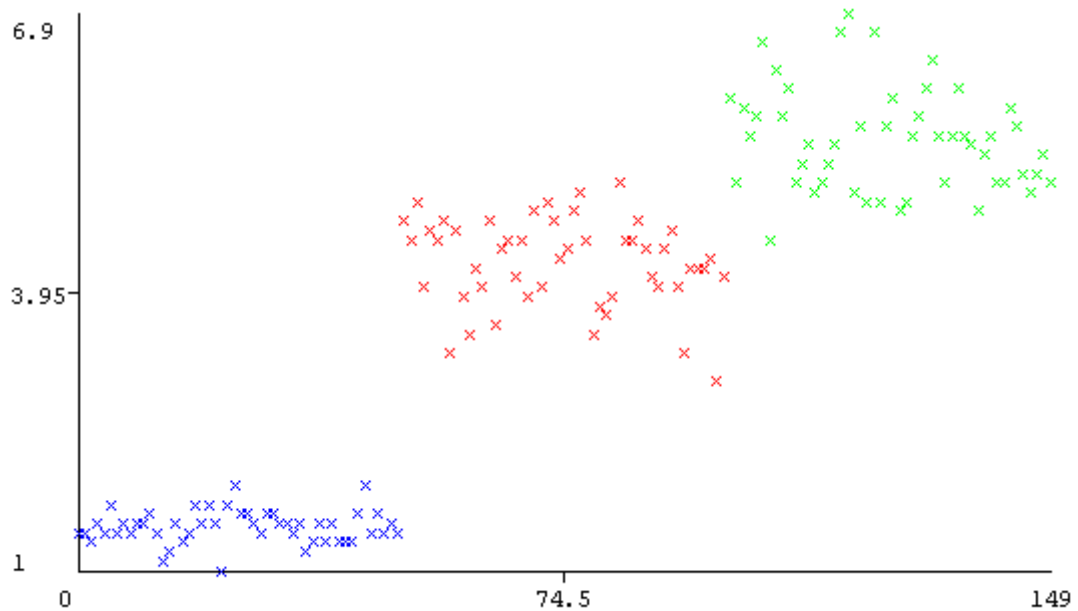
Figura 3.20: Grafica de puntos agrupados para el atributo Sepallength

- Sepalwidth



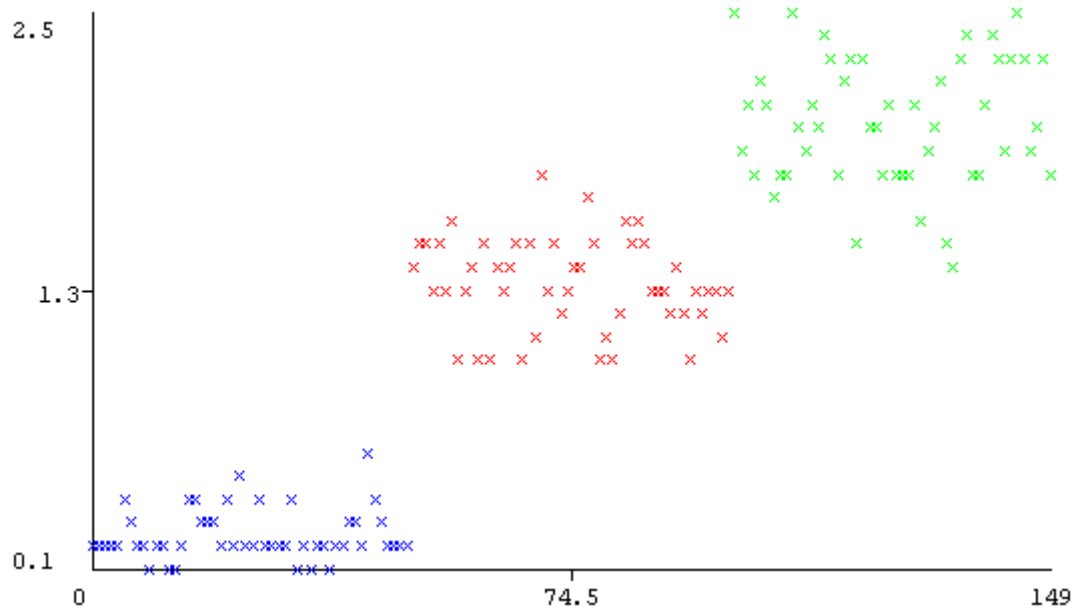
*Figura 3.21: Grafica de puntos agrupados
para el atributo Sepalwidth*

- Petallength



*Figura 3.22: Grafica de puntos agrupados
para el atributo Petallength*

- Petalwidth



*Figura 3.22: Grafica de puntos agrupados
para el atributo Petalwidth*

Como se puede observar, en el caso del atributo Petallength y Petalwidth si se aprecian unos grupos bien diferenciados con fronteras claras ya que no hay patrones de una clase dentro del grupo de otra.

3.9. Ejercicio 9: Ejecute el algoritmo DBScan sin emplear la etiqueta de clase.

3.9.1. Realice una batería de pruebas cambiando los valores de los parámetros ϵ 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 ϵ .

Voy a ejecutar el algoritmo primero con los parámetros por defecto y después voy a ir variando los valores de ϵ y minPoints para ir viendo cómo van variando las métricas que muestro en la siguiente tabla:

epsilon	minPoints	N.º Clusters	Inst. Agrupadas	Inst. Sin Agrupar	Inst. Mal Agrupadas
0.9	6	2	335	1	187
0.1	5	4	71	265	29
0.1	6	4	53	283	23

*Tabla 3.3: Resultados de las pruebas
variando ϵ y minPoints*

A pesar de haber probado diversas configuraciones, en la mayoría de casos nunca consigo mas de 2 clusters y cuando consigo mas de dos clusters los resultados son bastante malos como se puede observar en la tabla. Aun normalizando los valores de los atributos sigo sin conseguir buenos resultados.

En cuanto a la precisión de los clusters, podemos verla gracias a la matriz de confianza que nos muestra Weka:

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

Figura 3.23: Asignación de clases por clúster

- Cluster 0 = $35/(35+1) = 0.97$
- Cluster 1 = Sin clase
- Cluster 2 = 0.2
- Cluster 3 = 0.85

3.9.2. 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 esta prueba voy a tomar como mejor configuración la que obtengo con épsilon de 0.1 y minPoints de 5. Voy a probar a ejecutar esta configuración con las demás funciones de distancia lo cual se refleja en la siguiente tabla:

	N.º clusters	Inst. Agrupadas	Inst. Sin Agrupar	Inst. Mal Agrupadas
Euclídea	4	71	265	29
Chebyshev	3	236	100	62
Minkowski	4	71	265	29

*Tabla 3.4: Resultados de las pruebas
variando la función de distancia*

Como se observa en la tabla, con la distancia de Chebyshev parece que se obtiene un mejor resultado ya que se agrupan muchas mas instancias, pero las instancias que se agrupan mal son mayores también luego tampoco se podría decir que se obtiene un buen resultado con otra función de distancia diferente.

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.

4.4.1. 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.

Tras cargar la base de datos Ecoli, vamos a la pestaña de Classify y ejecutamos el algoritmo J48, que es como se llama el algoritmo C4.5 en Weka, con los parámetros por defecto. Para separar el 75% en train y el 25% en test hay que hacerlo tal y como se muestra en la imagen:

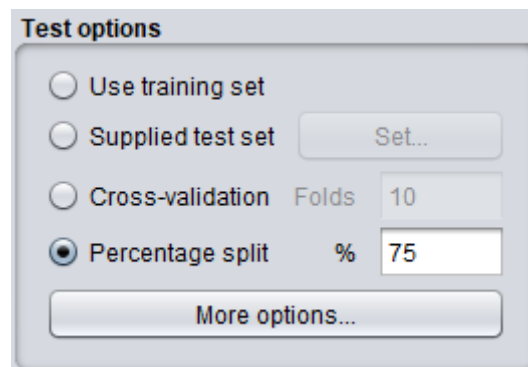


Figura 4.1: Configuración para 75% train y 25% test

Además, no sería necesario discretizar los datos ya que el algoritmo J48 acepta valores continuos. Una vez ejecutado el algoritmo con los parámetros por defecto, el árbol obtenido es el siguiente:

- El nodo principal o raíz es alm1 (marcado en rojo)
- El número de hojas es de 22 (ejemplo marcado en azul)
- El número de nodos es de 21 (ejemplo marcado en verde)
- La profundidad es de 9
- Variables presentes: cp, im, pp, imU, om, omL
- Variables omitidas: imL, imS

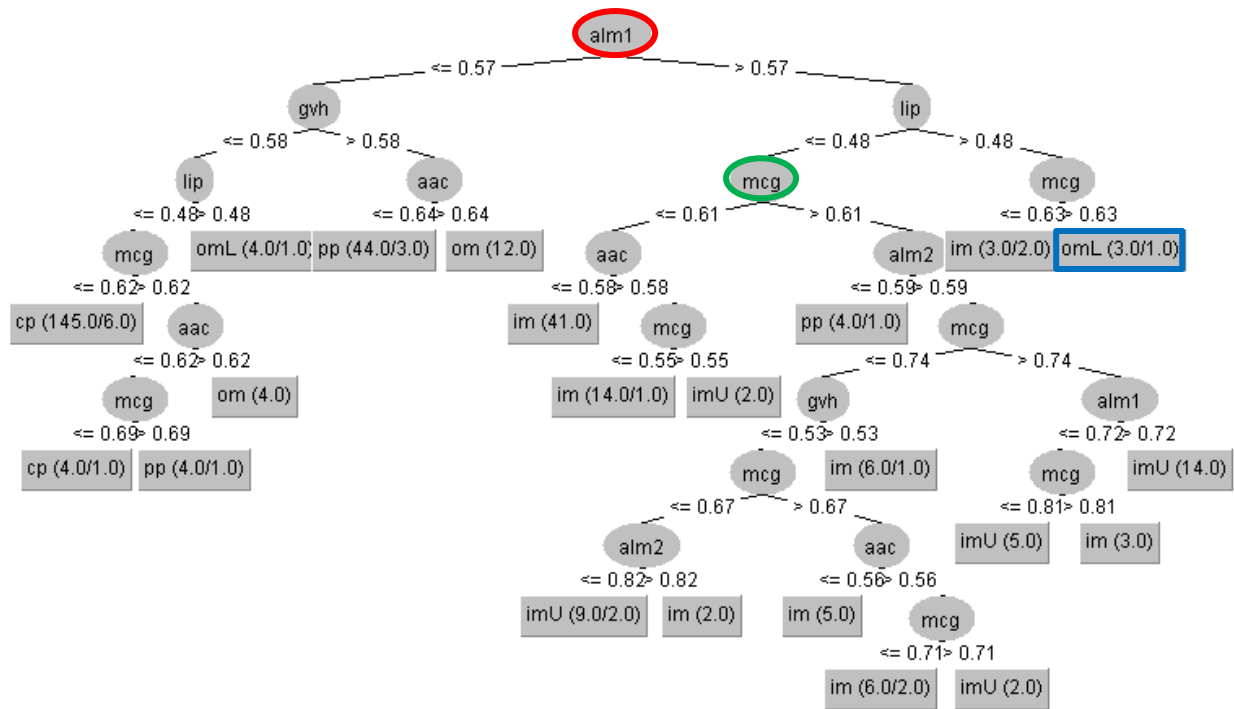


Figura 4.2: Arbol obtenido tras ejecutar J48
con *ecoli*

En cuanto a las métricas son las siguientes:

```

Correctly Classified Instances      69           82.1429 %
Incorrectly Classified Instances    15           17.8571 %
Kappa statistic                    0.7627
Mean absolute error                0.0512
Root mean squared error            0.2012
Relative absolute error            27.2789 %
Root relative squared error        64.3604 %
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    0,119    0,667     0,941    0,780     0,731    0,912     0,701     im
      0,778    0,061    0,778     0,778    0,778     0,717    0,880     0,786     pp
      0,556    0,013    0,833     0,556    0,667     0,651    0,747     0,551     imU
      0,571    0,000    1,000     0,571    0,727     0,742    0,818     0,633     om
      0,500    0,000    1,000     0,500    0,667     0,703    1,000     1,000     omL
      0,000    0,000    ?         0,000    ?         ?        0,494     0,012     imL
      ?        0,000    ?         ?         ?         ?        ?         ?         imS
Weighted Avg.  0,821    0,052    ?         0,821    ?         ?        0,894     0,773

```

*Figura 4.3: Métricas obtenidas tras ejecutar
J48*

Como se puede observar, las métricas están reflejadas para el 25% de las instancias que son usadas para el test, es decir, 84 instancias para test de 336 totales.

En cuanto al número de instancias bien clasificadas es de un 82%, 69 de 84 instancias, lo cual es un resultado bastante favorable. En cuanto al estadístico kappa obtenemos 0.7, valor cercano al 1 que es lo que se busca cuando se analiza este estadístico luego el resultado es también bastante bueno ya que quiere reflejar que hay una mayor concordancia que la que se esperaría por puro azar.

Los errores toman valores bajos que es lo que se busca. Exceptuando el Root Relative Squared Error el cual alcanza el 64%.

4.4.2. Analice y muestre cómo cambia el árbol (nodo principal, tamaño, número de hojas, variables y omitidas), al modificar los siguientes parámetros: confidenceFactor, minNumObj, unpruned, subtreeRaising.

- **Unpruned = True**
 - **confidenceFactor = 0,05**

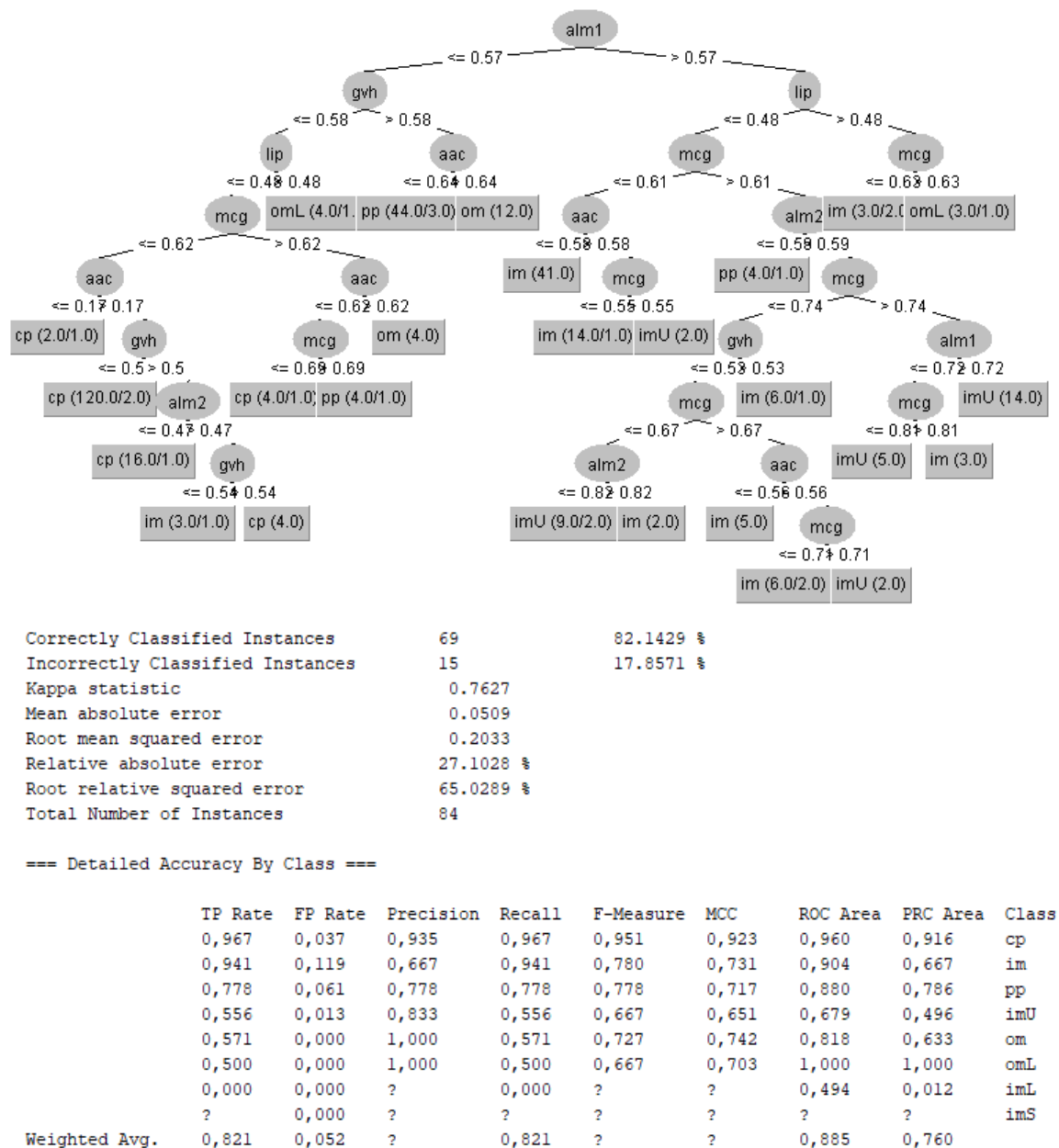
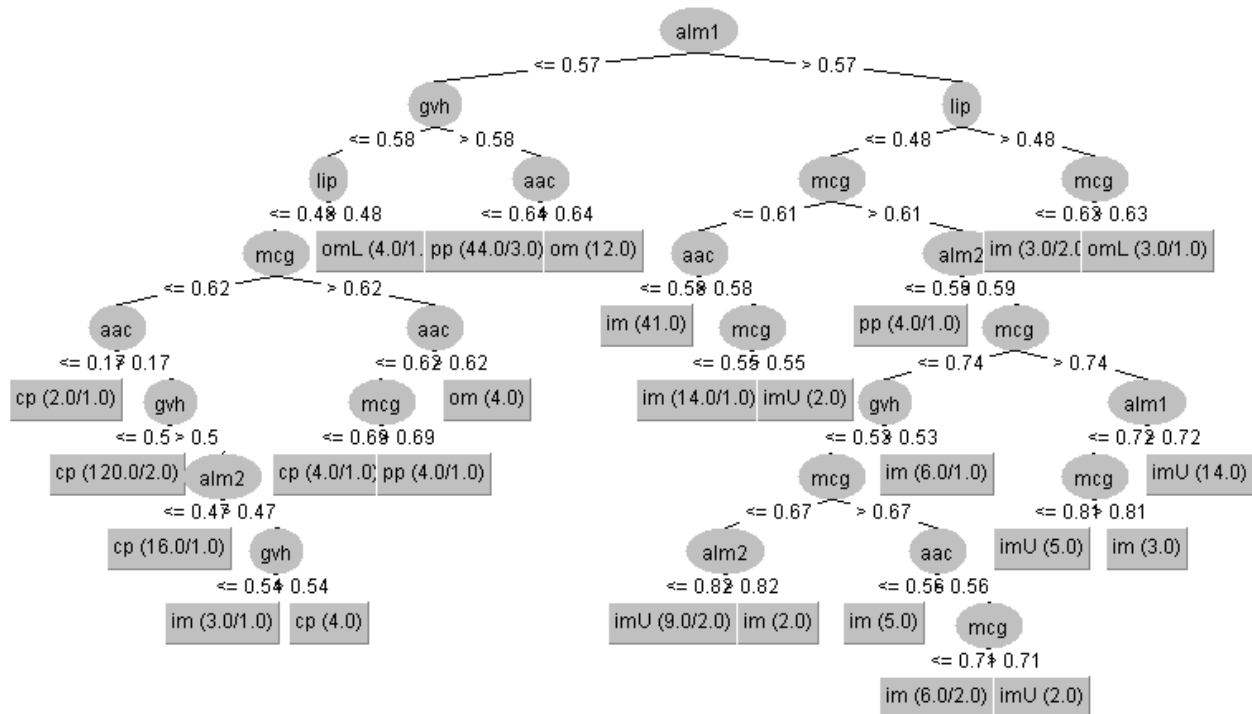


Figura 4.4: Árbol y métricas para unpruned=true y confidenceFactor=0.05

- **Unpruned = True**
 - **confidenceFactor = 0,25**



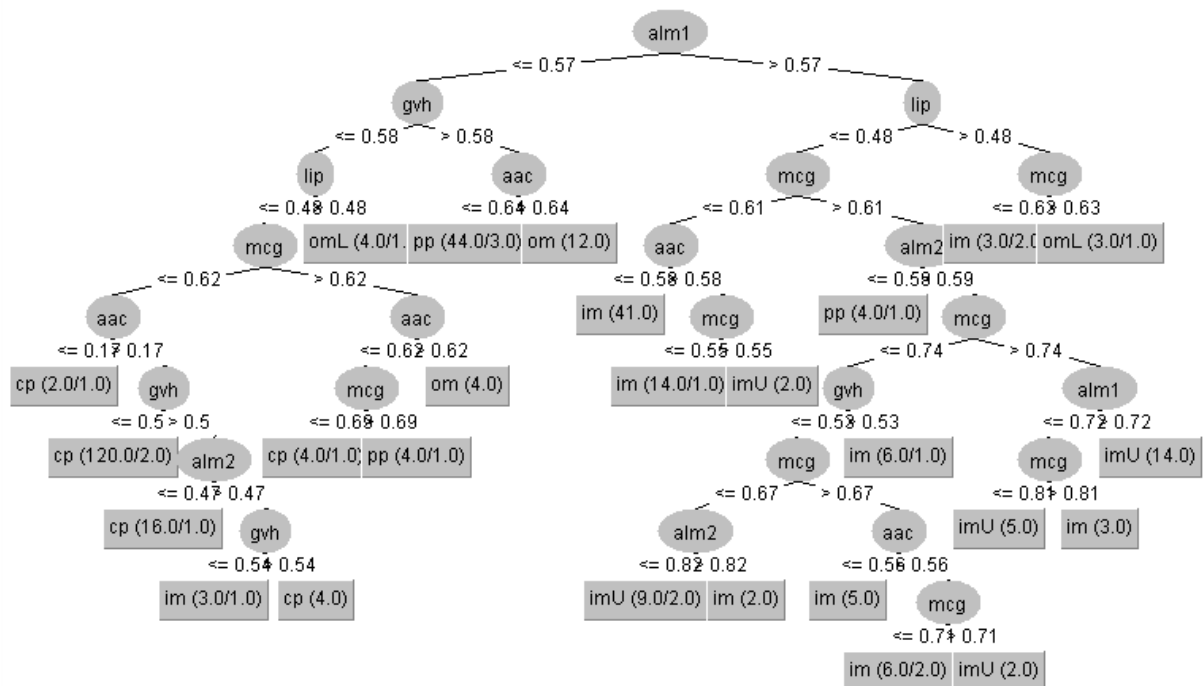
Correctly Classified Instances	69	82.1429 %
Incorrectly Classified Instances	15	17.8571 %
Kappa statistic	0.7627	
Mean absolute error	0.0509	
Root mean squared error	0.2033	
Relative absolute error	27.1028 %	
Root relative squared error	65.0289 %	
Total Number of Instances	84	

=== 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	0,119	0,667	0,941	0,780	0,731	0,904	0,667	im
	0,778	0,061	0,778	0,778	0,778	0,717	0,880	0,786	pp
	0,556	0,013	0,833	0,556	0,667	0,651	0,679	0,496	imU
	0,571	0,000	1,000	0,571	0,727	0,742	0,818	0,633	om
	0,500	0,000	1,000	0,500	0,667	0,703	1,000	1,000	omL
	0,000	0,000	?	0,000	?	?	0,494	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,821	0,052	?	0,821	?	?	0,885	0,760	

Figura 4.5: Árbol y métricas para
unpruned=true y confidenceFactor=0.25

- **Unpruned = True**
 - **confidenceFactor = 0,5**



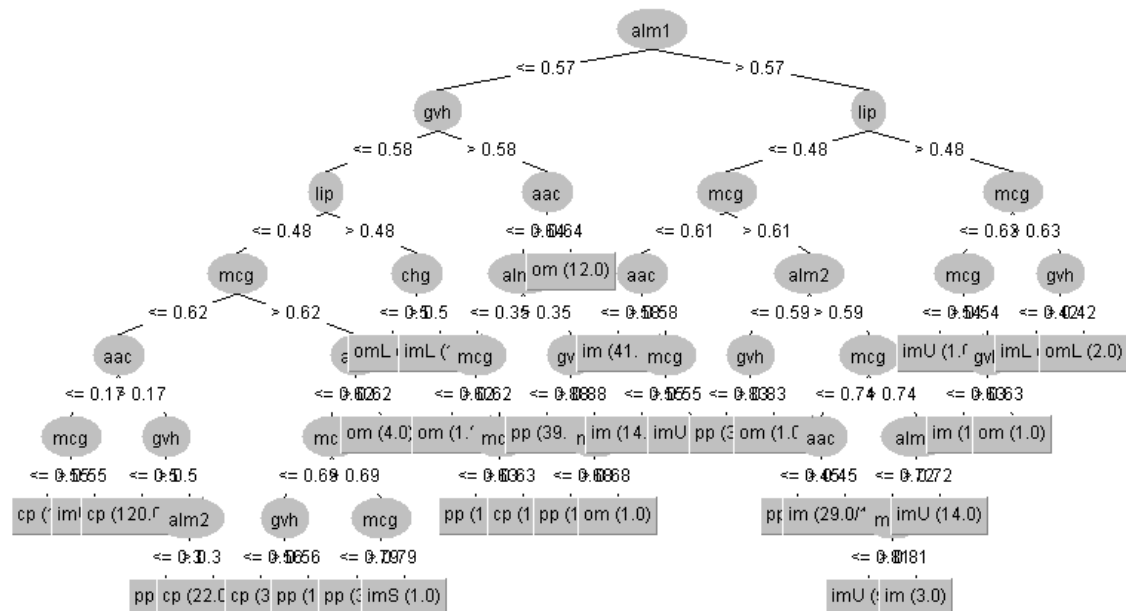
Correctly Classified Instances	69	82.1429 %
Incorrectly Classified Instances	15	17.8571 %
Kappa statistic	0.7627	
Mean absolute error	0.0509	
Root mean squared error	0.2033	
Relative absolute error	27.1028 %	
Root relative squared error	65.0289 %	
Total Number of Instances	84	

=== 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	0,119	0,667	0,941	0,780	0,731	0,904	0,667	im
	0,778	0,061	0,778	0,778	0,778	0,717	0,880	0,786	pp
	0,556	0,013	0,833	0,556	0,667	0,651	0,679	0,496	imU
	0,571	0,000	1,000	0,571	0,727	0,742	0,818	0,633	om
	0,500	0,000	1,000	0,500	0,667	0,703	1,000	1,000	omL
	0,000	0,000	?	0,000	?	?	0,494	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,821	0,052	?	0,821	?	?	0,885	0,760	

Figura 4.6: Árbol y métricas para
unpruned=true y confidenceFactor=0.5

- **Unpruned = True**
 - **minNumObj = 0**



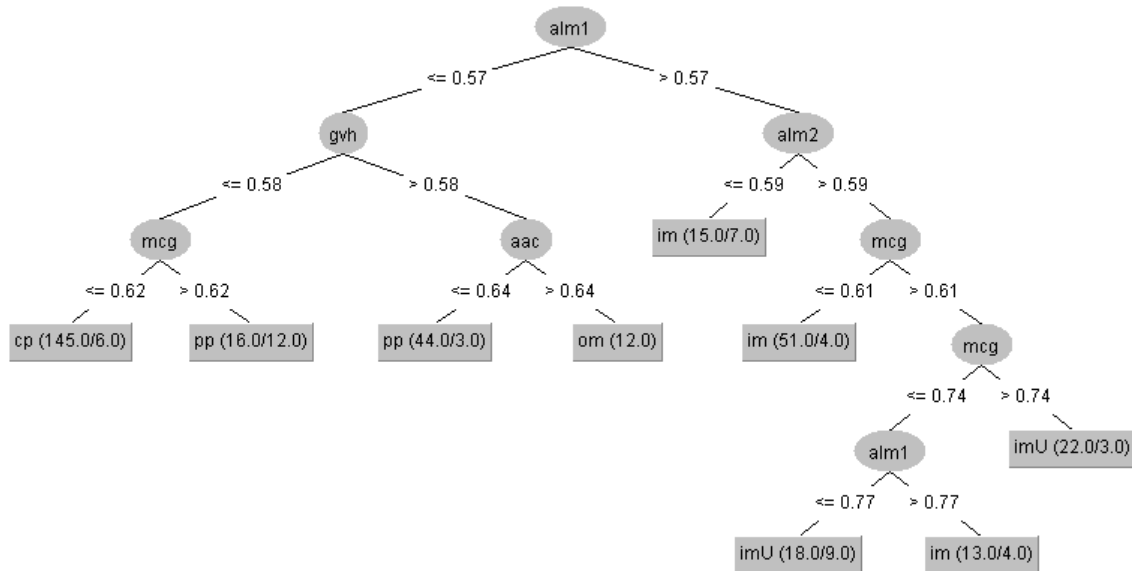
Correctly Classified Instances	67	79.7619 %
Incorrectly Classified Instances	17	20.2381 %
Kappa statistic	0.7352	
Mean absolute error	0.0523	
Root mean squared error	0.2239	
Relative absolute error	27.8411 %	
Root relative squared error	71.6366 %	
Total Number of Instances	84	

=== 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	omL
	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	

Figura 4.7: Árbol y métricas para
unpruned=true y minNumObj=0

- **Unpruned = True**
 - **minNumObj = 10**



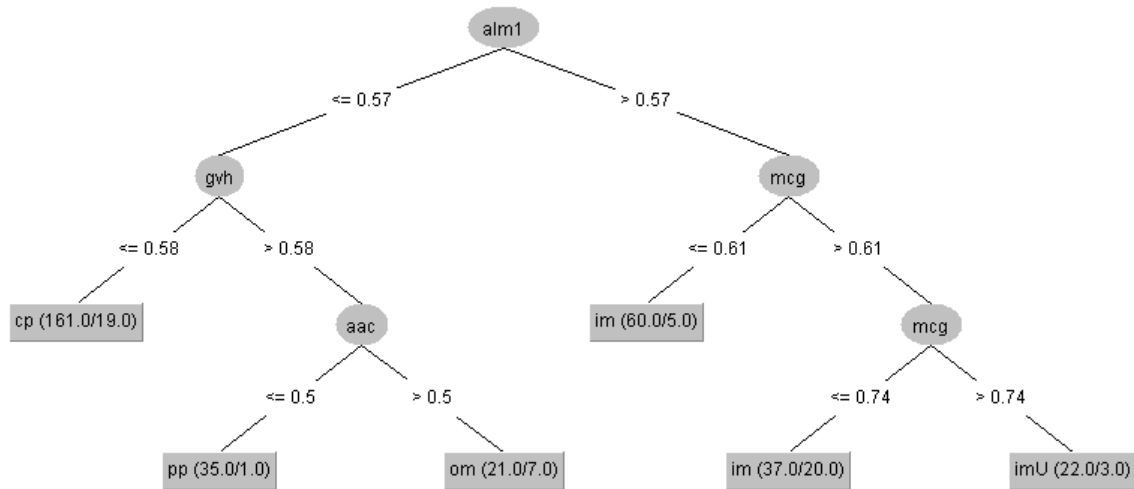
Correctly Classified Instances	67	79.7619 %
Incorrectly Classified Instances	17	20.2381 %
Kappa statistic	0.7315	
Mean absolute error	0.0594	
Root mean squared error	0.1958	
Relative absolute error	31.6165 %	
Root relative squared error	62.6287 %	
Total Number of Instances	84	

=== 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	

*Figura 4.8: Árbol y métricas para
unpruned=true y minNumObj=10*

- **Unpruned = True**
 - **minNumObj = 20**



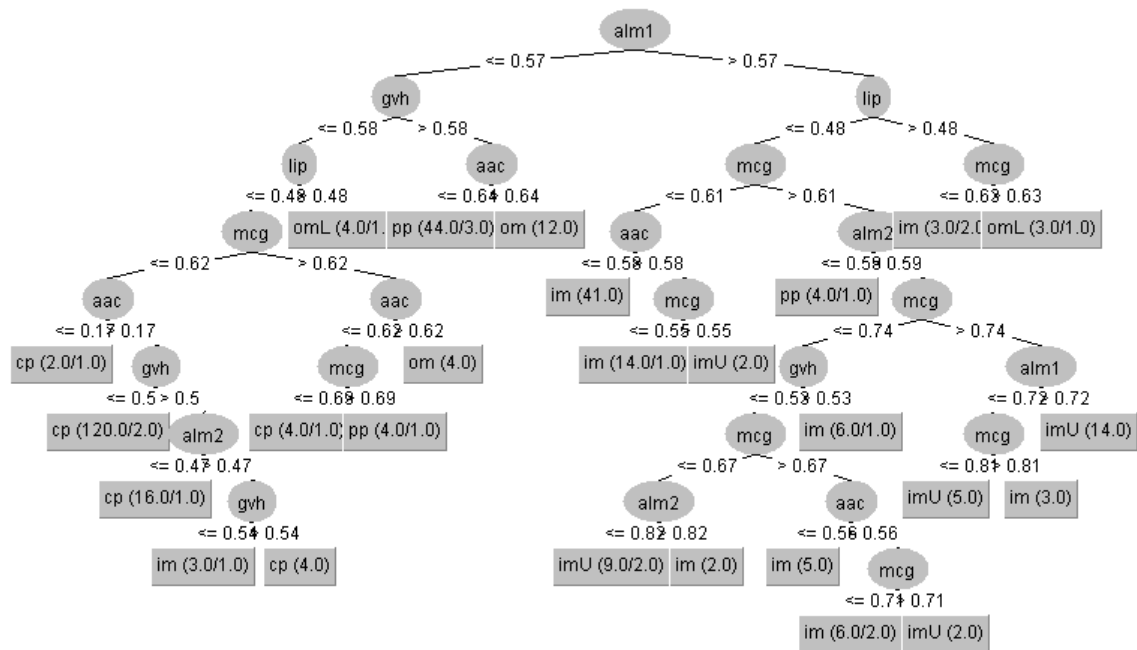
Correctly Classified Instances	64	76.1905 %
Incorrectly Classified Instances	20	23.8095 %
Kappa statistic	0.6765	
Mean absolute error	0.0845	
Root mean squared error	0.2131	
Relative absolute error	44.9818 %	
Root relative squared error	68.175 %	
Total Number of Instances	84	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,093	0,857	1,000	0,923	0,882	0,954	0,857	cp
	0,941	0,134	0,640	0,941	0,762	0,709	0,960	0,789	im
	0,778	0,076	0,737	0,778	0,757	0,689	0,886	0,652	pp
	0,444	0,013	0,800	0,444	0,571	0,564	0,930	0,568	imU
	0,000	0,000	?	0,000	?	?	0,729	0,213	om
	0,000	0,000	?	0,000	?	?	0,805	0,067	omL
	0,000	0,000	?	0,000	?	?	0,289	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,762	0,078	?	0,762	?	?	0,908	0,686	

Figura 4.9: Árbol y métricas para unpruned=true y minNumObj=20

- **Unpruned = True**
 - **subtreeRaising = True**



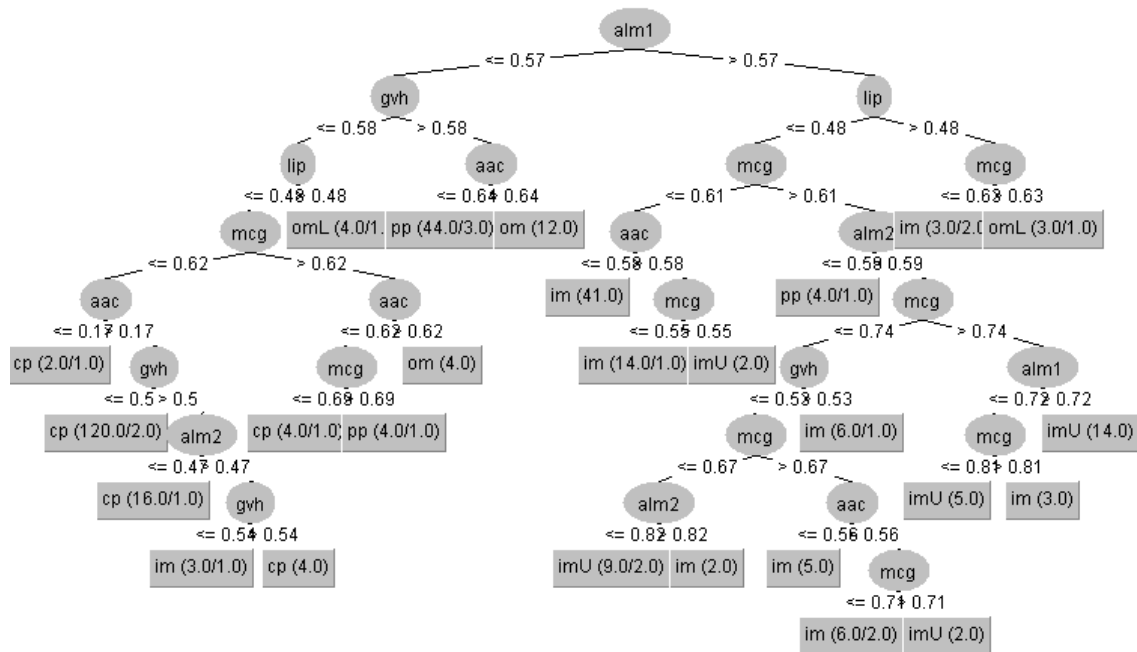
Correctly Classified Instances	69	82.1429 %
Incorrectly Classified Instances	15	17.8571 %
Kappa statistic	0.7627	
Mean absolute error	0.0509	
Root mean squared error	0.2033	
Relative absolute error	27.1028 %	
Root relative squared error	65.0289 %	
Total Number of Instances	84	

=== 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	0,119	0,667	0,941	0,780	0,731	0,904	0,667	im
	0,778	0,061	0,778	0,778	0,778	0,717	0,880	0,786	pp
	0,556	0,013	0,833	0,556	0,667	0,651	0,679	0,496	imU
	0,571	0,000	1,000	0,571	0,727	0,742	0,818	0,633	om
	0,500	0,000	1,000	0,500	0,667	0,703	1,000	1,000	omL
	0,000	0,000	?	0,000	?	?	0,494	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,821	0,052	?	0,821	?	?	0,885	0,760	

*Figura 4.10: Árbol y métricas para
unpruned=true y subtreeRaising = true*

- **Unpruned = True**
 - **subtreeRaising = False**



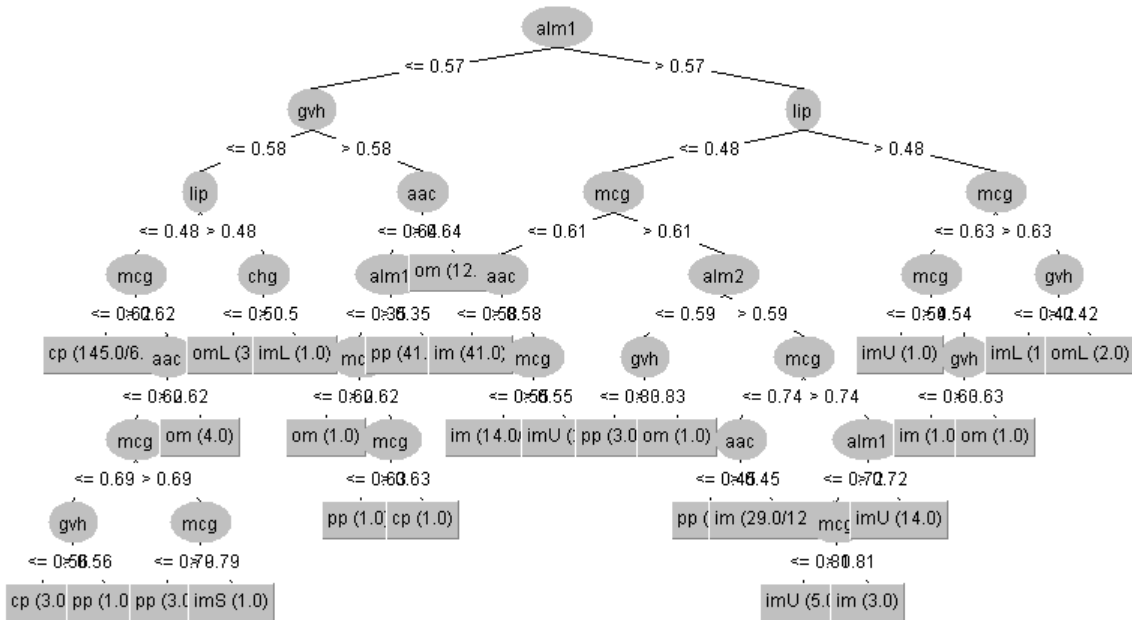
Correctly Classified Instances	69	82.1429 %
Incorrectly Classified Instances	15	17.8571 %
Kappa statistic	0.7627	
Mean absolute error	0.0509	
Root mean squared error	0.2033	
Relative absolute error	27.1028 %	
Root relative squared error	65.0289 %	
Total Number of Instances	84	

```
=== 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	0,119	0,667	0,941	0,780	0,731	0,904	0,667	im
	0,778	0,061	0,778	0,778	0,778	0,717	0,880	0,786	pp
	0,556	0,013	0,833	0,556	0,667	0,651	0,679	0,496	imU
	0,571	0,000	1,000	0,571	0,727	0,742	0,818	0,633	om
	0,500	0,000	1,000	0,500	0,667	0,703	1,000	1,000	omL
	0,000	0,000	?	0,000	?	?	0,494	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,821	0,052	?	0,821	?	?	0,885	0,760	

Figura 4.11: Árbol y métricas para `unpruned=true` y `subtreeRaising = false`

- **minNumObj = 0**



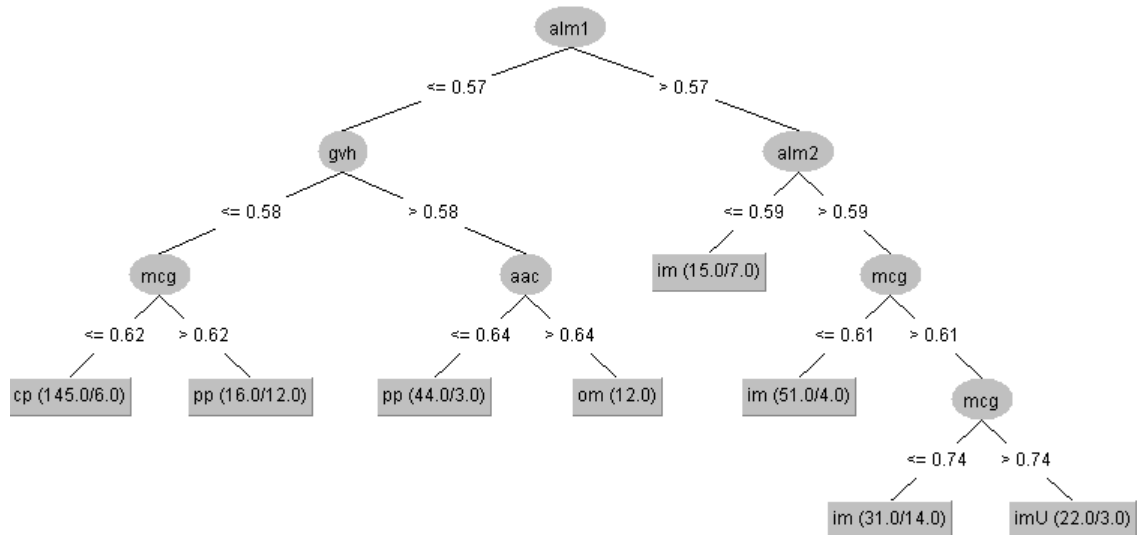
Correctly Classified Instances	71	84.5238 %
Incorrectly Classified Instances	13	15.4762 %
Kappa statistic	0.7968	
Mean absolute error	0.0469	
Root mean squared error	0.1921	
Relative absolute error	24.9594 %	
Root relative squared error	61.4698 %	
Total Number of Instances	84	

```
=== 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	0,090	0,727	0,941	0,821	0,778	0,954	0,738	im
	0,778	0,015	0,933	0,778	0,848	0,817	0,887	0,788	pp
	0,667	0,013	0,857	0,667	0,750	0,731	0,826	0,687	imU
	0,714	0,013	0,833	0,714	0,769	0,753	0,895	0,648	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.	0,845	0,037	0,866	0,845	0,848	0,820	0,913	0,785	

Figura 4.12: Árbol y métricas para $unpruned=false$ y $minNumObj = 0$

- **Unpruned = False**
 - **minNumObj = 10**



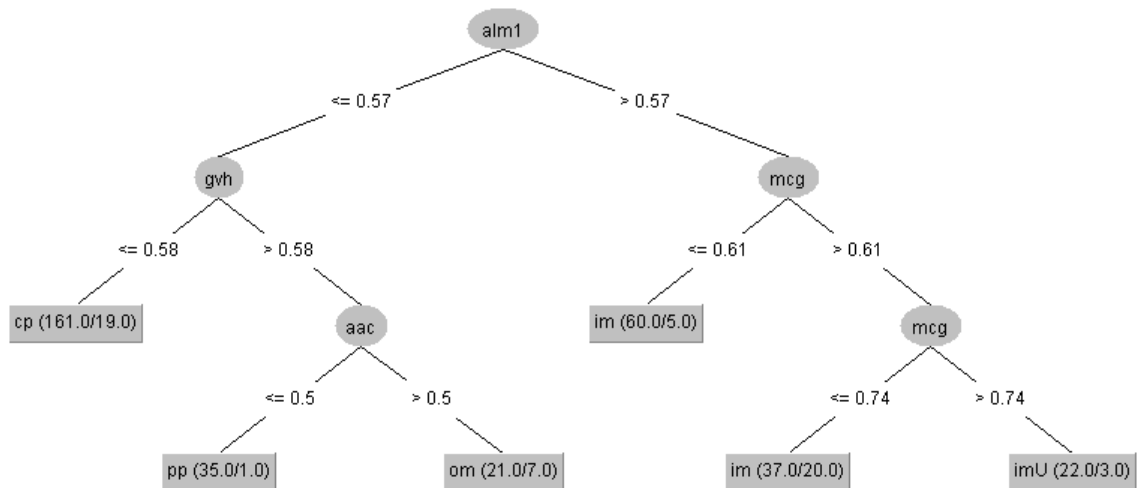
Correctly Classified Instances	67	79.7619 %
Incorrectly Classified Instances	17	20.2381 %
Kappa statistic	0.7299	
Mean absolute error	0.0656	
Root mean squared error	0.1958	
Relative absolute error	34.9339 %	
Root relative squared error	62.6288 %	
Total Number of Instances	84	

=== 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	

Figura 4.13: Árbol y métricas para
unpruned=false y minNumObj = 10

- **Unpruned = False**
 - **minNumObj = 20**



```

Correctly Classified Instances      64          76.1905 %
Incorrectly Classified Instances    20          23.8095 %
Kappa statistic                    0.6765
Mean absolute error                 0.0845
Root mean squared error             0.2131
Relative absolute error             44.9818 %
Root relative squared error         68.175 %
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	0,093	0,857	1,000	0,923	0,882	0,954	0,857	cp
	0,941	0,134	0,640	0,941	0,762	0,709	0,960	0,789	im
	0,778	0,076	0,737	0,778	0,757	0,689	0,886	0,652	pp
	0,444	0,013	0,800	0,444	0,571	0,564	0,930	0,568	imU
	0,000	0,000	?	0,000	?	?	0,729	0,213	om
	0,000	0,000	?	0,000	?	?	0,805	0,067	omL
	0,000	0,000	?	0,000	?	?	0,289	0,012	imL
	?	0,000	?	?	?	?	?	?	imS
Weighted Avg.	0,762	0,078	?	0,762	?	?	0,908	0,686	

*Figura 4.14: Árbol y métricas para
unpruned=false y minNumObj=20*

Voy a mostrar a continuación una tabla en la que recojo los cambios de los arboles dependiendo de los parámetros ajustados:

			Nodo Principal	Hojas	Tamaño
Unpruned=True	confidenceFactor	0.05	alm1	26	51
		0.25	alm1	26	51
		0.5	alm1	26	51
	minNumObj	0	alm1	34	67
		10	alm1	9	17
		20	alm1	6	11
	subtreeRaising	False	alm1	26	51
		True	alm1	26	51
Unpruned=False	minNumObj	0	alm1	28	55
		10	alm1	6	11
		20	alm1	6	11

Table 4.1: Tabla de comparación de configuraciones y resultados

En cuanto a que configuración arroja los mejores resultados, podría decir que la mejor configuración es la siguiente (marcado en rojo en la tabla):

- unpruned = False
 - minNumObj = 0
 - Demás parámetros por defecto.

Ya que obtengo las mejores métricas con un CCR de 84,52% de instancias bien clasificadas.

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.

Tras ejecutar el algoritmo Weka nos muestra una serie de información en la que se encuentran modelos logísticos obtenidos para cada clase, el árbol y las métricas.

Primero vamos a analizar los modelos logísticos los cuales muestro a continuación por cada hoja generada:

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	LM_3: Class cp : 2.63 + [mcg] * -4.21 + [alml] * -6.39 Class im : 0.05 + [mcg] * -6.35 + [gvh] * 1.59 + [alml] * 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 + [alml] * 9.9 Class imS : -6.25 + [mcg] * 1.41 + [aac] * -3.82 +	LM_4: Class cp : 2.69 + [mcg] * -4.19 + [alml] * -6.49 Class im : -11.4 + [mcg] * 1.01 + [gvh] * 1.59 + [alml] * 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 + [alml] * 9.95 Class imS : -7.99 + [mcg] * 1.41 + [alml] * -0.04 + [alm2] * 4.72
--	---	--	---

El modelo logístico de cada hoja nos representa los atributos que más peso tienen en cada clase según el modelo. En el caso de la primera hoja, “LM_1”, podemos ver que atributos como “chg” tienen un gran peso en la clase “imL”, o como el atributo “mcg” tiene gran peso en la clase “imS” mientras que otros atributos ni se pintan, señal de que no tienen nada de peso a la hora de clasificar esa clase. El primer valor que se muestra tras cada clase es el sesgo que se aplica.

Por otro lado, Weka nos muestra el siguiente árbol para este modelo:

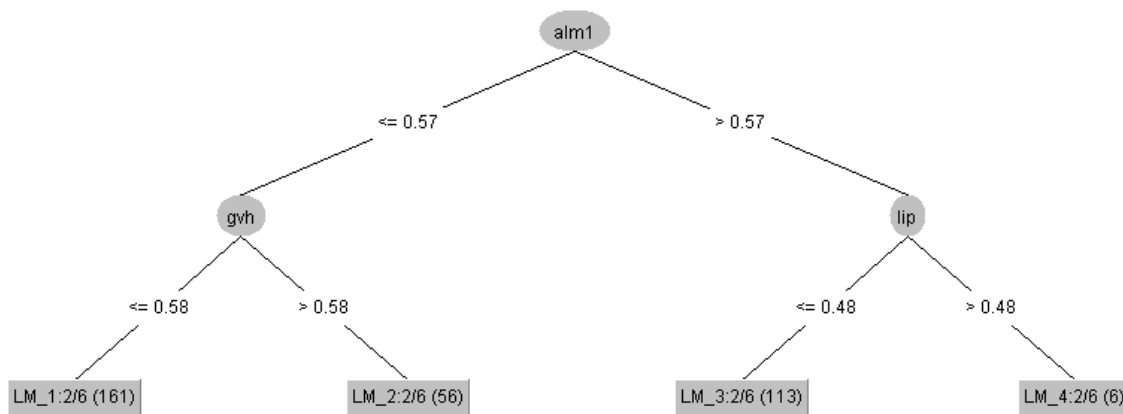


Figura 4.15: Árbol para el algoritmo LMT en ecoli

Como podemos ver es un árbol mucho más simple que los analizados anteriormente, aunque es menos interpretable. Vemos las cuatro hojas que hemos obtenido y el número de instancias a las que pertenece cada hoja, 161 para la hoja LM_1, 56 para la hoja LM_2, 113 para la hoja LM_3 y 6 para la hoja LM_4. El nodo raíz es el atributo alm1 y las reglas del modelo son las siguientes:

IF alm1 <=0.57 **AND** gvh <=0.58 **THEN** instancia pertenece a **LM_1**

IF alm1 <=0.57 **AND** gvh >0.58 **THEN** instancia pertenece a **LM_2**

IF alm1 >0.57 **AND** lip <=0.48 **THEN** instancia pertenece a **LM_3**

IF alm1 >0.57 **AND** lip >0.48 **THEN** instancia pertenece a **LM_4**

Por último, las métricas obtenidas son las siguientes:

```

Correctly Classified Instances      70              83.3333 %
Incorrectly Classified Instances   14              16.6667 %
Kappa statistic                    0.7794
Mean absolute error                 0.0549
Root mean squared error            0.1678
Relative absolute error            29.2666 %
Root relative squared error        53.6988 %
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    0,988     0,968     cp
      0,882    0,104    0,682     0,882    0,769     0,711    0,973     0,923     im
      0,778    0,030    0,875     0,778    0,824     0,781    0,940     0,898     pp
      0,444    0,040    0,571     0,444    0,500     0,453    0,954     0,624     imU
      0,714    0,000    1,000     0,714    0,833     0,834    0,998     0,982     om
      1,000    0,000    1,000     1,000    1,000     1,000    1,000     1,000     omL
      0,000    0,000    ?         0,000    ?         ?        0,663     0,034     imL
      ?       0,000    ?         ?         ?         ?        ?         ?         imS
Weighted Avg.    0,833    0,045    ?         0,833    ?         ?        0,968     0,898

=== 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 14  1  0  0  0  0  | c = pp
 0  5  0  4  0  0  0  0  | d = imU
 0  0  2  0  5  0  0  0  | e = om
 0  0  0  0  0  2  0  0  | f = omL
 0  1  0  0  0  0  0  0  | g = imL
 0  0  0  0  0  0  0  0  | h = imS

```

*Figura 4.16: Métricas para el algoritmo LMT
en ecoli*

4.3. Ejercicio 3: Sobre el algoritmo RandomForest realice una labor de investigación:

4.3.1. Explique con sus palabras en que se basa este árbol de decisión.

RandomForest es un algoritmo el cual basa su método en combinar una gran cantidad de árboles de decisión independientes fabricados a partir de entradas ligeramente cambiadas.

Para generar esos datos de entrada iniciales se selecciona aleatoriamente un porcentaje de datos del total y se genera un árbol de decisión. Tras haber evaluado cada árbol de forma independiente, la predicción del bosque será la media del resultado de todos los árboles.

4.3.2. 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
```

Figura 4.17: Métricas para el algoritmo
RandomForest en ecoli

4.4. Ejercicio 4: Utilizando la base de datos “Iris” (Moodle) con un 75/25 %, ejecute el algoritmo MultilayerPerceptron con los parámetros por defecto.

4.4.1. Represente gráficamente la red resultante y añada en la gráfica (con el editor que estime oportuno) el valor de todos los pesos de la red neuronal.

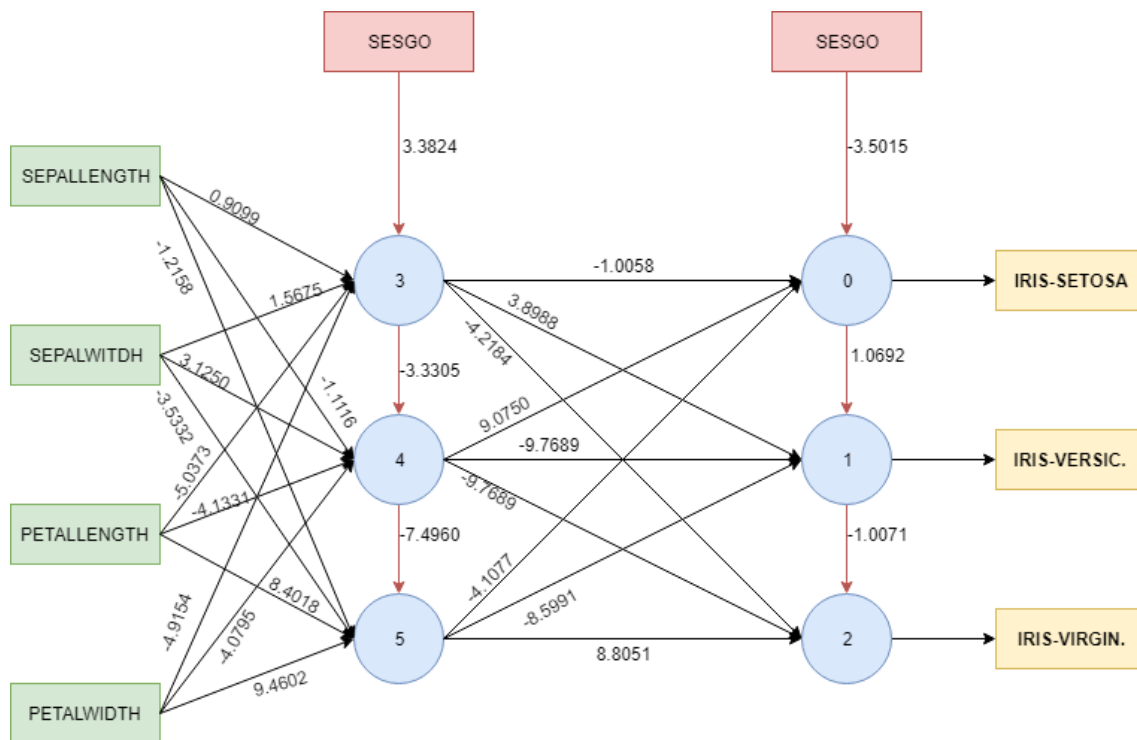


Figura 4.18: Red neuronal resultante de ejecutar MultilayerPerceptron

4.5. Ejercicio 5: Utilizando su base de datos con un 75/25% y el algoritmo MultilayerPerceptron.

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

El atributo hiddenLayers define el número de capas ocultas que tendrá la red neuronal. Por defecto Weka pone de valor el parámetro “a” que equivale a el número de atributos más el número de clases, todo ello entre dos. Es decir:

$$a = (\text{attribs} + \text{classes}) / 2$$

$$a = (8 + 8) / 2$$

$$a = 8$$

Luego el número de capas ocultas de nuestra red será de **1 capa con 8 nodos**.

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

El learningRate hace referencia al número de enlaces que son modificados en cada iteración. La tasa de aprendizaje es un parámetro que va ajustando los pesos en cada iteración de la red neuronal mediante la siguiente formula:

$$\text{nuevoPeso} = \text{pesoExistente} - \text{learningRate} * \text{gradiente}$$

Es decir, cuanto más se acerque a 1, menor van a ser las modificaciones. A continuación, mostrare una tabla en la que comparo una serie de métricas para cada valor del parámetro learningRate:

LeraningRate	CCR	Kappa	MAE	RMSE	RMSE	RRSE
0.1	85.71%	0.8116	0.0575	0.1704	30.60%	54.51%
0.3	84.52%	0.7956	0.0516	0.1777	27.50%	56.86%
0.6	80.95%	0.7473	0.0521	0.1879	27.73%	60.13%
1	82.14%	0.7653	0.0497	0.1855	26.46%	59.34%

Table 4.2: Comparación de valores de learningRate y resultados

Como se puede observar en la tabla, el mejor resultado (en rojo) se obtiene con una tasa de aprendizaje de 0.1, es decir, con un valor mínimo el cual va a influir en gran medida a él cálculo de los pesos de la neurona. Además, obtengo el mejor valor de Kappa y unos buenos errores.

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

El parámetro momentum influye en la modificación de los enlaces. Nos ayuda a hacer que nuestro modelo no quede estancado en óptimos locales. Cuanto más cercano a 1, las modificaciones serán más fuertes. Al igual que con el apartado anterior, voy a realizar una serie de pruebas y voy a mostrarlas en una tabla para su mejor análisis:

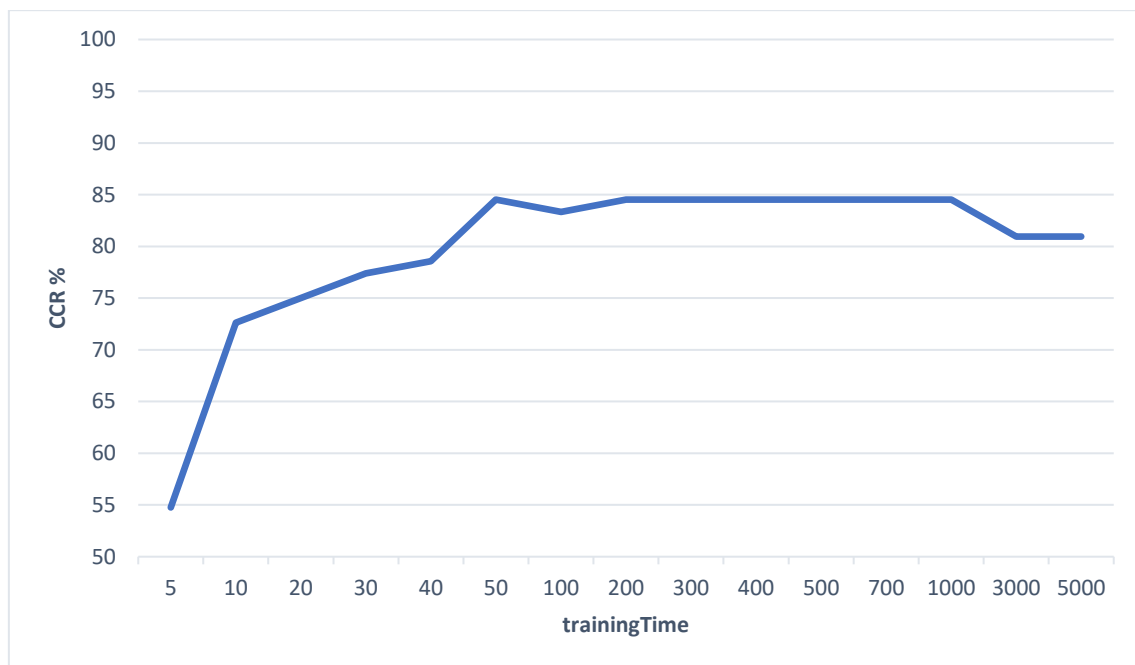
Momentum	CCR	Kappa	MAE	RMSE	RMSE	RRSE
0.1	84.52%	0.7955	0.0534	0.179	28.45%	57.26%
0.3	79.76%	0.7308	0.054	0.1878	28.73%	60.08%
0.6	80.95%	0.7473	0.0517	0.1869	27.55%	59.79%
0.9	82.14%	0.7648	0.0533	0.1903	28.38%	60.89%

Table 4.3: Comparación de valores de momentum y resultados

Los mejores resultados se han vuelto a alcanzar con un momentum mínimo de 0.1 ya que se obtiene el mayor número de instancias bien clasificadas, el estadístico Kappa más alto y los errores mínimos.

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

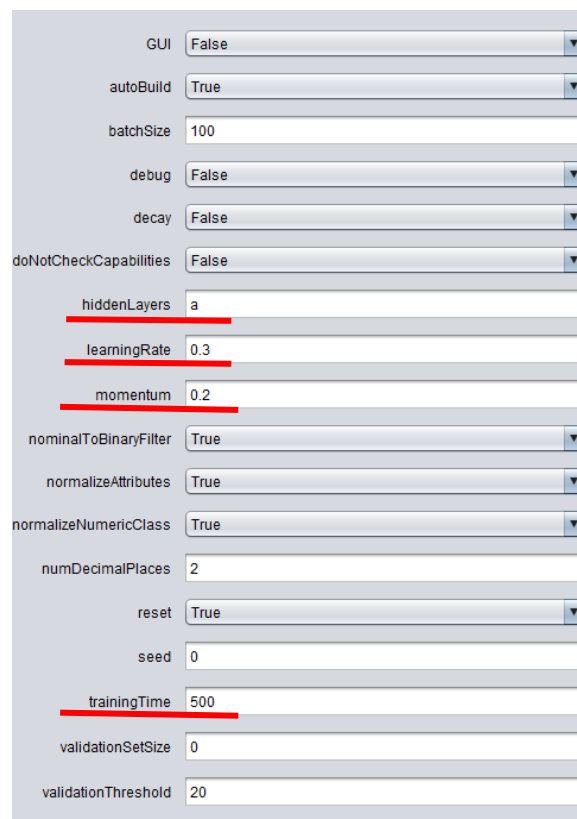
Como se refleja en la gráfica, conforme va aumentando el tiempo de entrenamiento, el CCR aumenta hasta alcanzar un tope de 85% para un trainingTime de 50. A partir de ese valor, el CCR se va manteniendo hasta alcanzar un trainingTime de 1000, cifra a partir de la cual el CCR comienza a disminuir.



*Figura 4.19: Gráfica de relación
CCR/trainingTime*

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

Voy a proceder a lanzar varias pruebas variando los parámetros del Multilayer Perceptron que muestro a continuación marcados en rojo:



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

Figura 4.20: Parametros a variar para ejecutar Multilayer Perceptron

A continuación, mostraré una tabla con las configuraciones y los resultados obtenidos para ver cual es la mejor configuración:

Capas	Nodos x Capa	learningRate	momentum	trainingTime	CCR	Kappa	MAE	RMSE
1	8	0.3	0.2	500	84.52%	0.795	0.051	0.177
2	8	0.3	0.2	500	84.52%	0.796	0.055	0.188
3	8	0.3	0.2	500	85.71%	0.812	0.053	0.181
3	8	0.5	0.2	500	78.57%	0.713	0.062	0.208
3	8	0.3	0.1	500	80.95%	0.749	0.060	0.188
3	8	0.3	0.5	500	79.76%	0.730	0.730	0.730
3	8	0.3	0.2	1000	84.52%	0.795	0.052	0.185
1	5	0.3	0.2	1000	85.71%	0.812	0.054	0.187
2	5	0.3	0.2	1000	79.76%	0.732	0.062	0.202

*Table 4.4: Configuración y resultados para
Multilayer Perceptron*

Como se puede observar en la tabla, el mejor resultado se obtiene con la siguiente configuración:

- Numero de capas ocultas = 3
- Nodos por capa oculta = 8
- Tasa de aprendizaje (learningRate) = 0.3
- Momentum = 0.2
- Tiempo de entrenamiento (trainingTime) = 500

En cuanto a la tasa de aprendizaje, básicamente lo que esta nos indica es que medida va a aprender nuestro algoritmo. Un valor alto de la tasa nos va a indicar que nuestro algoritmo va a aprender mas rápidamente, con los problemas que esto conlleva. Mientras que un valor muy bajo daría lugar a que nuestro algoritmo no aprenda lo suficiente como para mostrar un buen resultado. Por otro lado, el momentum nos va a ayudar a que el algoritmo no

se quede estancado en óptimos locales y que salga de ellos. Un valor alto hará que las modificaciones en cada iteración sean más fuertes.

Los valores para ambos parámetros han sido bajos ya que usando valores más altos como 0.8 o 0.9 se han obtenido resultados muy malos.

Como curiosidad, al ejecutar Multilayer Perceptron con un cross-validation de 10 en vez de dividir el dataset en 75/25, consigo alcanzar un 86.01% de CCR lo cual supera las pruebas anteriores. La configuración de parámetros usada para ello han sido los por defecto.

5. Práctica 4: Competición con la plataforma Kaggle

El resultado de la subida a Kaggle se ha conseguido mediante la aplicación de un perceptrón multicapa con los parámetros por defecto tras haber realizado un preprocesamiento previo a el dataset.

Este preprocesamiento ha consistido en una primera eliminación del atributo MWD ya que contenía un 100% de missing values. Luego he aplicado el filtro: `filters/unsupervised/ReplaceMissingValues` para eliminar los missing values de los demás atributos ya que contenían bastantes. Por último, normalice los valores de los atributos.

Con esto conseguí un F-measure de 0.56 que fue el mejor resultado que obtuve.

Apliqué varios filtros más como el Random Forest, pero no conseguí mejorar esa marca.

6. Referencias bibliográficas

[1] P. A. Gutiérrez, J.C. Fernández, D. Guijo, “Material docente para la asignatura de Introducción al Aprendizaje Automático” [online], Dpto. de Informática y análisis Numérico, Universidad de Córdoba, 2018.

Disponible en: <http://moodle.uco.es/m1718/course/view.php?id=991>

[2] Documentación de WEKA [online], The University of Waikato, 2018.

Disponible en: <https://www.cs.waikato.ac.nz/ml/weka/>

[3] Información sobre el dataset Iris [online], UCI.

Disponible en: <https://archive.ics.uci.edu/ml/datasets/iris>

[4] Información sobre el dataset Ecoli [online], UCI.

Disponible en: <https://archive.ics.uci.edu/ml/datasets/ecoli>

[5] Información sobre Cross-validation [online], Wikipedia, 2018.

Disponible en: [https://en.wikipedia.org/wiki/Cross-validation_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))

[6] Diferencia entre distintos tipos de errores [online].

Disponible en: <https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d>

[7] Información sobre K-means [online], Wikipedia, 2018.

Disponible en: https://en.wikipedia.org/wiki/K-means_clustering

[8] Información sobre Random Forest [online].

Disponible en: <https://quantdare.com/random-forest-vs-simple-tree/>