

UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba



INTRODUCCIÓN AL APRENDIZAJE AUTOMÁTICO

BASE DE DATOS: DATASET DE CRIATURAS E IRIS

Índice general

1. Práctica 1 - Introducción a Weka	8
1.1. Cargue la base de datos Iris. Observe los atributos.	8
1.1.1. ¿Cuántos atributos caracterizan los datos de esta base de datos?	8
1.1.2. ¿Se trata de regresión o clasificación?	8
1.1.3. ¿Cuál es el rango de valores del atributo petalwidth? Y su media? Y su desviación típica?	9
1.1.4. Determinar que atributo permite discriminar linealmente entre la clase Iris Setosa y las otras dos clases.	9
1.1.5. ¿Es posible separar linealmente la clase iris-versicolor de la clase iris-virginica?	10
1.1.6. ¿Con qué dos atributos te quedarías para discriminar entre las tres clases del problema?	10
1.1.7. ¿Qué diferencia hay entre instancias Distinct y Unique?	10
1.2. Cargue la base de datos audiology	11
1.2.1. Aplique el filtro filters/ unsupervised/ attribute/ NominalToBinary y describa como quedan ahora los atributos.	11
1.2.2. ¿Podría saber con antelación el número de atributos finales al aplicar este filtro?	12
1.2.3. ¿Qué ha pasado con algunos atributos nominales?	12
1.3. Particione su base de datos Criaturas tenebrosas usando filters/ supervised/ instance/ StratifiedRemoveFolds	12

1.3.1. Divida el dataset en train y test mediante un 3-fold . Describa el proceso realizado.	12
1.3.2. Divida el dataset en train y test mediante un 4-holdOut con un 75 % train y 25 % test. Describa el proceso realizado.	14
1.4. Criaturas tenebrosas como base de datos para los guiones	14
1.4.1. Construya a partir del fichero dataset371.csv un fichero .arff para Weka	14
1.4.2. Describa de forma concienzuda tanto los atributos como las clases	15
1.4.3. Observe si hay atributos identificadores. Si no existen diga por qué.	16
1.4.4. Use el entorno Visualice, ¿Hay alguna relación que sea visualmente significativa?	16
1.5. Describa con sus propias palabras los siguientes filtros No Supervisados	18
1.5.1. filters/unsupervised/attribute/Normalize	18
1.5.2. filters/unsupervised/attribute/ReplaceMissingValues	19
1.5.3. filters/unsupervised/attributes/NominalToBinary	20
1.5.4. filters/unsupervised/intance/RemoveDuplicates	20
1.5.5. filters/unsupervised/instance/Resample	22
1.5.6. filters/unsupervised/attribute/Remove	22
1.5.7. filters/unsupervised/attributes/RemoveUseless	23
1.6. Describa con sus propias palabras los siguientes filtros Supervisados .	23
1.6.1. filters/supervised/attribute/Discretize	23
1.6.2. filters/supervised/attribute/NominalToBinary	24
1.6.3. filters/supervised/intance/SpreadSubsample	24
1.6.4. filters/supervised/instance/Resample	26

1.7. Actualice su base de datos con los filtros anteriores que crea que tienen sentido con lo que sabe hasta ahora indicando cuales va a usar	28
1.7.1. filters/unsupervised/attribute/Remove	28
1.7.2. filters/unsupervised/attribute/NominalToBinary	28
1.7.3. filters/unsupervised/intance/RemoveDuplicates	29
1.7.4. filters/supervised/instance/StratifiedRemoveFolds	29
2. Práctica 2 - Clasificación y Regresión con Weka	30
2.1. Utilice el algoritmo de clasificación IBK con un 3-fold crossvalidation.	30
2.1.1. Use un valor de vecinos k=2 y deje por defecto el resto de los parámetros.	30
2.1.2. Interprete la salida en cuanto a los valores resumen de las métricas que proporciona Weka.	33
2.1.3. Tenga en cuenta si se clasifican bien todas las clases de su problema (TP Rate por clase). Comente este aspecto en función de la salida proporcionada por Weka (“Detailed Accuracy By Class”).	35
2.1.4. Fíjese en la matriz de confusión y haga una interpretación de la misma.	36
2.1.5. Con el botón derecho del ratón sobre la lista de resultados del panel izquierdo puede acceder también a gráficas. Comente lo que considere necesario sobre “Visualize Classifier Errors” y “Visualize Threshold Curve”.	37
2.2. Haga el mismo ejercicio, pero con un valor de vecinos k=3 y haga una interpretación de los resultados en comparación con k=2.	39

2.3. Con su base de datos, utilice el algoritmo de clasificación IBK con un 3-fold crossvalidation. Use un valor de vecinos k igual a valor con el que haya obtenido los mejores resultados.	41
2.3.1. Calcule la media y la desviación típica de las medidas Accuracy, Kappa, RMSE, F-Measure.	41
2.3.2. ¿Se corresponde el (TP Rate por clase) en el fichero csv con los valores que observó en el ejercicio usando el EXPLORER?	44
2.3.3. ¿Qué diferencia habría si hiciera el mismo ejercicio indicando que el número de repeticiones por fold fuese igual a 3? ¿Qué es lo que cambiará en cada repetición?	45
2.4. Usando el entorno EXPLORER ejecute el algoritmo Logistic con su base de datos, use un 3-fold crossvalidation como ya se hizo anteriormente.	46
2.4.1. Analice los modelos obtenidos, métricas, las variables que podrían ser más influyentes (valores beta), variables que no se usan, etc.	46
2.4.2. Asocie las fórmulas de las salidas por clase aportadas en las transparencias de esta práctica (transparencia titulada “Salidas por clase en Simplelogistic y Logistic de Weka”) con los modelos de probabilidad obtenidos en la salida de Weka.	47
2.5. Haga lo mismo usando el algoritmo SimpleLogistic y compare resultados	47
2.6. 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.	51
2.6.1. Pruebe con 2 y 3 clusters analizando y discutiendo los clusters usados.	51
2.6.2. Para cada valor del anterior, visualice los clusters resultantes al representar los atributos petallength y petalwidth y coméntelos.	53

2.7. Cargue su base de datos de criaturas tenebrosas, seleccione la opción “Use training set” y analice qué ocurre al aplicar K-means, fijando k=número-de-clases de la base de datos.	55
2.8. ¿Qué ocurriría en K-means si fijásemos el número de clúster igual al número de patrones de una base de datos?	56
2.9. Utilizando su base de datos criaturas tenebrosas, el algoritmo K-means, y seleccionando la opción Classes to clusters evaluation, ¿con qué número de clusters obtiene mejores resultados?.	56
2.10. Ejecute el algoritmo HierarchicalClusterer con tipo de link Complete y métrica de distancia euclídea. ¿Cuáles de ellos producen los grupos mejor diferenciados y con fronteras claras?	57
2.11. Ejecute el algoritmo HierarchicalClusterer sobre su base de datos criaturas tenebrosas	59
2.11.1. ¿Con qué linkType obtiene los mejores resultados en cuanto a instancias mal agrupadas? Pruebe con Single y Complete. . .	59
2.11.2. Analice los clusters creados con cada linkType y sus asignaciones.	60
3. Práctica 3 - Árboles y Redes Neuronales	62
3.1. Cargue su base de datos y ejecute el algoritmo C4.5 usando un 75 % para entrenar y un 25 % para generalizar.	62
3.1.1. Analice y muestre el árbol obtenido con los parámetros por defecto: nodo principal, número de nodos u hojas, variables presentes y omitidas.	62
3.1.2. Analice y muestre cómo cambia el árbol, al modificar los siguientes parámetros. Comente también los resultados de las métricas obtenidas	65

3.2. Utilizando su base de datos con un 75/25 % y el algoritmo MultilayerPerceptron	67
3.2.1. ¿Cuál sería el valor por defecto para el atributo hidden Layers en su base de datos?	67
3.2.2. Con los valores por defecto, ¿qué observa al ir modificando solo el learningRate?	67
3.2.3. Con los valores por defecto, ¿qué observa al ir modificando solo el momentum?	68
3.2.4. Con los valores por defecto, ¿qué observa al ir modificando solo el training-Time? Realice una gráfica que muestre cómo cambia el valor de CorrectlyClassified instances al modificar el parámetro trainingTime, de forma que localice con cuántas épocas se estanca el aprendizaje o incluso se empieza a sobreentrenar.	70
4. Práctica 4 - Preprocesamiento	71
4.1. Preprocesamiento en conjunto de train	72
4.1.1. Tratamiento de atributos	72
4.1.2. Correlación de atributos	73
4.1.3. Detección y tratamiento de outliers	77
4.1.4. Normalización	77
4.2. Preprocesamiento en conjunto de test	78
4.2.1. Tratamiento de atributos	78
4.2.2. Normalización	78
4.2.3. Construcción del modelo	79
Bibliografía	85

Práctica 1 - Introducción a Weka

1.1. Cargue la base de datos Iris. Observe los atributos.

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

Se encuentran 4 atributos dentro de esta clase, y son los siguientes: sepallength, sepalwidth, petallength, petalwidth.

1.1.2. ¿Se trata de regresión o clasificación?

Es un proceso de clasificación en el cual mediante unos datos dados por un experto. Posteriormente el modelo se entrenará con estos datos y será capaz de clasificar

No.	Name
1	sepallength
2	sepalwidth
3	petallength
4	petalwidth
5	class

Figura 1.1: Datos

el tipo de planta: Iris Setosa, Iris Versicolor, Iris Virginica.

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

Es un proceso de clasificación en el cual mediante unos datos dados por un experto. Posteriormente el modelo se entrenará con estos datos y será capaz de clasificar el tipo de planta: Iris Setosa, Iris Versicolor, Iris Virginica.

- Rango de valores: 0.1 – 2.5
- Media: 1.199
- Desviación Típica: 0.763

Statistic	Value
Minimum	0.1
Maximum	2.5
Mean	1.199
StdDev	0.763

Figura 1.2: Rango de valores, media y desviación típica en Weka

1.1.4. Determinar que atributo permite discriminar linealmente entre la clase Iris Setosa y las otras dos clases.

Cómo podemos observar mediante la figura 1.4, un atributo que nos permite establecer una recta que separe perfectamente a las clases de iris-setosa, indicada en azul oscuro, con el resto, sería el atributo petallength.

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

No sería posible la separación lineal, debido a que las muestras de datos obtenidas son muy similares, por tanto, en caso de establecer una separación lineal, podría incurrir en establecer un patrón como una clase errónea o bien trazar una línea que divida un patrón en dos mitades.

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

Emplearía el par de atributos petalwidth y sepallength, debido a que es la más cercana a diferenciar las tres posibles clases con el menor error posible, véase figura 1.4.

1.1.7. ¿Qué diferencia hay entre instancias Distinct y Unique?

La diferencia entre Distinct y Unique radica en que, unique sólo cuenta los valores que han aparecido una sola vez, mientras que Distinct nos indica el número de patrones que encontramos en un dataset eliminando aquellos repetidos. Haciendo uso de la base de datos de la flor Iris, se han establecido 5 patrones, de los cuales 4 son completamente distintos y el último es un patrón similar a uno ya establecido previamente, mediante la figura 1.5 se puede ver cómo hay 4 valores distintos (Nº 1-4), debido que se ha eliminado el último patrón al estar repetido. Por tanto, encontramos únicamente 3 patrones únicos debido que aparecen sólo una vez en nuestro dataset.

1.2. Cargue la base de datos audiology

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

Al aplicar este filtro, la base de datos varía en cuanto a que los atributos que sean nominales se transformarán en atributos numéricos binarios. Es decir, los n valores que pueda tomar un atributo, se transformaran en n atributos binarios.

No.		Name
1	<input type="checkbox"/>	age_gt_60=t
2	<input type="checkbox"/>	air=mild
3	<input type="checkbox"/>	air=moderate
4	<input type="checkbox"/>	air=normal
5	<input type="checkbox"/>	air=profound
6	<input type="checkbox"/>	air=severe
7	<input type="checkbox"/>	airBoneGap=t
8	<input type="checkbox"/>	ar_c=absent
9	<input type="checkbox"/>	ar_c=elevated
10	<input type="checkbox"/>	ar_c=normal
11	<input type="checkbox"/>	ar_u=absent
12	<input type="checkbox"/>	ar_u=elevated
13	<input type="checkbox"/>	ar_u=normal
14	<input type="checkbox"/>	bone=mild
15	<input type="checkbox"/>	bone=moderate
16	<input type="checkbox"/>	bone=normal
17	<input type="checkbox"/>	bone=unmeasured
18	<input type="checkbox"/>	boneAbnormal=t
19	<input type="checkbox"/>	bser=normal
20	<input type="checkbox"/>	history_buzzing=t
21	<input type="checkbox"/>	history_dizziness=t
22	<input type="checkbox"/>	class

Figura 1.3: Filtro unsupervised/NominalToBinary

El filtro NominalToBinary supervisado se aplica cuando la clase es numérica y esto generará n-1 atributos binarios teniendo en cuenta el valor promedio de la clase.

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

Esto será posible por lo explicado previamente, por cada valor de un atributo nominal se generará un nuevo atributo haciendo esto que a priori sea posible saber cuantos atributos se generarán.

1.2.3. ¿Qué ha pasado con algunos atributos nominales?

Al aplicar el filtro los atributos nominales son convertidos en atributos numéricos binarios, sin embargo aquellos que toman solo dos valores generan n-1 atributos, donde n en este caso sería 2, por lo que un valor se identifica con 1.0 y el otro valor con 0.0

1.3. Particione su base de datos Criaturas tenebrosas usando filters/ supervised/ instance/ StratifiedRemoveFolds

1.3.1. Divida el dataset en train y test mediante un 3-fold . Describa el proceso realizado.

Mediante el filtro mencionado se ha realizado una configuración del mismo estableciendo el parámetro numFolds a 3. Posteriormente se escoge que fold se usará para test y se guardará para su posterior uso. Realizado esto, se deben deshacer los

cambios y establecer el parámetro de invertSelection a true. De esta forma cogerá los dos fold restantes, que se usarán para entrenamiento. Este proceso se realizará tantas veces como numFolds hayamos indicado, modificando a cada vez el fold con el cual estemos trabajando. El parámetro de la semilla deberá permanecer constante a lo largo de este proceso.

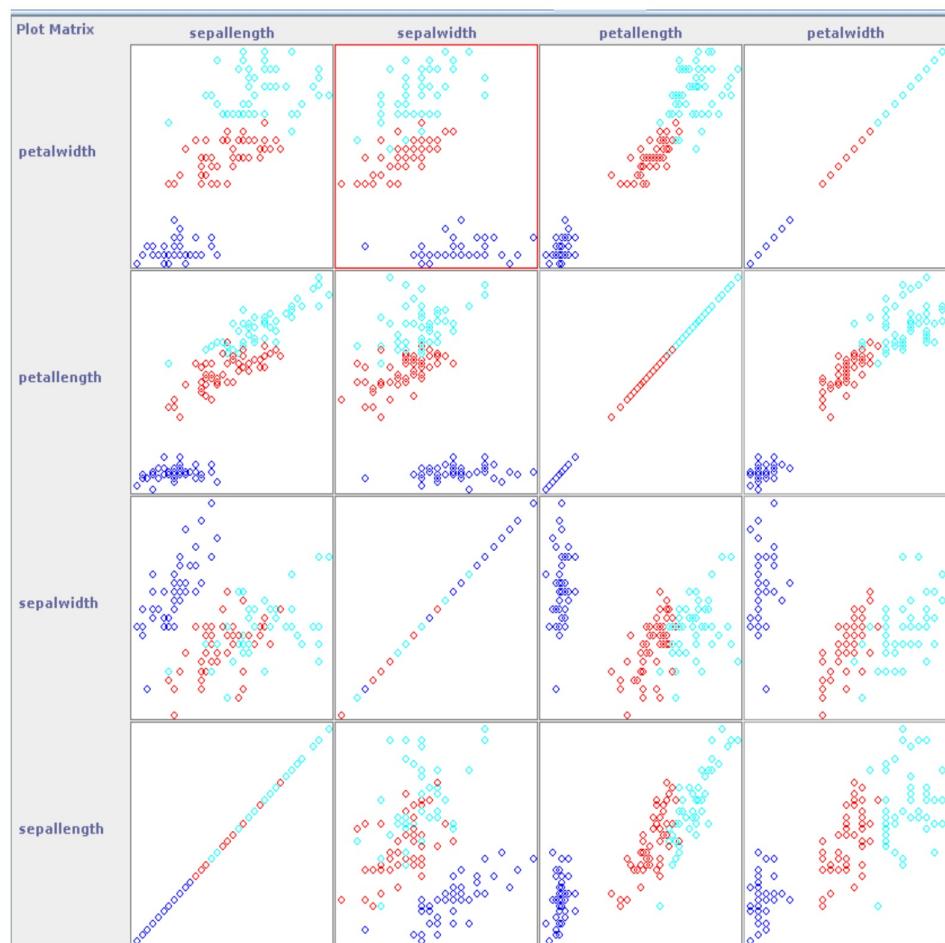


Figura 1.4: Filtro supervised/instance/StratifiedRemoveFolds

1.3.2. Divida el dataset en train y test mediante un 4-holdOut con un 75 % train y 25 % test. Describa el proceso realizado.

Para realizar esto, haciendo uso del filtro mencionado se configura el mismo estableciendo el parámetro numFolds a 4. Posteriormente se escoge que fold se usará para test y se guardará para su posterior uso. El proceso continua igual que el apartado anterior, pero si queremos seleccionar 75 % y 25 % habrá que escoger un fold para test y 3 fold para train. A diferencia del apartado anterior se deberá ir variando la semilla por cada holdOut, para de esta forma variar las muestras escogidas en cada división.

1.4. Criaturas tenebrosas como base de datos para los guiones

1.4.1. Construya a partir del fichero dataset371.csv un fichero .arff para Weka

Aunque Weka dispone de compatibilidad para cargar bases de datos a partir de ficheros con formato CSV, es decir, podríamos cargar un fichero CSV con todos los datos y atributos sin necesidad de una conversión previa al formato arff.

Sin embargo, si deseásemos hacer esto de forma manual, es importante conocer el formato de Weka y el formato CSV.

El formato CSV se compone de una primera fila que será el nombre de cada atributo de datos, y las posteriores filas serán los datos que tomarán estos atributos, correspondiendo a cada fila un patrón distinto.

El formato Weka, que al que deseamos convertir deberemos de definir cada atributo como '@attribute *nombre del atributo'*'. En caso de ser un atributo nominal y tomar una serie de valores, estos se indicarán justamente después mediante llaves. Si el atributo fuese numérico simplemente indicamos posteriormente que es de tipo '*numeric*'.

Para indicar el comienzo de los datos se hará mediante '@data'. Cada patrón deberá tener los valores de los atributos separados por comas. Para diferenciar los patrones entre ellos, se hará mediante un salto de linea.

1.4.2. Describa de forma concienzuda tanto los atributos como las clases

Los atributos de este fichero son los siguientes:

- **id**: Este atributo, se eliminará posteriormente ya que no aporta información alguna.
- **bone length**: Este atributo define la longitud de los huesos de cada criatura, siendo esta un valor numérico continuo. Este atributo se encuentra normalizado.
- **rotting flesh**: Este atributo define la cantidad de carne podrida que tiene cada criatura, siendo esta un valor numérico continuo. Representa un porcentaje. Este atributo se encuentra normalizado.
- **hair length**: Este atributo define la longitud del pelo de cada criatura, siendo esta un valor numérico continuo. Este atributo se encuentra normalizado.
- **has_soul**: Este atributo define la cantidad de alma que tiene cada criatura, siendo esta un valor numérico continuo. Este atributo se encuentra normalizado.

- **color:** Este atributo define el color de cada criatura, siendo este un atributo nominal que toma los valores: clear, green, black, white, blue, blood.
- **type:** Este atributo es la clase, define el tipo de criatura identificado, pudiendo tomar como valores: ghoul, ghost y goblin.

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

Existe un atributo identificador denominado id, que se eliminará ya que no aporta información alguna.

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

Mediante el entorno Visualize, podemos ver todas las muestras en un plano de coordenadas. Cada color representa una clase distinta. La relación más significativa que se ha encontrado ha sido representando los atributos hair_length y has_soul, como muestra la 1.5, cuando los valores de alma-pelo son cercanos a 0, es más probable que se trate de un ghost, sin embargo, a mayor cantidad de pelo-alma es más probable que sea un ghoul.

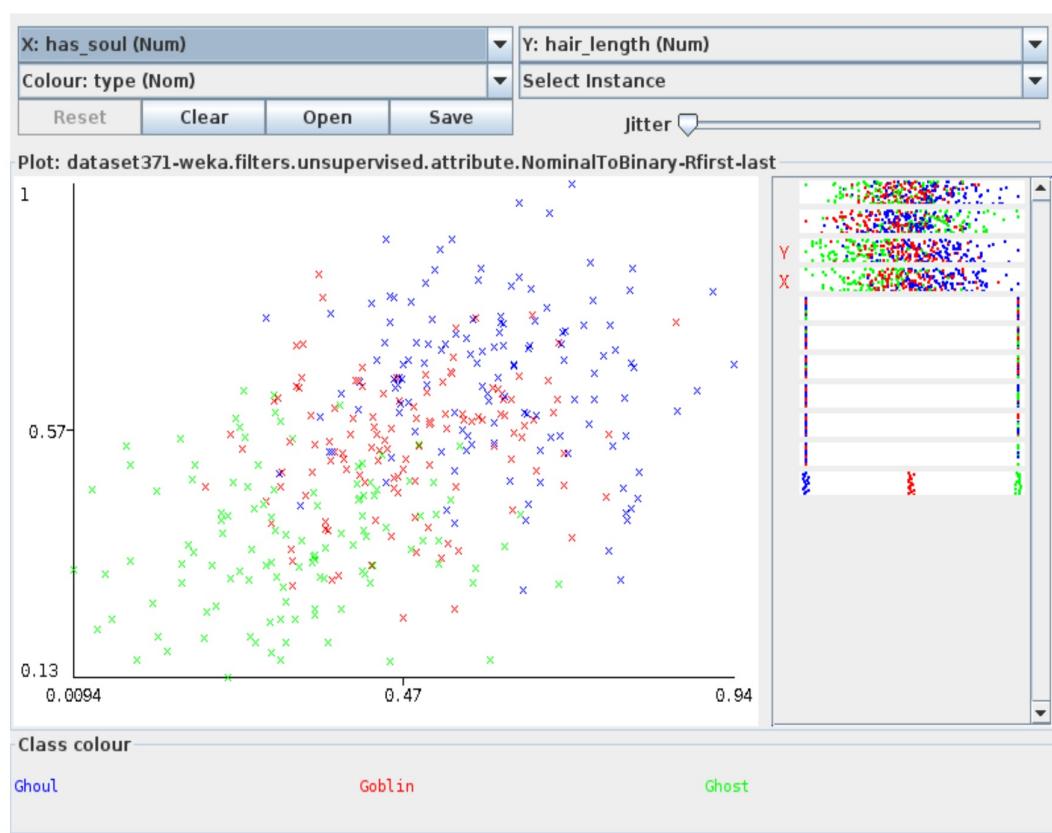


Figura 1.5: Representación de datos

1.5. Describa con sus propias palabras los siguientes filtros No Supervisados

1.5.1. filters/unsupervised/attribute/Normalize

Normaliza todos los valores numéricos del dataset a valores por defecto entre [0,1], el valor indicado en translation, será el valor de inicio del rango en el cual se normalizará, mientras que scale, será el propio rango de normalización. En el caso del dataset de criaturas, los valores ya se encuentran normalizados, por tanto, una segunda normalización no tendría efecto alguno.

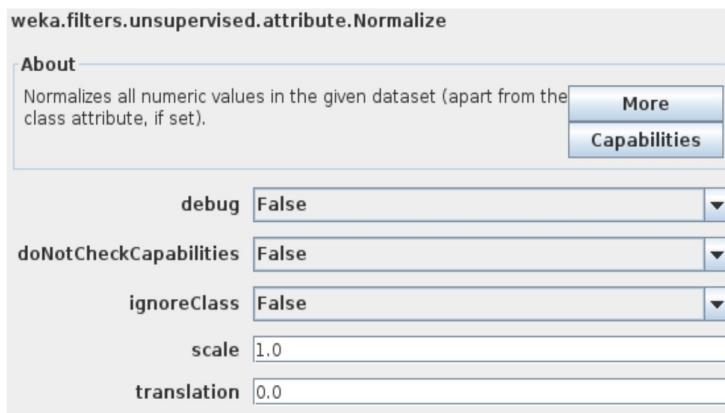


Figura 1.6: Representación de datos

1.5.2. filters/unsupervised/attribute/ReplaceMissingValues

Para ello se ha insertado un patrón con datos vacíos, como se puede ver en la figura 1.7, de tal forma que al aplicar el filtro debería corregir dicha anomalía, como se ve en la figura 1.8.

Selected attribute	Distinct	Type
Name: bone_length Missing: 1 (0%)	371	Numeric Unique: 371 (100%)

Figura 1.7: Dataset antes de aplicar el filtro ReplaceMissingValues

El funcionamiento que realiza es que aquellos patrones que tengan algún valor vacío será reemplazado por la media o la moda en caso de ser nominal. Sin embargo este proceso sería mucho mas preciso, si hiciéramos una regresión lineal.

Selected attribute	Distinct	Type
Name: bone_length Missing: 0 (0%)	372	Numeric Unique: 372 (100%)

Figura 1.8: Dataset después de aplicar el filtro ReplaceMissingValues

1.5.3. filters/unsupervised/attributes/NominalToBinary

Al aplicar este filtro, la base de datos varía en cuanto a que los atributos que sean nominales se transformarán en atributos numéricos binarios. Es decir, los n valores que pueda tomar un atributo, se transformaran en n atributos binarios. Esto se puede ver en la figura 1.3.

1.5.4. filters/unsupervised/intance/RemoveDuplicates

La funcionalidad de este filtro es la de eliminar todos aquellos patrones o instancias que se encuentren repetidas. De esta forma eliminamos la duplicidad de datos.

Selected attribute			
Name:	id	Type:	Numeric
Missing:	0 (0%)	Distinct:	6
Unique: 4 (50%)			
Viewer			
Relation: Universidad			
No.	1: id Numeric	2: Nota Numeric	3: class Nominal
1	1.0	3.0	PL
2	2.0	9.0	IAA
3	3.0	6.0	MH
4	4.0	8.2	PL
5	5.0	5.5	LE
6	6.0	0.0	MH
7	5.0	5.5	LE
8	4.0	8.2	PL

Figura 1.9: Dataset antes de aplicar el filtro RemoveDuplicates

Como se puede ver tras aplicar el filtro los patrones unicos, varían de 4 a 6. Esto nos indica que ha eliminado ciertos patrones que estaban duplicados.

Selected attribute			
	Name: id	Type: Numeric	
	Missing: 0 (0%)	Distinct: 6	Unique: 6 (100%)
Relation: Universidad-weka.filters.unsupervised.instance.RemoveDuplicates			
No.	1: id Numeric	2: Nota Numeric	3: class Nominal
1	1.0	3.0	PL
2	2.0	9.0	IAA
3	3.0	6.0	MH
4	4.0	8.2	PL
5	5.0	5.5	LE
6	6.0	0.0	MH

Figura 1.10: Dataset después de aplicar el filtro RemoveDuplicates

1.5.5. filters/unsupervised/instance/Resample

Este filtro permite equilibrar el conjunto de datos, generando nuevos datos sintéticos para la clase con menor número de muestras. Estos datos se generarán de forma aleatoria, sesgando el conjunto de datos para una clase de las que hay pocas muestras disponibles.

1.5.6. filters/unsupervised/attribute/Remove

Este filtro se encarga de eliminar un atributo del dataset, por ejemplo si deseásemos borrar la id del dataset de criaturas, lo haríamos mediante este filtro.

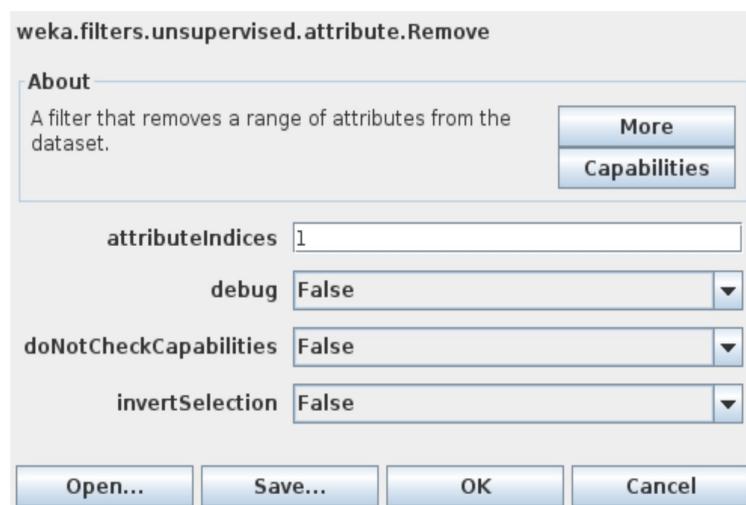


Figura 1.11: Propiedades del filtro

1.5.7. filters/unsupervised/attributes/RemoveUseless

1.6. Describa con sus propias palabras los siguientes filtros Supervisados

1.6.1. filters/supervised/attribute/Discretize

Este filtro se encarga de discretizar todos los atributos numéricos continuos. Para ello estos atributos tomarán rangos de valores, en vez de un único valor, sacrificando precisión pero transformándolos en valores discretos. Estos valores convertidos serían nominales.

No.	1: bone_length Numeric	2: rotting_flesh Numeric	No.	1: bone_length Nominal	2: rotting_flesh Nominal	3: hair_length Nominal
1	0.354512	0.350839	1	'(0.2-0.41]'	'(-inf-0.54]'	'(0.39-0.51]'
2	0.57556	0.425868	2	'(0.41-0.63]'	'(-inf-0.54]'	'(0.51-0.68]'
3	0.467875	0.35433	3	'(0.41-0.63]'	'(-inf-0.54]'	'(0.68-inf)'
4	0.776652	0.508723	4	'(0.63-inf)'	'(-inf-0.54]'	'(0.51-0.68]'
5	0.566117	0.875862	5	'(0.41-0.63]'	'(0.54-inf)'	'(0.39-0.51]'
6	0.40568	0.253277	6	'(0.2-0.41]'	'(-inf-0.54]'	'(0.39-0.51]'
7	0.399331	0.568952	7	'(0.2-0.41]'	'(0.54-inf)'	'(0.51-0.68]'
8	0.516224	0.536429	8	'(0.41-0.63]'	'(-inf-0.54]'	'(0.51-0.68]'
9	0.314295	0.67128	9	'(0.2-0.41]'	'(0.54-inf)'	'(0.39-0.51]'
10	0.280942	0.701457	10	'(0.2-0.41]'	'(0.54-inf)'	'(-inf-0.39]'
11	0.431685	0.438959	11	'(0.41-0.63]'	'(-inf-0.54]'	'(-inf-0.39]'
12	0.584543	0.593082	12	'(0.41-0.63]'	'(0.54-inf)'	'(0.68-inf)'
13	0.390712	0.335069	13	'(0.2-0.41]'	'(-inf-0.54]'	'(0.51-0.68]'
14	0.351559	0.471078	14	'(0.2-0.41]'	'(-inf-0.54]'	'(0.39-0.51]'
15	0.513387	0.301345	15	'(0.41-0.63]'	'(-inf-0.54]'	'(0.68-inf)'
16	0.500197	0.438418	16	'(0.41-0.63]'	'(-inf-0.54]'	'(0.51-0.68]'
17	0.25077	0.246258	17	'(0.2-0.41]'	'(-inf-0.54]'	'(0.51-0.68]'
18	0.585559	0.585939	18	'(0.41-0.63]'	'(0.54-inf)'	'(0.68-inf)'
19	0.605836	0.587943	19	'(0.41-0.63]'	'(0.54-inf)'	'(0.51-0.68]'
20	0.52408	0.750988	20	'(0.41-0.63]'	'(0.54-inf)'	'(0.51-0.68]'
21	0.503164	0.464055	21	'(0.41-0.63]'	'(-inf-0.54]'	'(0.39-0.51]'
22	0.472603	0.427151	22	'(0.41-0.63]'	'(-inf-0.54]'	'(0.51-0.68]'
23	0.374449	0.384183	23	'(0.2-0.41]'	'(-inf-0.54]'	'(0.51-0.68]'
24	0.34335	0.39745	24	'(0.2-0.41]'	'(-inf-0.54]'	'(0.39-0.51]'
25	0.687256	0.301576	25	'(0.63-inf)'	'(-inf-0.54]'	'(0.68-inf)'
26	0.22901	0.567313	26	'(0.2-0.41]'	'(0.54-inf)'	'(-inf-0.39]'
27	0.388501	0.342306	27	'(0.2-0.41]'	'(-inf-0.54]'	'(0.51-0.68]'
28	0.492438	0.623197	28	'(0.41-0.63]'	'(0.54-inf)'	'(0.51-0.68]'
29	0.411663	0.454075	29	'(0.41-0.63]'	'(-inf-0.54]'	'(0.39-0.51]'

Figura 1.12: Antes y después de la aplicación del filtro

1.6.2. filters/supervised/attribute/NominalToBinary

El filtro NominalToBinary supervisado se aplica cuando la clase es numérica y esto generará n-1 atributos binarios teniendo en cuenta el valor promedio de la clase.

1.6.3. filters/supervised/instance/SpreadSubsample

Este filtro generará un nuevo conjunto de datos de forma aleatoria. Este nuevo conjunto se puede configurar variando los parámetros del filtro. Los parámetros son

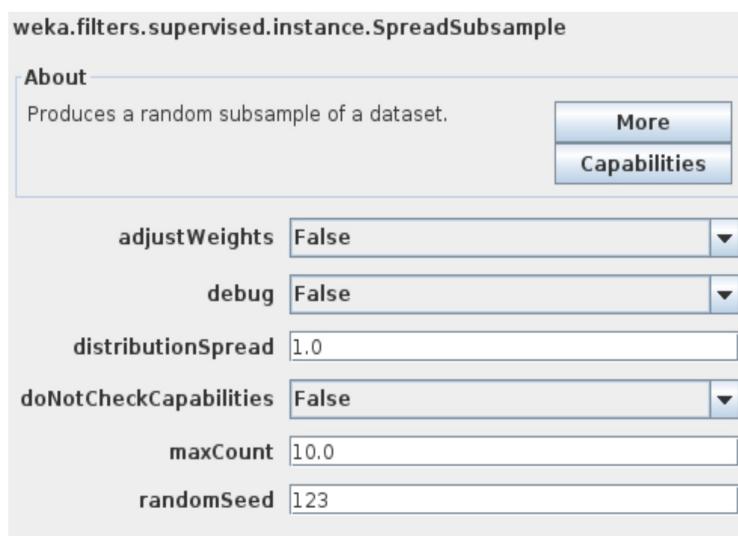


Figura 1.13: Filtro SpreadSubsample

los siguientes:

- **distributionSpread**: Permite indicar la variación de la distribución de cada clase, en caso de ser 1 generará una distribución uniforme.
- **maxCount**: Permite indicar el número máximo de instancias por cada clase.
- **randomSeed**: Semilla para generar distintos conjuntos.

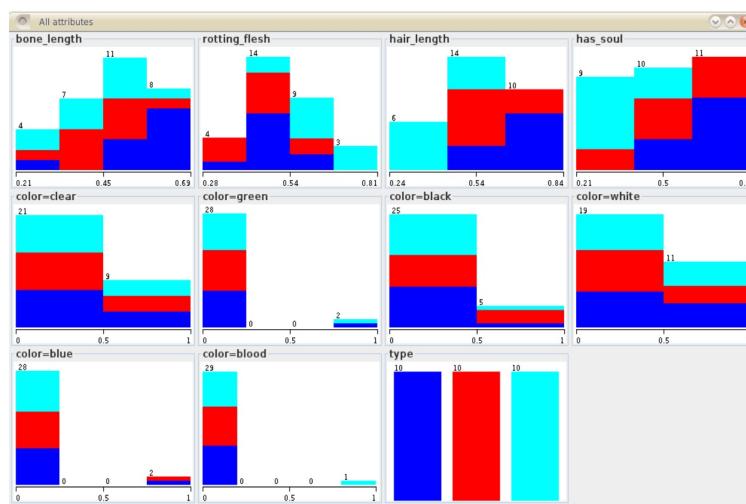


Figura 1.14: Resultados de la aplicación del filtro

1.6.4. filters/supervised/instance/Resample

Este filtro produce un nuevo subconjunto de datos del dataset, siendo necesario que la clase sea de tipo nominal. Dependiendo de los parámetros se generará un subconjunto distinto.

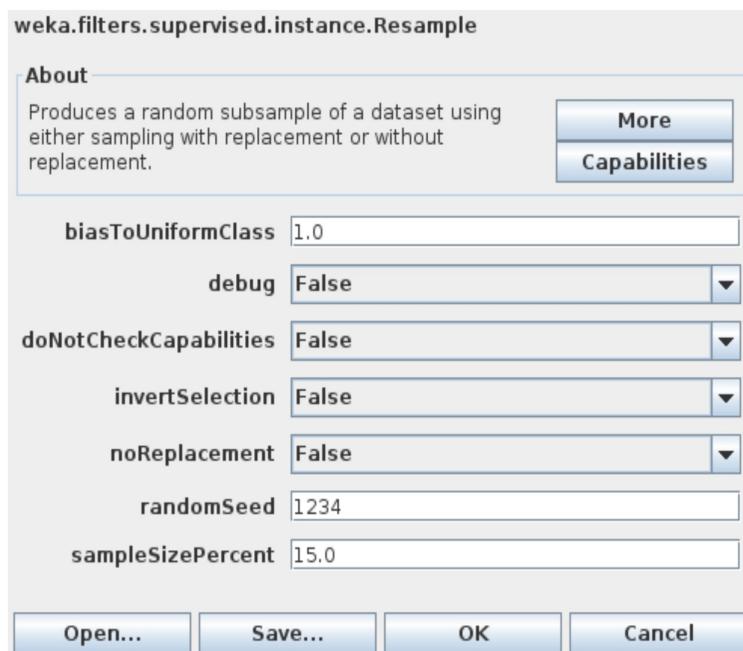


Figura 1.15: Filtro Resample

Los parámetros se detallan a continuación:

- **biasToUniformClass**: Permite indicar la distribución que tendrá el nuevo conjunto.
- **sampleSizePercent**: Permite indicar el porcentaje del dataset que se usará para el subconjunto.
- **noReplacement**: Permite introducir reemplazamiento de instancias dentro del nuevo subconjunto.
- **randomSeed**: Semilla para generar distintos conjuntos.

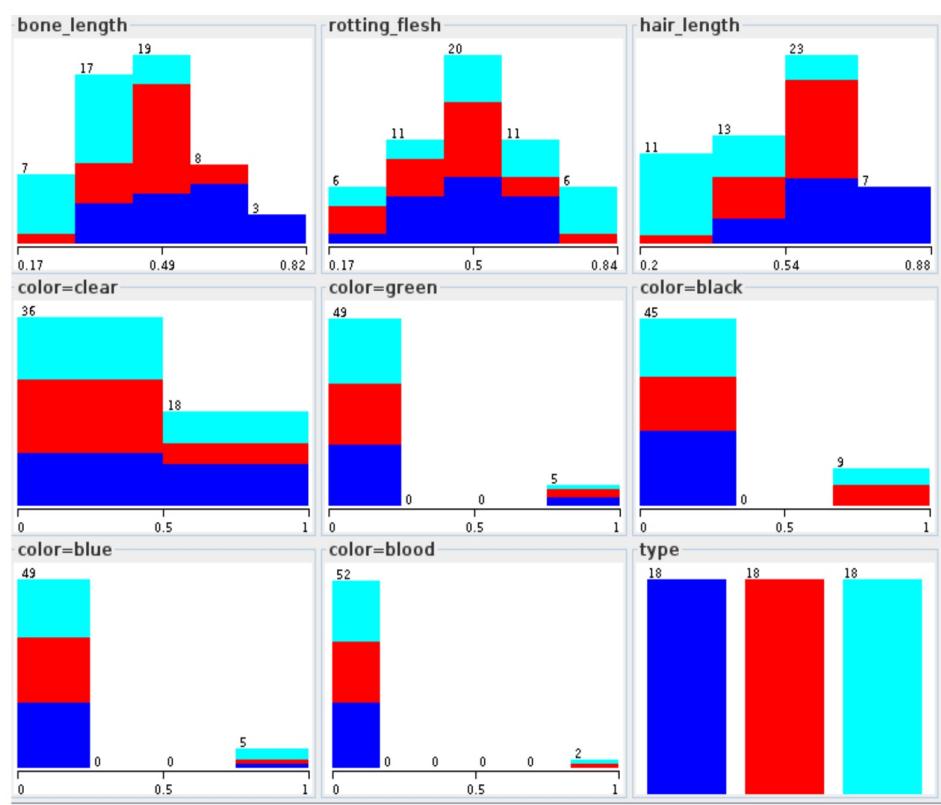


Figura 1.16: Resultados de la aplicación del filtro

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

Trabajando con el dataset de criaturas, es necesario realizar la aplicación de una serie de filtro para poder trabajar con el posteriormente para la generación de un modelo de clasificación.

Los filtros usados se explicarán por orden de aplicación.

1.7.1. filters/unsupervised/attribute/Remove

Este filtro explicado previamente, se usará para la eliminación del atributo 'íd', que existe en el dataset, que no aporta información alguna. Para su aplicación debemos de indicar el indice del atributo, que en este caso será el número 1. Esto se puede ver en la figura 1.11.

1.7.2. filters/unsupervised/attribute/NominalToBinary

En el dataset escogido, se encuentra el atributo 'color'. Este atributo es de carácter nominal por lo que es necesario transformarlo en binario para poder trabajar posteriormente con el dataser. La aplicación de este filtro, aunque en el dataset audiology, la podemos ver en la figura 1.3.

1.7.3. filters/unsupervised/instance/RemoveDuplicates

Como el dataset tiene una amplia cantidad de muestras como para conocer si existen muestras duplicadas, se aplica este filtro que no permitirá eliminar todas aquellas que estén duplicadas.

1.7.4. filters/supervised/instance/StratifiedRemoveFolds

Para la generación del modelo será necesario la creación de los conjuntos de entrenamiento y test. Como existen distintas técnicas para esto, se ha escogido la realización de un 4-holdOut donde se usará un 75 % para entrenamiento y un 25 % para test. La aplicación del filtro se puede ver en la figura 1.4, será necesario variar la semilla, por cada holdOut. Cada resultado será guardado y nombrado de forma legible para su posterior uso.

Práctica 2 - Clasificación y Regresión con Weka

2.1. Utilice el algoritmo de clasificación IBK con un 3-fold crossvalidation.

2.1.1. Use un valor de vecinos k=2 y deje por defecto el resto de los parámetros.

Se ha procedido a realizar la configuración por defecto obteniendo un 63% de instancias correctamente clasificadas, con una precisión media de 0.66. Sin embargo, tras realizar una serie de modificaciones en los parámetros del algoritmo obteniendo así una mayor tasa de instancias correctamente clasificadas y reduciendo el error de clasificación, de forma que se ha conseguido modificando:

- DistanceWeighting: Permite establecer un método de ponderación basado en pesos, en nuestro caso hemos elegido que sea $1/\text{distancia}$, donde distancia será la distancia euclídea entre el punto a clasificar y los K vecinos más cercanos.

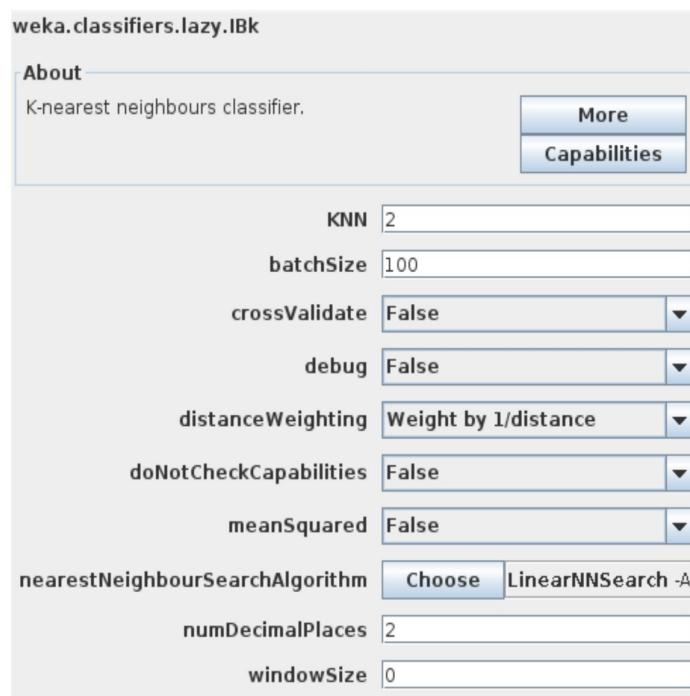


Figura 2.1: Parámetros introducidos para la ejecución de IBK

La figura 2.2 nos muestra los resultados obtenidos con los parámetros establecidos, los cuales podemos ver en la figura 2.1. En esta se puede apreciar que el valor de k establecido es 2.

```
==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances      235           63.3423 %
Incorrectly Classified Instances   136           36.6577 %
Kappa statistic                   0.4465
Mean absolute error               0.2474
Root mean squared error          0.4178
Relative absolute error           55.7041 %
Root relative squared error     88.6698 %
Total Number of Instances        371

==== Detailed Accuracy By Class ====
      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC    ROC Area  PRC Area  Class
      0,853    0,285    0,615     0,853    0,714     0,541   0,815    0,641    Ghoul
      0,464    0,240    0,496     0,464    0,479     0,228   0,668    0,471    Goblin
      0,573    0,031    0,893     0,573    0,698     0,626   0,901    0,769    Ghost
Weighted Avg.      0,633    0,190    0,662     0,633    0,630     0,462   0,792    0,624

==== Confusion Matrix ====
      a     b     c  <-- classified as
110  19   0 |   a = Ghoul
  59  58   8 |   b = Goblin
  10  40  67 |   c = Ghost
```

Figura 2.2: Resultados de IBK con valor de k igual a 2

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

- Correctly Classified Instances: Muestra el porcentaje de instancias que han sido correctamente clasificadas.
- Incorrectly Classified Instances: Muestra el porcentaje de instancias que han sido erróneamente clasificadas.
- Kappa statistic: Es una medida estadística que ajusta el efecto del azar entre las clasificaciones y las clases correctamente interpretadas, siendo un valor mayor a cero que el clasificador es mejor que lanzar una moneda al aire.
- Tasas de error: Son usadas para la predicción numérica en lugar de la clasificación, tal que permiten reflejar la magnitud del error.
- TP Rate: Tasa de verdaderos positivos o instancias clasificadas correctamente para una clase.
- FP Rate: Tasa de falsos positivos o instancias erróneamente clasificadas como una clase dada.
- Precisión: Proporción de instancias que son correctas de una clase y dividida por el total de instancias clasificadas como esa clase.
- Recall: Proporción de instancias clasificadas como una clase dada y dividida por el total real en esa clase.
- F-Measure: Medida de precisión y recuperación obtenida mediante $2 * \text{Precisión} * \text{Recall} / (\text{Precisión} + \text{Recall})$
- ROC Area: Se usa para modelos que predigan la probabilidad de que un patrón pertenezca a la clase Positiva

En la figura 2.3 podemos ver los resultados obtenidos con los mismos parámetros establecidos y mostrados en la figura 2.1, sin embargo, el valor de k varía de 2 a 3. Y como se puede visualizar los resultados son mejores ya que un valor de k igual a 3 coincide con el número de clases existentes en el problema.

```

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

    Correctly Classified Instances      245           66.0377 %
    Incorrectly Classified Instances   126           33.9623 %
    Kappa statistic                   0.4898
    Mean absolute error              0.2449
    Root mean squared error          0.4188
    Relative absolute error          55.1519 %
    Root relative squared error     88.8753 %
    Total Number of Instances        371

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

    TP Rate   FP Rate   Precision   Recall   F-Measure   MCC      ROC Area   PRC Area   Class
    0,651     0,165     0,677      0,651    0,664      0,491    0,801     0,656     Ghoul
    0,576     0,276     0,514      0,576    0,543      0,292    0,670     0,517     Goblin
    0,761     0,071     0,832      0,761    0,795      0,708    0,911     0,837     Ghost
    Weighted Avg.       0,660     0,173     0,671      0,660    0,665      0,492    0,792     0,666

==== Confusion Matrix ====

    a   b   c   <- classified as
84 43  2 |  a = Ghoul
37 72 16 |  b = Goblin
 3 25 89 |  c = Ghost

```

Figura 2.3: Resultados de IBK con valor de k igual a 3

2.1.3. Tenga en cuenta si se clasifican bien todas las clases de su problema (TP Rate por clase). Comente este aspecto en función de la salida proporcionada por Weka (“Detailed Accuracy By Class”).

El porcentaje de instancias correctamente clasificadas suele denominarse precisión o exactitud de la muestra, sin embargo, este parámetro cuenta con algunas desventajas. Una de estas desventajas sería la falta de sensibilidad respecto a clases desbalanceadas, por lo que nos interesa fijarnos también en otros valores, como pueden ser el área bajo la curva ROC, medida bastante efectiva y precisa. En el caso

== Detailed Accuracy By Class ==									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,651	0,165	0,677	0,651	0,664	0,491	0,801	0,656	Ghoul
	0,576	0,276	0,514	0,576	0,543	0,292	0,670	0,517	Goblin
	0,761	0,071	0,832	0,761	0,795	0,708	0,911	0,837	Ghost
Weighted Avg.	0,660	0,173	0,671	0,660	0,665	0,492	0,792	0,666	

Figura 2.4: Resultados por clase con valor de k igual a 2

mostrado de la figura 2.4, se puede observar que la clasificación de la clase Ghost es relativamente buena seguida de los Ghoul, sin embargo, la clase goblin es ligeramente mejor a la probabilidad de lanzar una moneda al aire. De esta forma podemos observar que la precisión obtenida en la clase Ghost es muy alta, así como la tasa de falsos positivos obtenida en dicha clase es muy baja, es decir ha clasificado pocas instancias erróneamente y al tener una alta precisión ha clasificado correctamente muchas instancias. A su vez, podemos fijarnos en el AUC, observando que la clase Ghost tiene un área cercana a la unidad, por tanto está cercana a un modelo perfecto, sin embargo Goblin se encuentra más cercana a 0.5, por tanto tendrá menor precisión.

2.1.4. Fíjese en la matriz de confusión y haga una interpretación de la misma.

Mediante la matriz de confusión podemos describir el rendimiento de un determinado modelo o clasificador. Dada la figura (anterior), podemos observar como no se han clasificado correctamente todos los patrones de generalización, de forma que:

- Clase Ghoul: Ha clasificado correctamente como ghoul a 84 patrones, así como también ha determinado que 43 que son de clase Ghoul los ha clasificado erróneamente como Goblin y 2 como Ghost.
- Clase Goblin: Al igual que en el anterior caso, ha clasificado 37 goblins como Ghouls, ha clasificado correctamente 72 Goblin y por último clasificado erróneamente a 16 goblins como Ghost.
- Clase Ghost: Nuestro clasificador ha determinado que 3 ghost son clasificados como Ghoul, 25 como Goblin y 89 han sido correctamente clasificados.

2.1.5. Con el botón derecho del ratón sobre la lista de resultados del panel izquierdo puede acceder también a gráficas. Comente lo que considere necesario sobre “Visualize Classifier Errors” y “Visualize Threshold Curve”.

Mediante la herramienta ‘Visualize Classifier Errors’ podemos ver todos los patrones en un plano de coordenadas. Aquellos patrones representados con un cuadrado son aquellos que han sido mal clasificados, sin embargo, los representados con una cruz, son aquellos que han sido clasificados de forma correcta.

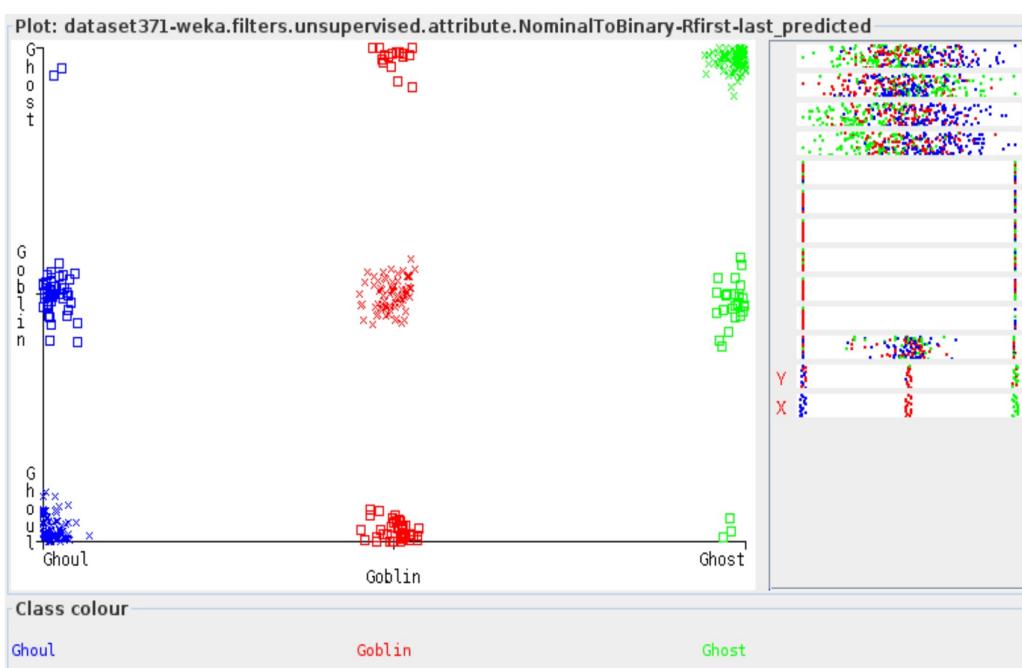


Figura 2.5: Herramienta Visualize Classifier Error con valor de k igual a 2

La herramienta 'Visualize Threshold Curveños permite visualizar la curva ROC, aunque también nos ofrece el área bajo la curva ROC siendo esto más claro que la propia visualización. Esta curva representa el área bajo la curva ROC de la clase que

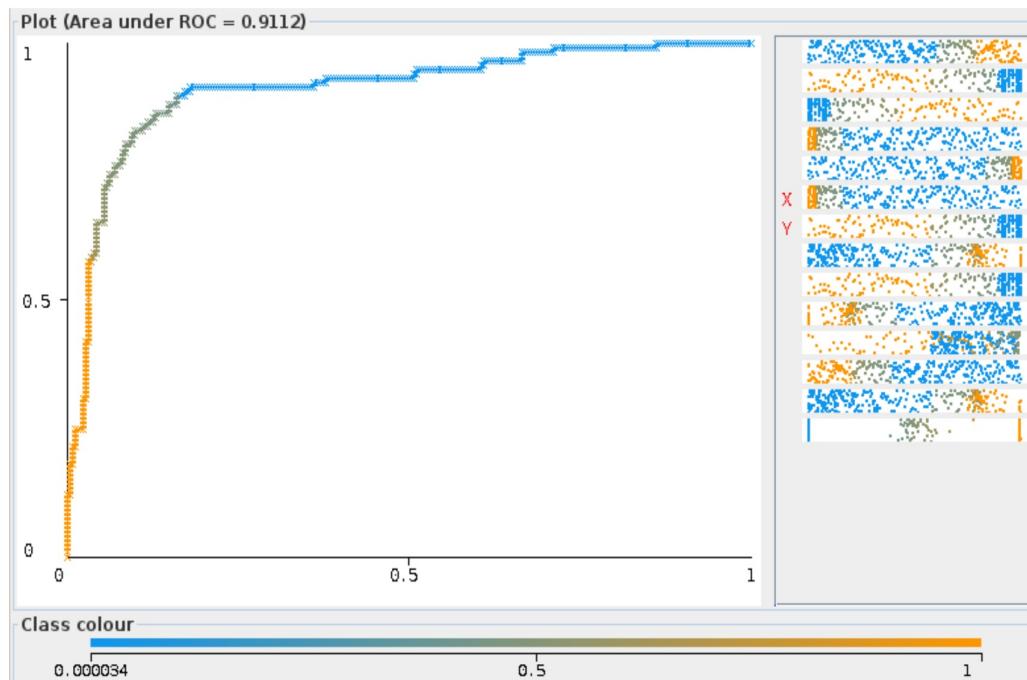


Figura 2.6: Curva ROC clase ghost

mejor se ha clasificado, que en este caso es ghost. A simple vista se pude apreciar que el AUC es bastante mayor en comparación con las otras dos clases. Por tanto podemos concluir que a mayor valor obtenido en el AUC clasificará mejor esa determinada clase.

2.2. Haga el mismo ejercicio, pero con un valor de vecinos k=3 y haga una interpretación de los resultados en comparación con k=2.

Si comparamos los resultados ofrecidos por Weka en la figura 2.7, podemos ver que haciendo uso de k igual a 3 en el algoritmo de kNN obtenemos unos mejores resultados de clasificación generales.

Sin embargo, si hacemos un análisis de cada clase, podemos ver lo siguiente:

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

Correctly Classified Instances      244           65.7682 %
Incorrectly Classified Instances   127           34.2318 %
Kappa statistic                      0.4858
Mean absolute error                  0.2566
Root mean squared error              0.4012
Relative absolute error              57.7733 %
Root relative squared error         85.1511 %
Total Number of Instances            371

== Detailed Accuracy By Class ==

          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC     ROC Area  PRC Area  Class
          0,659    0,186    0,654     0,659    0,656     0,472    0,828    0,725    Ghoul
          0,528    0,260    0,508     0,528    0,518     0,265    0,695    0,532    Goblin
          0,795    0,071    0,838     0,795    0,816     0,735    0,931    0,864    Ghost
Weighted Avg.    0,658    0,175    0,663     0,658    0,660     0,485    0,816    0,704

== Confusion Matrix ==

  a  b  c  <- classified as
85 42  2 |  a = Ghoul
43 66 16 |  b = Goblin
  2 22 93 |  c = Ghost
```

Figura 2.7: Resultados Knn con k = 3

- En todas las clases ha aumentado el TP (True Positive) rate, el AUC y el FP (False Positive) rate.
- La precisión se mantiene constante (Es muy similar)
- Si miramos la matriz de confusión y la precisión podemos ver que la clasificación de ghoul es peor.

Como es posible visualizar en los resultados, el AUC de la clase ghost es el mayor, por lo que si lo visualizamos veremos la curva que se muestra en la figura 2.8.

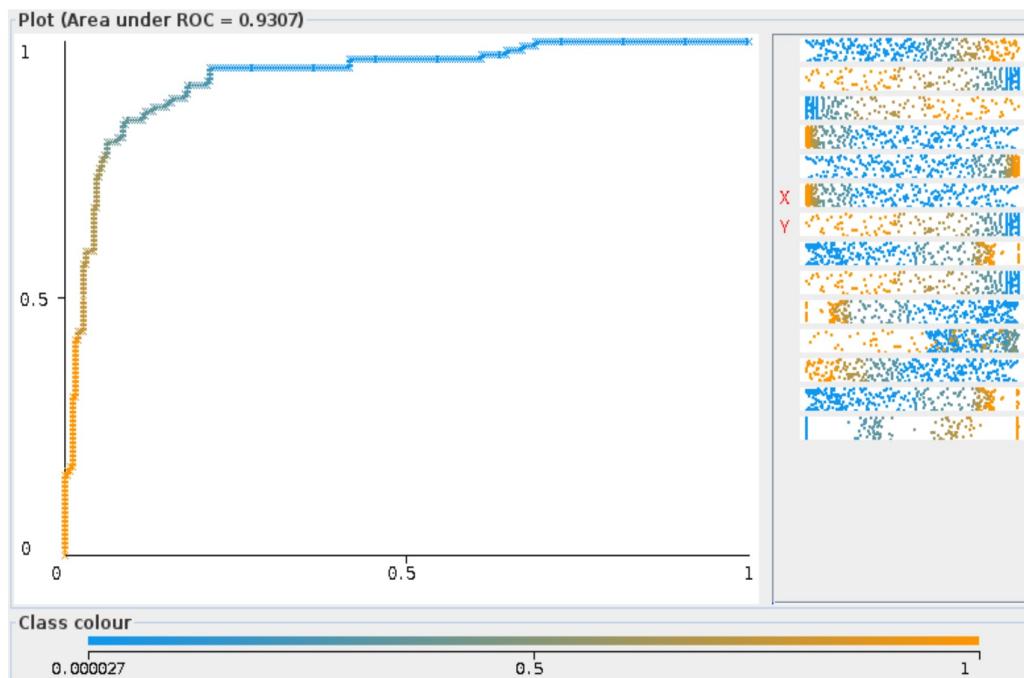


Figura 2.8: Curva ROC de la clase ghost

2.3. Con su base de datos, utilice el algoritmo de clasificación IBK con un 3-fold crossvalidation. Use un valor de vecinos k igual a valor con el que haya obtenido los mejores resultados.

2.3.1. Calcule la media y la desviación típica de las medidas Accuracy, Kappa, RMSE, F-Measure.

La medida Kappa_statistic es una métrica que compara la precisión observada con una precisión esperada. Esta estadística nos permite evaluar clasificadores entre ellos aparte de evaluar un único modelo.

El RMSE es una métrica de puntuación cuadrática que mide el promedio del error. Cuanto mayor sean estos errores, mayor será la puntuación que consigamos con esta métrica. En nuestro caso el valor que vemos en la tabla 2.1 es un valor intermedio. La métrica F es conjunto de métricas combinadas (precisión y recall). Esto es útil ya que nos ofrece una mejor forma de comparación de clasificadores.

Los resultados obtenidos en la tabla son las medias y desviaciones típicas de realizar un 3-fold, los cuales se pueden ver en la tabla 2.1.

	Kappa_statistic	Root_mean_squared_error	IR_precision	F_measure
Media	0,485604485	0,401141508	0,654500375	0,65611578
SD	0,021563949	0,009638934	0,011927989	0,009906977

Cuadro 2.1: Resultados medios de las diferentes métricas

A continuación se exponen varias gráficas de los resultados obtenidos. Realmente estos resultados se pueden ver de igual manera, ya que Weka los proporciona en forma de tabla. Pero también es cierto que mediante una serie de gráficas quedan mejor representados.

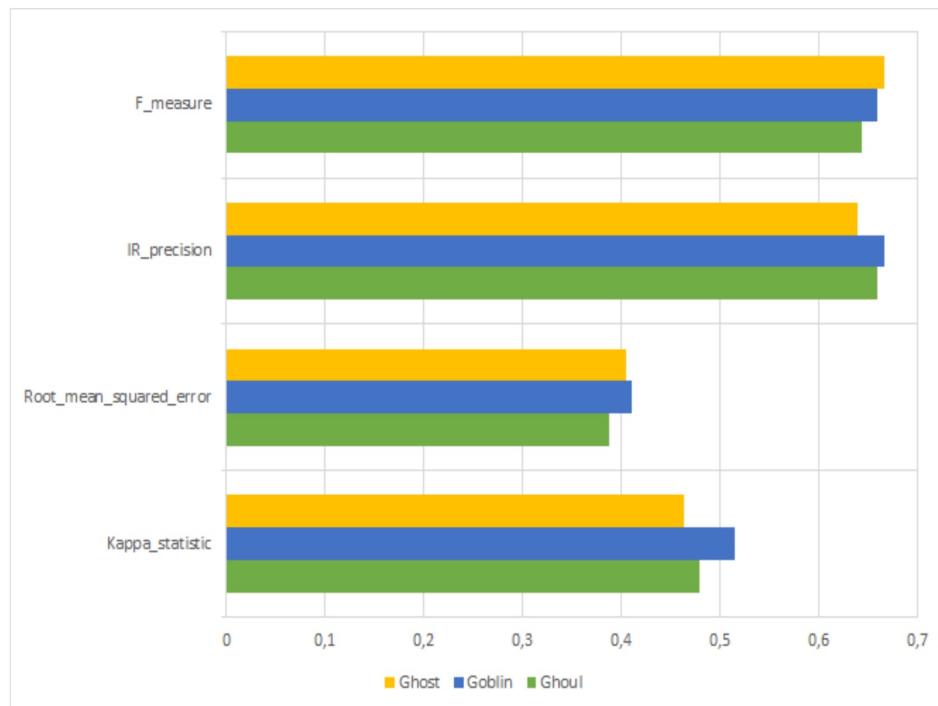


Figura 2.9: Representación de las medidas por clase

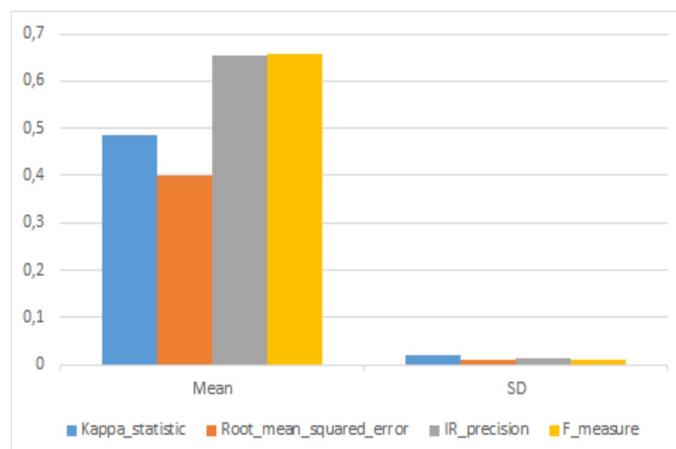


Figura 2.10: Representación gráfica de la tabla 2.1: media y sd

2.3.2. ¿Se corresponde el (TP Rate por clase) en el fichero csv con los valores que observó en el ejercicio usando el EXPLORER?

No, no se corresponden. El algoritmo tiene componentes aleatorios porque, por cada fold, cada vez que ha sido lanzado tiene un resultado distinto y por media podemos ver que tiene un valor distinto a una repetición por fold.

2.3.3. ¿Qué diferencia habría si hiciera el mismo ejercicio indicando que el número de repeticiones por fold fuese igual a 3? ¿Qué es lo que cambiará en cada repetición?

Básicamente a mayor número de repeticiones, nos aproximaremos mejor a un óptimo global, debido que al tener mayor cantidad de valores, al hacer la media es posible aproximarse a dicho óptimo. Por el contrario al tener un número de repeticiones igual a 1 o muy bajo, al ser un algoritmo con componente aleatoria, es posible que dicho valor estocástico caiga en un valle o en un mejor caso en un óptimo local, alejándonos de obtener el óptimo global del problema.

2.4. Usando el entorno EXPLORER ejecute el algoritmo Logistic con su base de datos, use un 3-fold crossvalidation como ya se hizo anteriormente.

2.4.1. Analice los modelos obtenidos, métricas, las variables que podrían ser más influyentes (valores beta), variables que no se usan, etc.

El modelo obtenido ha clasificado de forma correcta el 75% de los patrones. Los coeficientes de regresión logística se muestran a continuación: Las variables más

Logistic Regression with ridge parameter of 1.0E-8		
Coefficients...		
Variable	Class	
	Ghoul	Goblin
=====	=====	=====
bone_length	16.2742	10.4018
rotting_flesh	-11.278	-12.4957
hair_length	21.5638	14.9056
has_soul	18.3537	11.9192
color=clear	-0.8005	-0.5326
color=green	0.3077	0.3996
color=black	0.4026	0.0741
color=white	0.7838	0.4833
color=blue	0.6337	1.3704
color=blood	-3.4715	-3.5145
Intercept	-20.0206	-9.2431

Figura 2.11: Resultados obtenidos en la regresión logística

influyentes en el modelo son aquellas que su valor β en valor absoluto es alto. Por ello bone_length, hair_length y has_soul son las variables más influyentes, sin embargo, aquellas más próximas a cero no son influyentes, en este caso el color.

2.4.2. Asocie las fórmulas de las salidas por clase aportadas en las transparencias de esta práctica (transparencia titulada “Salidas por clase en Simplelogistic y Logistic de Weka”) con los modelos de probabilidad obtenidos en la salida de Weka.

La probabilidad para cada clase j a excepción de la ultima es:

$$P_j(X_i) = \frac{\exp(X_i B_j)}{\left(\sum_{k=1}^{j-1} \exp(X_i * B_k)\right) + 1} \quad (2.1)$$

Donde

- X_i : Corresponde a la variable θ (bone_length, rotting_flesh...)
- B_j : Corresponde a la variable β que son los coeficientes de regresión logística de las variables del modelo

2.5. Haga lo mismo usando el algoritmo Simple-Logistic y compare resultados

La ventaja de SimpleLogistic es que tiene una selección de atributos incorporada, pero si ejecutamos SimpleLogistic con un número suficientemente grande de iteraciones, obtendremos un predictor similar que usando regresión Logística.

```

Correctly Classified Instances      268          72.2372 %
Incorrectly Classified Instances   103          27.7628 %
Kappa statistic                   0.5833
Mean absolute error               0.2441
Root mean squared error           0.3461
Relative absolute error            54.9582 %
Root relative squared error       73.4456 %
Total Number of Instances         371

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

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC     ROC Area  PRC Area  Class
      0,760    0,153    0,726     0,760    0,742     0,601   0,900    0,818     Ghoul
      0,560    0,179    0,614     0,560    0,586     0,390   0,799    0,612     Goblin
      0,855    0,087    0,820     0,855    0,837     0,760   0,970    0,941     Ghost
Weighted Avg.                     0,722    0,141    0,718     0,722    0,719     0,580   0,888    0,787

==== Confusion Matrix ====

      a   b   c  <- classified as
  98  28  3 |  a = Ghoul
  36  70  19 |  b = Goblin
    1  16 100 |  c = Ghost

```

Figura 2.12: Resultados obtenidos con el algoritmo SimpleLogistic

Si queremos ajustar un modelo de regresión logística con el algoritmo Logistic, será más eficiente que SimpleLogistic, esto es debido a que Logistic está basado en todos los atributos del modelo, es decir el predictor obtenido será mejor a diferencia del algoritmo SimpleLogistic que realiza una construcción de atributos a media que realiza la ejecución del algoritmo.

```
SimpleLogistic:  
  
Class Ghoul :  
-7.08 +  
[bone_length] * 4.71 +  
[rotting_flesh] * -1.78 +  
[hair_length] * 5.93 +  
[has_soul] * 4.98  
  
Class Goblin :  
2.4 +  
[rotting_flesh] * -4.54 +  
[hair_length] * 0.43 +  
[color=white] * -0.09 +  
[color=blue] * 0.29 +  
[color=blood] * -0.49  
  
Class Ghost :  
7.21 +  
[bone_length] * -5.82 +  
[rotting_flesh] * 2.33 +  
[hair_length] * -7.68 +  
[has_soul] * -5.92
```

Figura 2.13: Resultados obtenidos con el algoritmo SimpleLogistic

El uso de cada uno varía dependiendo del problema, es decir, con otro problema el resultado de ambos será completamente distinto, por lo que sería necesario realizar la ejecución de ambos.

```
Correctly Classified Instances      276          74.3935 %
Incorrectly Classified Instances   95           25.6065 %
Kappa statistic                   0.6156
Mean absolute error               0.2203
Root mean squared error           0.3443
Relative absolute error            49.597 %
Root relative squared error       73.0688 %
Total Number of Instances         371

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

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC     ROC Area  PRC Area  Class
      0,744    0,132    0,750     0,744   0,747    0,613   0,903    0,819    Ghoul
      0,624    0,191    0,624     0,624   0,624    0,433   0,804    0,609    Goblin
      0,872    0,063    0,864     0,872   0,868    0,807   0,971    0,939    Ghost
Weighted Avg.                     0,744    0,130    0,744     0,744   0,744    0,614   0,891    0,786

==== Confusion Matrix ====

      a     b     c  <- classified as
96  32  1 |  a = Ghoul
32  78  15 |  b = Goblin
 0  15 102 |  c = Ghost
```

Figura 2.14: Resultados obtenidos con el algoritmo Logistic

2.6. 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.

2.6.1. Pruebe con 2 y 3 clusters analizando y discutiendo los clusters usados.

Mediante el algoritmo K-means minimizamos la suma de errores al cuadrado (SSE), por tanto, la solución final tiene más probabilidad de ser un mínimo local, debido que no tenemos la certeza de que devuelva el mínimo global ya que la inicialización de los centroides es de forma estocástica.

El SSE para un clúster dado, es calculado de forma que, para cada patrón en el clúster, suma las diferencias cuadráticas entre cada valor de atributo y el correspondiente centroide del clúster.

El SSE obtenido es el medido en el conjunto de datos de entrenamiento por lo que esto no quiere decir que generalice de forma correcta en clasificaciones futuras.

Se han obtenido los siguientes resultados: Cómo podemos observar en la figura an-

```

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:
          Cluster#
Attribute   Full Data      0        1
              (150.0)  (100.0)  (50.0)
=====
sepallength    5.8433    6.262    5.006
sepalwidth     3.054     2.872    3.418
petallength    3.7587    4.906    1.464
petalwidth     1.1987    1.676    0.244

```

Figura 2.15: Resultados obtenidos con 2 clusters

terior, respecto al procedimiento del algoritmo:

- En primer lugar, se asignan aleatoriamente los centroides de cada clúster, esta asignación variará según el valor de la semilla introducida, si esta no varía siempre cogerá los mismos centroides.
- Tras las distintas iteraciones, los centroides de cada clúster y para cada atributo van variando respecto a la distancia euclídea entre el valor del atributo del patrón respecto al centroide, es decir cada asignación de un patrón a un clúster hace que se recalcule el centroide del mismo para continuar siendo el centro del cluster.
- Una vez finalizada, podemos observar los centroides finales de cada clúster, donde será la media ponderada para los valores numéricos y un recuento de frecuencias respecto a los valores nominales.

```

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:
          Cluster#
Attribute   Full Data      0        1        2
          (150.0)  (61.0)  (50.0)  (39.0)
=====
sepallength    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

```

Figura 2.16: Resultados obtenidos con 3 clusters

2.6.2. Para cada valor del anterior, visualice los clusters resultantes al representar los atributos petallength y petalwidth y coméntelos.

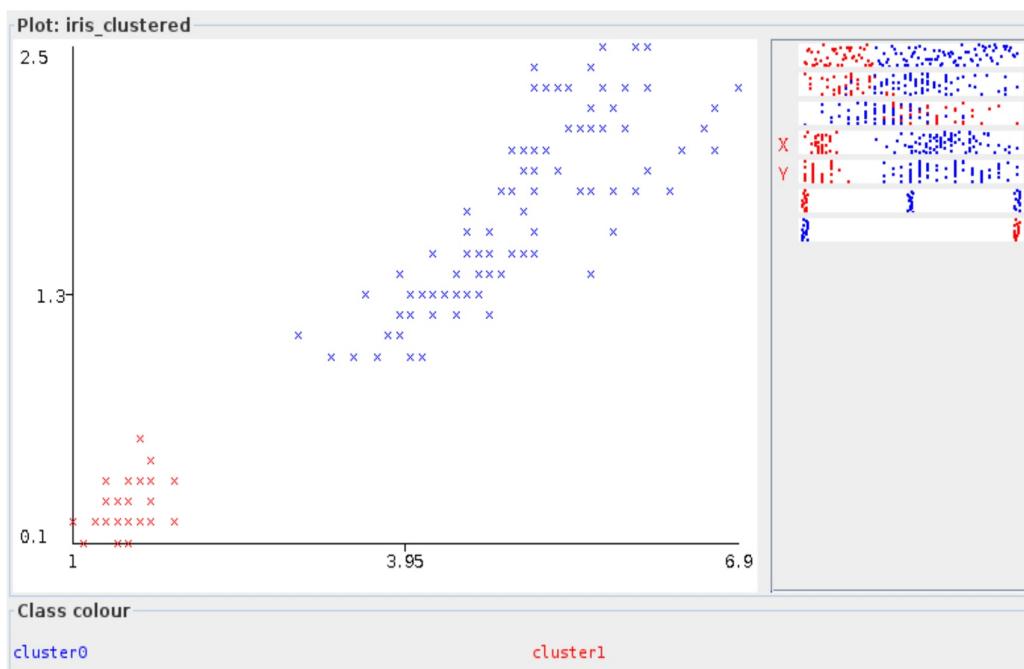


Figura 2.17: Representación de instancias obtenidos con 2 clusters

Como es visible en las imágenes de visualización de Weka, cuando lo realizamos con 2 clústeres clasifica como el mismo clúster las clases Iris-Versicolor e Iris-Virgínica que son aquellas que podemos ver de color azul en la figura 2.17.

Sin embargo, cuando realizamos la ejecución del algoritmo con 3 clústeres es capaz de diferenciar entre estas dos clases que no ha sido capaz de diferenciar previamente ya que se le indicaron únicamente 2 clústeres.

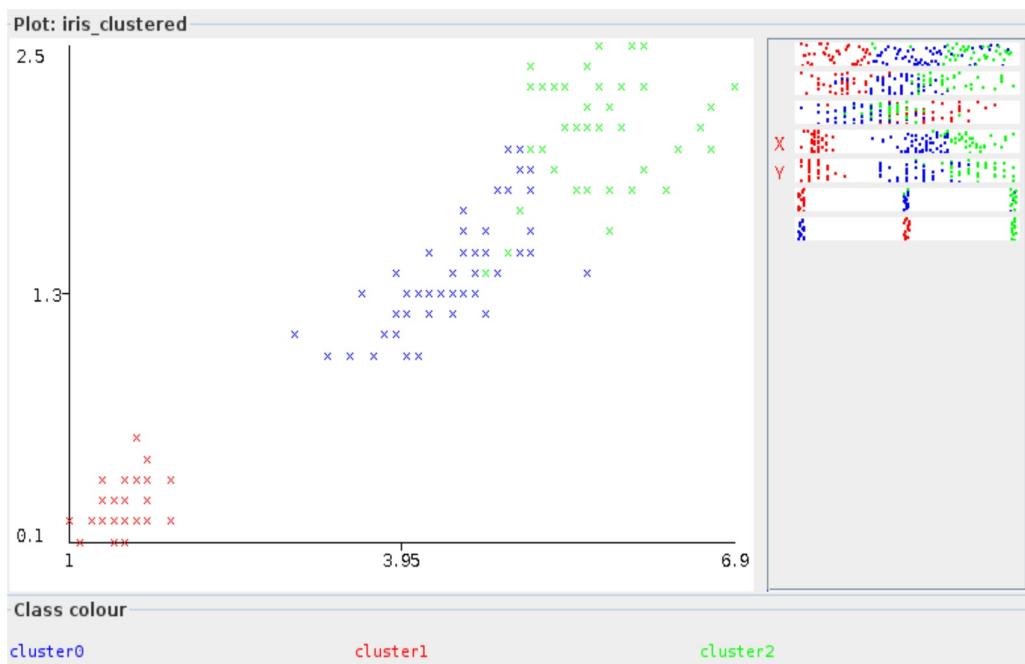


Figura 2.18: Representación de instancias obtenidos con 3 clusters

2.7. Cargue su base de datos de criaturas tenebrosas, seleccione la opción “Use training set” y analice qué ocurre al aplicar K-means, fijando k=número-de-clases de la base de datos.

Cuando añadimos la etiqueta de clase se obtienen mejores clasificaciones, esto sucede ya que la etiqueta de clase indica a qué clase pertenece el patrón, por ello el algoritmo obtiene mejores resultados y un SSE más elevado.

Considerando la etiqueta de clase.					Ignorando la etiqueta de clase.				
Attribute	Full Data (371.0)	Cluster# 0 (137.0)	1 (120.0)	2 (114.0)	Attribute	Full Data (371.0)	Cluster# 0 (137.0)	1 (60.0)	2 (174.0)
bone_length	0.4342	0.429	0.4716	0.4009	bone_length	0.4342	0.429	0.4585	0.4298
rotting_flesh	0.5068	0.512	0.5311	0.4751	rotting_flesh	0.5068	0.512	0.5419	0.4907
hair_length	0.5291	0.5196	0.5582	0.5099	hair_length	0.5291	0.5196	0.4999	0.5467
has_soul	0.4714	0.4651	0.514	0.4341	has_soul	0.4714	0.4651	0.4793	0.4736
color=clear	0.3235	0	0.35	0.6842	color=clear	0.3235	0	0	0.6897
color=green	0.1132	0	0.2333	0.1228	color=green	0.1132	0	0	0.2414
color=black	0.1105	0	0.2333	0.114	color=black	0.1105	0	0.6833	0
color=white	0.3693	1	0	0	color:white	0.3693	1	0	0
color=blue	0.0512	0	0.1	0.0614	color:blue	0.0512	0	0.3167	0
color=blood	0.0323	0	0.0833	0.0175	color=blood	0.0323	0	0	0.069
type	Ghoul	Ghoul	Ghoul	Goblin					
Time taken to build model (full training data) : 0.06 seconds									
*** Model and evaluation on training set ***									
Clustered Instances									
0	137	(37%)			0	137	(37%)		
1	120	(32%)			1	60	(16%)		
2	114	(31%)			2	174	(47%)		

Figura 2.19: Comparativa clusters respecto a la etiqueta de clase

2.8. ¿Qué ocurriría en K-means si fijásemos el número de clúster igual al número de patrones de una base de datos?

Si fijásemos el número de clúster igual al número de patrones de una base de datos, se crearía finalmente un clúster por cada patrón.

Como la iniciación de los centroides en el algoritmo K-means es de forma aleatoria, dependiendo de donde se inicien estos el algoritmo será capaz de encontrar unos clústeres u otros. Precisamente por esto el algoritmo puede ofrecernos unos resultados muy buenos algunas veces y otros bastantes malos otras tantas.

2.9. Utilizando su base de datos criaturas tenebrosas, el algoritmo K-means, y seleccionando la opción Classes to clusters evaluation, ¿con qué número de clusters obtiene mejores resultados?.

- Para ello realice por cada valor de k 3 pruebas (cada una con una semilla diferente), y obtenga la media de la métrica SSE.
- Use la métrica de distancia que quiera, pero indique cuál ha usado. Se ha usado como métrica la distancia euclídea.
- Represeñe en una gráfica el número de clusters frente al valor medio del SSE, y determine cuál es el valor de k más idóneo para su base de datos.



Figura 2.20: Número de clusters vs Media SSE

2.10. Ejecute el algoritmo HierarchicalClusterer con tipo de link Complete y métrica de distancia euclídea. ¿Cuáles de ellos producen los grupos mejor diferenciados y con fronteras claras?

En la figura 2.5 se puede apreciar que la visualización de instance_number respecto al petalwidth produce el grupo mejor diferenciado.

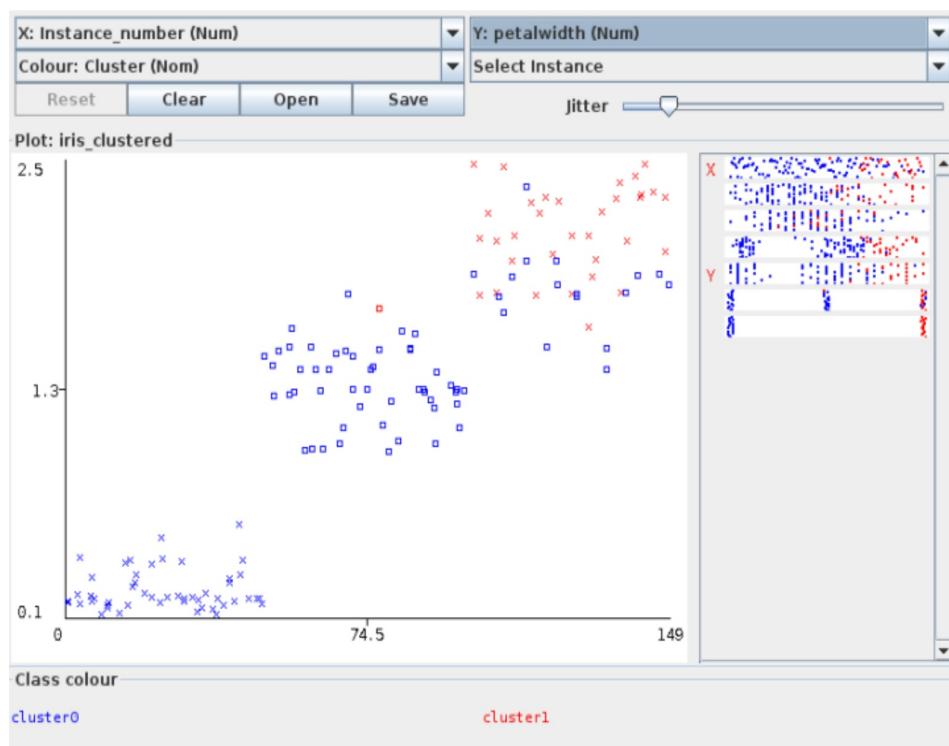


Figura 2.21: Comparativa instance_number frente petalwith

2.11. Ejecute el algoritmo HierarchicalClusterer sobre su base de datos criaturas tenebrosas

2.11.1. ¿Con qué linkType obtiene los mejores resultados en cuanto a instancias mal agrupadas? Pruebe con Single y Complete.

Simple	Complete
Clustered Instances	Clustered Instances
0 340 (92%)	0 132 (36%)
1 19 (5%)	1 102 (27%)
2 12 (3%)	2 137 (37%)
Class attribute: type	Class attribute: type
Classes to Clusters:	Classes to Clusters:
0 1 2 <- assigned to cluster	0 1 2 <- assigned to cluster
119 6 4 Ghoul	46 33 50 Ghoul
116 7 2 Goblin	48 34 43 Goblin
105 6 6 Ghost	38 35 44 Ghost
Cluster 0 <- Ghoul	Cluster 0 <- Goblin
Cluster 1 <- Goblin	Cluster 1 <- Ghost
Cluster 2 <- Ghost	Cluster 2 <- Ghoul
Incorrectly clustered instances : 239.0 64.4205 %	Incorrectly clustered instances : 238.0 64.1509 %

Figura 2.22: Comparativa de resultados Single y Complete linkTypes

Como es posible observar cuando realizamos la agrupación con linkType simple, obtenemos un peor resultado, sin embargo este apenas se diferencia de cuando lo hemos realizado con linkType completo.

2.11.2. Analice los clusters creados con cada linkType y sus asignaciones.

En la agrupación jerárquica con linkType simple, la distancia entre dos clústeres se define como la distancia más corta entre dos puntos en cada clúster.

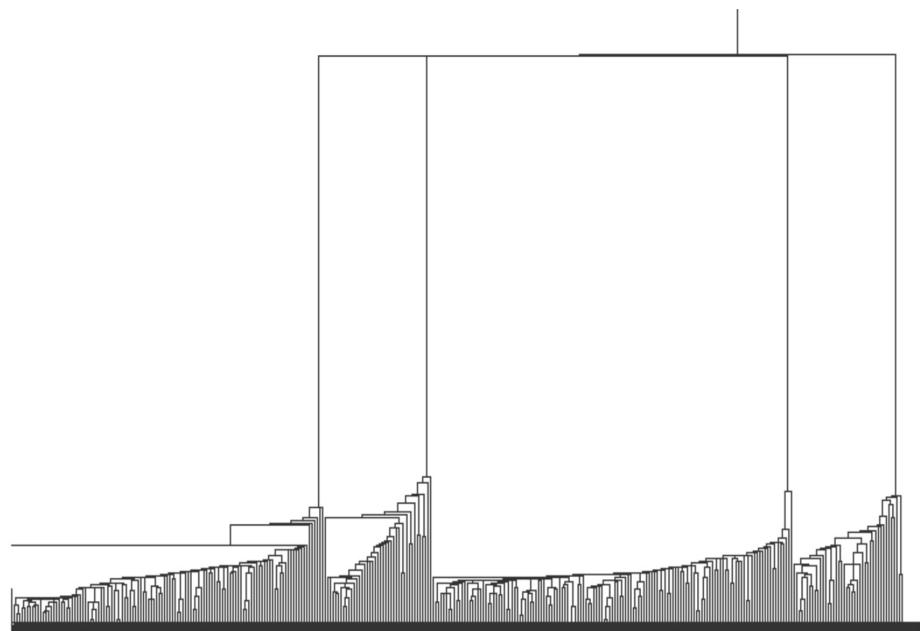


Figura 2.23: Comparativa Single y Complete linkTypes: Single

En la agrupación jerárquica con linkType complete, la distancia entre dos clústeres se define como la distancia más larga entre dos puntos en cada clúster.

Si se cortara el dendrograma más arriba, entonces habría menos clústeres finales, pero su nivel de similitud sería menor.

Si se cortara el dendrograma más abajo, entonces el nivel de similitud sería mayor, pero habría más clústeres finales.

Como nosotros hemos establecido que corte con tres clústeres es donde el realizará la división, quedándose al final con esos 3 clústeres.

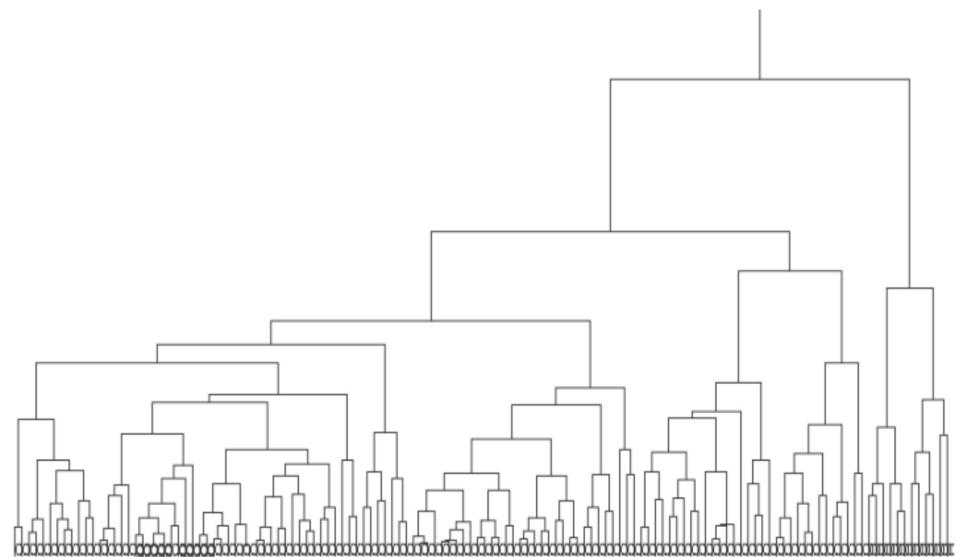


Figura 2.24: Comparativa Single y Complete linkTypes: Complete linkType

Práctica 3 - Árboles y Redes Neuronales

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

3.1.1. Analice y muestre el árbol obtenido con los parámetros por defecto: nodo principal, número de nodos u hojas, variables presentes y omitidas.

Este algoritmo escoge atributos usando el porcentaje de ganancia para maximizarla. Como nodo principal se ha seleccionado como atributo “hair_length” ya que es aquel que mayor información aporta al modelo, seguido del atributo “has_soul”. El árbol generado por el algoritmo consta de 53 nodos en total, de los cuales 27 corresponden a nodos terminales u hoja. Respecto a los datos obtenidos mediante el algoritmo J48, nos ha generado un árbol con las siguientes características:

- Número de nodos hoja: 27

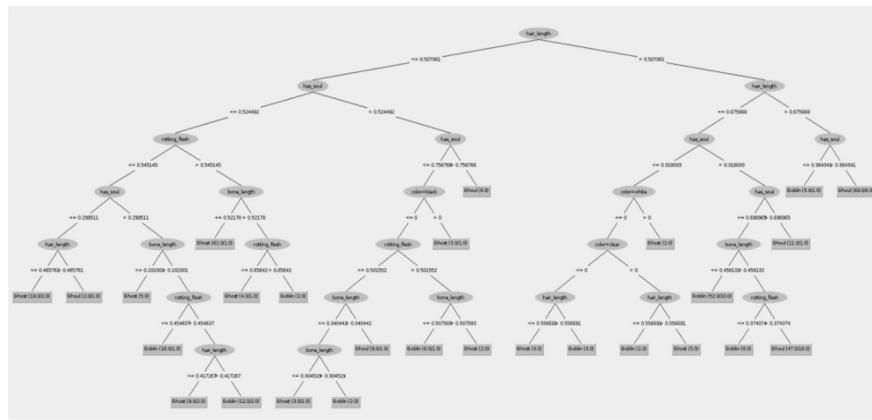


Figura 3.1: Árbol obtenido con el algoritmo C4.5

- Tamaño del árbol: 53

Siendo la clasificación de las clases observable en la matriz de confusión de un total de 93 instancias empleadas al usar el 75 % de train y 25 % de test. Cómo podemos

```

    --- Confusion Matrix ---
    a   b   c   <-- classified as
23   6   1 |   a = Ghoul
    7 23   4 |   b = Goblin
    0   9 20 |   c = Ghost

```

Figura 3.2: Matriz de confusión obtenida con el algoritmo C4.5

apreciar, la clasificación de las clases no ha sido mala, debido que ha clasificado casi todos los patrones correctamente salvo aproximadamente el 10% de cada instancia que ha clasificado erróneamente. Weka nos ofrece información resumida respecto a los resultados de clasificación del conjunto de entrenamiento.

==== Summary ===

Correctly Classified Instances	66	70.9677 %
Incorrectly Classified Instances	27	29.0323 %
Kappa statistic	0.5619	
Mean absolute error	0.231	
Root mean squared error	0.4316	
Relative absolute error	51.9605 %	
Root relative squared error	91.4991 %	
Total Number of Instances	93	

Figura 3.3: Información resumida de los resultados

Podemos ver que ha clasificado de forma correcta 66 patrones que conforma el 70,96 % y 27 de forma incorrecta que conforma el 29,04 % de un total de 93 patrones. Un ejemplo de correcta clasificación sería el siguiente:

Como podemos ver si seguimos las reglas que existen en el árbol llegamos al final

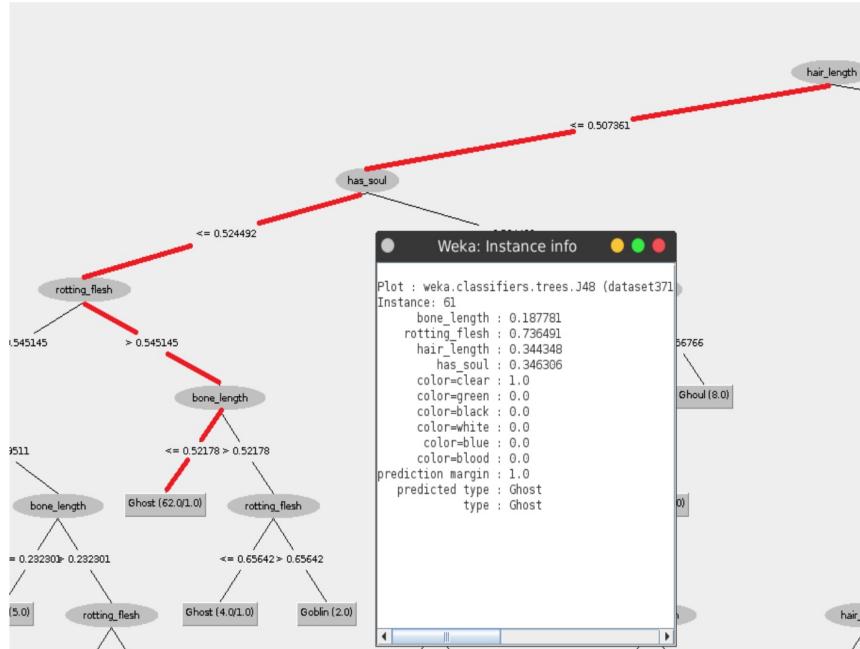


Figura 3.4: Ejemplo de correcta clasificación

a una correcta clasificación del patrón, puede haber casos en los que la secuencia de las reglas no nos lleve al patrón correcto, esto en parte puede ser debido a que

dicho patrón no ha sido correctamente clasificado, debido que el árbol generado está basado en el conjunto de train y no de test.

3.1.2. Analice y muestre cómo cambia el árbol, al modificar los siguientes parámetros. Comente también los resultados de las métricas obtenidas

- Unpruned: La activación o desactivación de este parámetro permitirá realizar poda a nuestro árbol.

De tal forma que algunos datos se han mantenido constantes como puede ser

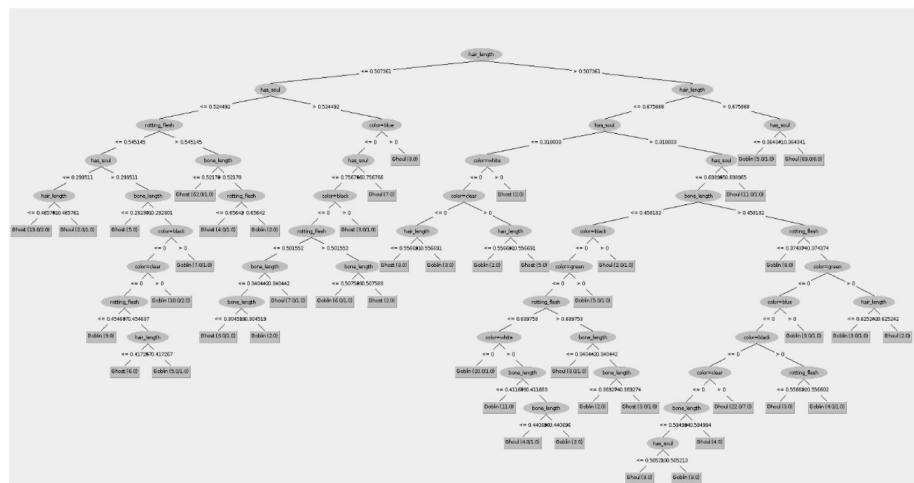


Figura 3.5: Árbol completo con poda

el nodo raíz y sus hijos más cercanos, sin embargo, los siguientes datos se han visto modificados:

- Instancias correctamente clasificadas: Ha descendido a un 65 %.
 - Instancias incorrectamente clasificadas: Ha aumentado al 34 %.
 - Número de nodos hoja: Ha aumentado a 46
 - Tamaño del árbol: Ha aumentado a 91

- ConfidenceFactor: El factor de confianza utilizado para la poda (los valores más pequeños incurren en más poda), por lo que un valor pequeño corresponde a la poda pesada, una poda grande a pequeña, por lo que será al disminuir el factor de confianza se disminuirá la cantidad de post poda. Cómo podemos

ConfidenceFactor	Número Nodos Hoja	Tamaño del árbol	CCR	Unpruned
0,01	20	39	0,69	False
0,25	27	53	0,7	False
0,45	38	75	0,67	False
0,51	41	81	0,66	False
0,75	41	81	0,66	False
0,9	41	81	0,66	False
0,01	46	91	0,65	True
0,25	46	91	0,65	True
0,45	46	91	0,65	True
0,51	46	91	0,65	True
0,75	46	91	0,68	True
0,9	46	91	0,68	True

Figura 3.6: Tabla con la comparativa de resultados

observar en la tabla, cuanto mayor es el factor de confianza, mayor será el tamaño del árbol resultante, así como también disminuye su CCR final, por lo que podemos determinar que no para al menos este problema una poda muy excesiva no brinda buenos resultados.

- MinNumObj: Representa el número mínimo de instancias por nodos hoja, las instancias mínimas por hoja garantizan que, en cada división, al menos 2 de las divisiones (pero no necesariamente más de 2) tendrán el número mínimo de instancias. Por lo que, el número mínimo de instancias por hoja se considera mejor como 'la cantidad mínima de separación de datos por rama', en el caso de árboles de múltiples vías.

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

3.2.1. ¿Cuál sería el valor por defecto para el atributo hidden Layers en su base de datos?

El atributo `hiddenLayers` nos indica el número de capas ocultas que tendrá nuestro perceptrón, de tal forma que por defecto vendrá determinado por “`a`”, siendo ésta el $(\text{número de atributos} + \text{clases}) / 2$.

En el ejercicio de esta práctica, estamos haciendo uso de la base de datos criaturas tenebrosas, por lo que tendríamos un total de 10 atributos y 3 clases, tal que el número de capas ocultas serán: $(10 + 3) / 2 = 6$

3.2.2. Con los valores por defecto, ¿qué observa al ir modificando solo el `learningRate`?

El valor de `learningRate` nos está indicando La cantidad de pesos que se actualizan por cada iteración, pudiendo tomar valores comprendidos entre 0 y 1.

Tal como podemos apreciar en la figura 3.7, cuando el LearningRate supera 0.6, comienza la red neuronal a aprender demasiado rápido, por lo que se comienza a sobre-entrenar debido que los “saltos” que va haciendo en el espacio de búsqueda son mucho mayores, obteniendo peores resultados. Al igual ocurrirá cuando el umbral

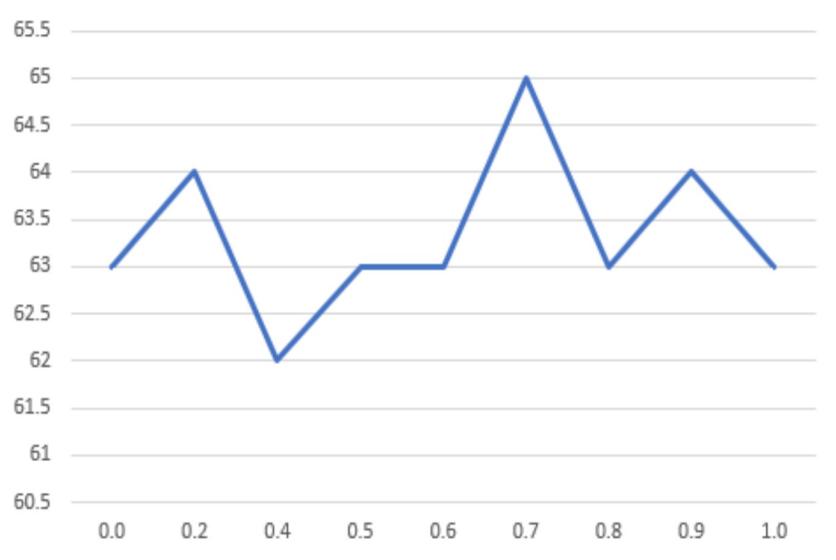


Figura 3.7: Gráfica con distintos valores de LearningRate (eje x) y sus resultados (eje y)

es inferior a 0.3, debido que estamos realizando un entrenamiento demasiado lento, por lo que estaríamos incurriendo de nuevo en sobre-entrenamiento.

3.2.3. Con los valores por defecto, ¿qué observa al ir modificando solo el momentum?

Podemos definir el momentum como la intensidad en la modificación de los enlaces, siendo este un valor entre [0,1].

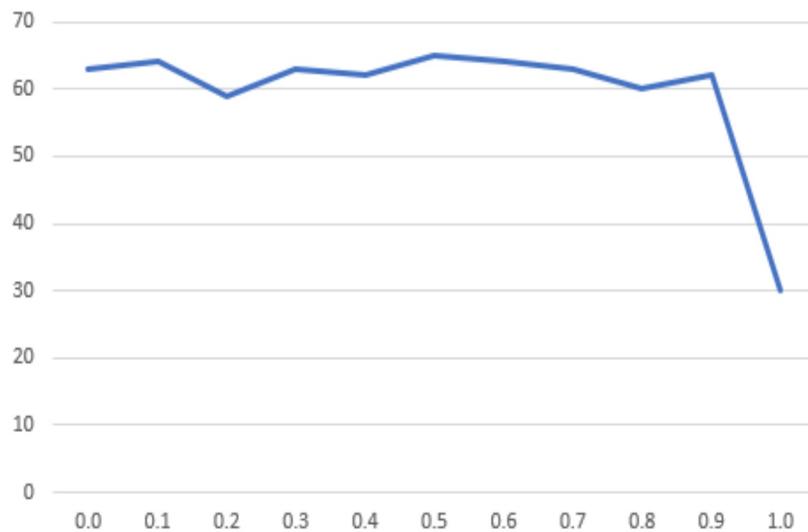


Figura 3.8: Gráfica con distintos valores de Momentum (eje x) y sus resultados (eje y)

En la gráfica de la figura 3.8 podemos apreciar como a partir de que el momentum alcanza 0.6 o superior, el ratio de instancias correctamente clasificadas comienza a descender bruscamente, como conclusión podemos aportar que un gran valor de momentum implica que la convergencia será más rápida.

Pero si tanto el momentum como la velocidad de aprendizaje se mantienen en valores altos, entonces puede omitir el mínimo con un gran paso. Un pequeño valor de momentum no puede evitar de manera confiable los mínimos locales, y también puede ralentizar el entrenamiento del sistema.

Por lo que el momentum también nos ayuda a suavizar estas variaciones, si el gradiante sigue cambiando de dirección. Un valor correcto de éste puede aprenderse en prueba y error, tal y como hemos realizado en la práctica.

3.2.4. Con los valores por defecto, ¿qué observa al ir modificando solo el training-Time? Realice una gráfica que muestre cómo cambia el valor de CorrectlyClassified instances al modificar el parámetro trainingTime, de forma que localice con cuántas épocas se estanca el aprendizaje o incluso se empieza a sobreentrenar.

Al ir modificando el training Time desde 50 hasta 400 iteraciones, podemos observar en la figura 3.9 como a partir de las 175 épocas comienza a sobre-entrenar nuestro modelo, debido que la tasa de CCR/TT comienza a decrecer, esto es posible ajustarlo para un correcto entrenamiento mediante ensayo y error.

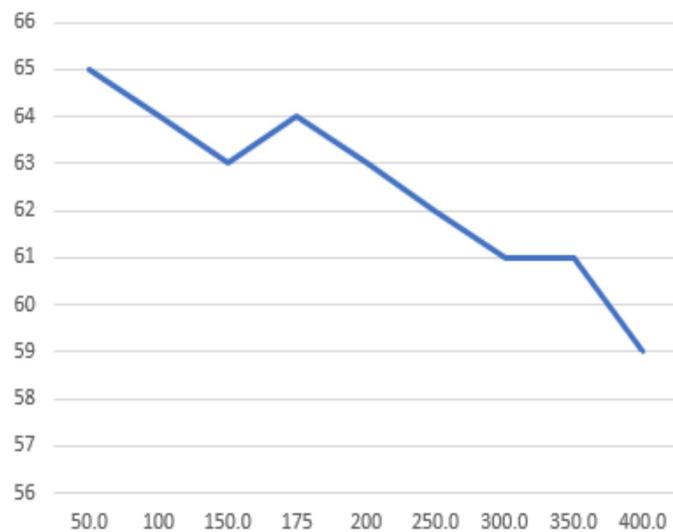


Figura 3.9: Gráfica con distintos valores de Iteraciones (eje x) y sus resultados (eje y)

Práctica 4 - Preprocesamiento

En el problema planteado para realizar la competición en Kaggle, hemos tomado la tarea de preprocesado de los datos como la de mayor importancia a la hora de tratar de mejorar siempre el resultado obtenido. Por lo que los pasos seguidos a la hora de realizarlo han seguido la siguiente sucesión de eventos, diferenciados en train y test.

4.1. Preprocesamiento en conjunto de train

4.1.1. Tratamiento de atributos

La primera tarea para realizar ha sido eliminar aquellos atributos que tuviesen más del 40 % de datos perdidos, la explicación a esta tarea es que al existir una gran cantidad de datos perdidos por atributo y posteriormente reemplazar dichos valores por la media/moda, esos valores obtenidos no son reales, sino que son una aproximación. De ahí, que se ha optado por eliminar dichos atributos, antes de insertar una gran cantidad de valores similares a nuestra base de datos, siendo el filtro empleado para esta tarea “RemoveUseless”.

El principal problema que se nos plantea al eliminar atributos es que podríamos perder información muy valiosa para nuestro problema, sin embargo, al tener una gran cantidad de datos perdidos, se ha preferido eliminar dicho atributo. Una vez eliminados los atributos con mayor cantidad de datos perdidos, se ha procedido a reemplazar dichos valores restantes por la media del atributo, haciendo uso de “ReplaceMissingValues”.

El procedimiento relacionado a reemplazar valores perdidos en train y test difiere un poco, por lo que se explicará en la sección de preprocesamiento del test.

4.1.2. Correlación de atributos

Actualmente nuestro problema dispone de muchos atributos, por lo que puede ser buena idea reducir el conjunto de datos hasta únicamente quedarnos con aquella información que es relevante para nuestro problema. El uso de métodos de selección de características en un proceso de aprendizaje aporta ventajas importantes entre los que destacan:

- Una mejora considerable en la eficacia del algoritmo de aprendizaje, debido que el coste computacional y de tiempo, crece de manera exponencial en determinados algoritmos, por lo que una reducción del espacio aumentará el rendimiento de estos.
- Mejoras en la precisión de los resultados obtenidos. Si una característica es irrelevante entonces podemos aumentar la calidad del modelo si nuestro algoritmo parte de un conjunto relevante de atributos.

Ante los dos posibles enfoques vistos en clase para selección de características, como son los métodos de filtrado y los métodos wrappers, nos hemos decantado por el segundo.

Esta selección viene determinada a que tienen generalmente mejor desempeño que los de filtrado, debido que, al mejorar el algoritmo de clasificación, estaremos mejorando a su vez el proceso de selección de atributos. Dado que el principal problema de los wrappers su alto coste computacional para un elevado número de atributos, hemos empleado este método ya que tenemos un problema con pocos atributos, por lo que el coste no será muy alto.

Mediante la selección de características, vemos los atributos que deseamos eliminar, que en nuestro caso han sido: air, WSPD y ATMP, haciendo uso de la matriz de correlación, así como la cantidad de información que aporta el atributo a la clase.

Cómo podemos ver en la matriz 4.1, los atributos WSPD y GST, están muy correlados, por lo que habría que decidir eliminar uno de ellos.

	air	pres	rhum	uwnd	vwnd	WDIR	WSPD	GST	DPD	APD	PRES2	ATMP	WTMP
air	1	0.23	0.26	0.12	0	0.1	-0.15	-0.18	-0.34	-0.44	0.26	0.98	0.94
pres	0.23	1	-0.03	0.14	-0.2	0.25	-0.34	-0.37	-0.26	-0.43	0.93	0.26	0.26
rhum	0.26	-0.03	1	-0.12	0.34	-0.12	0.05	0.03	-0.18	-0.17	-0.02	0.26	0.11
uwnd	0.12	0.14	-0.12	1	-0.17	0.68	-0.07	-0.07	0	-0.03	0.05	0.17	0.26
vwnd	0	-0.2	0.34	-0.17	1	-0.15	0.31	0.31	0.07	0.13	-0.18	-0.01	-0.12
WDIR	0.1	0.25	-0.12	0.68	-0.15	1	-0.06	-0.07	-0.04	-0.09	0.17	0.14	0.2
WSPD	-0.15	-0.34	0.05	-0.07	0.31	-0.06	1	0.99	0.01	0.1	-0.31	-0.17	-0.21
GST	-0.18	-0.37	0.03	-0.07	0.31	-0.07	0.99	1	0.04	0.14	-0.34	-0.2	-0.23
DPD	-0.34	-0.26	-0.18	0	0.07	-0.04	0.01	0.04	1	0.71	-0.27	-0.33	-0.31
APD	-0.44	-0.43	-0.17	-0.03	0.13	-0.09	0.1	0.14	0.71	1	-0.44	-0.44	-0.44
PRES2	0.26	0.93	-0.02	0.05	-0.18	0.17	-0.31	-0.34	-0.27	-0.44	1	0.28	0.29
ATMP	0.98	0.26	0.26	0.17	-0.01	0.14	-0.17	-0.2	-0.33	-0.44	0.28	1	0.95
WTMP	0.94	0.26	0.11	0.26	-0.12	0.2	-0.21	-0.23	-0.31	-0.44	0.29	0.95	1

Figura 4.1: Matriz de correlación de los atributos inicial

El motivo por el cual se decide a eliminar atributos correlados es muy importante de cara a posteriormente utilizar un modelo u otro, por ejemplo, uno de los usados en nuestra práctica ha sido el perceptrón multicapa. En dicho modelo, al tener dos atributos similares, hará que la importancia que le preste el modelo a dicho atributo sea mayor que el resto, debido que es similar a tener el mismo atributo dos veces. Técnicamente tras visualizar la figura 4.2, deberíamos quitar el que menos correlación tiene respecto la clase ya que es el que menos información nos aportará, sin embargo, hay casos en los que dicho atributo puede estar más correlado con otros atributos, es decir muy correlado con la clase, por lo que deberíamos mantenerlo, pero al mirar el resto de los atributos, también existe una fuerte correlación.

Por lo que al estar tan correlado con la clase y el resto de los atributos, es preferible eliminarlo, para mantener el que menos correlación tiene ya que nos aportará mayor cantidad de información, por tanto, finalizamos borrando el atributo WSPD.

```

Attribute Evaluator (supervised, Class
Correlation Ranking Filter
Ranked attributes:
0.6791 8 GST
0.6534 7 WSPD
0.6106 10 APD
0.3701 9 DPD
0.3236 5 vwind
-0.055 3 rhum
-0.0856 4 uwnd
-0.1369 6 WDIR
-0.2881 14 DEWP
-0.4055 1 air
-0.426 12 ATMP
-0.4522 13 WTMP
-0.5241 11 PRES
-0.5595 2 pres

```

Figura 4.2: Correlación de atributos con la clase

Continuando el mismo procedimiento, volveremos a ejecutar los diferentes filtros: “*PrincipalComponents*”, “*CorrelationAttributeEval*” y “*ClassifierSubEval*”, para obtener nuevamente las distintas correlaciones.

	air	pres	rhum	uwnd	vwind	WDIR	GST	DPD	APD	PRES2	ATMP	WTMP
air	1	0.23	0.26	0.12	0	0.1	-0.18	-0.34	-0.44	0.26	0.98	0.94
pres	0.23	1	-0.03	0.14	-0.2	0.25	-0.37	-0.26	-0.43	0.93	0.26	0.26
rhum	0.26	-0.03	1	-0.12	0.34	-0.12	0.03	-0.18	-0.17	-0.02	0.26	0.11
uwnd	0.12	0.14	-0.12	1	-0.17	0.68	-0.07	0	-0.03	0.05	0.17	0.26
vwind	0	-0.2	0.34	-0.17	1	-0.15	0.31	0.07	0.13	-0.18	-0.01	-0.12
WDIR	0.1	0.25	-0.12	0.68	-0.15	1	-0.07	-0.04	-0.09	0.17	0.14	0.2
GST	-0.18	-0.37	0.03	-0.07	0.31	-0.07	1	0.04	0.14	-0.34	-0.2	-0.23
DPD	-0.34	-0.26	-0.18	0	0.07	-0.04	0.04	1	0.71	-0.27	-0.33	-0.31
APD	-0.44	-0.43	-0.17	-0.03	0.13	-0.09	0.14	0.71	1	-0.44	-0.44	-0.44
PRES2	0.26	0.93	-0.02	0.05	-0.18	0.17	-0.34	-0.27	-0.44	1	0.28	0.29
ATMP	0.98	0.26	0.26	0.17	-0.01	0.14	-0.2	-0.33	-0.44	0.28	1	0.95
WTMP	0.94	0.26	0.11	0.26	-0.12	0.2	-0.23	-0.31	-0.44	0.29	0.95	1

Figura 4.3: Matriz de correlación de los atributos inicial tras eliminar WSPD

Cómo se comentó previamente, buscaremos las distintas correlaciones, determinando por nuestra parte que el siguiente atributo a eliminar será ATMP, debido que el atributo AIR, tiene menos correlación con la clase, así como el resto de los atributos respecto ATMP. El siguiente atributo con una correlación de 0.94, es WTMP

	air	pres	rhum	uwnd	vwnd	WDIR	GST	DPD	APD	PRES2	WTMP
air	1	0.23	0.26	0.12	0	0.1	-0.18	-0.34	-0.44	0.26	0.94
pres	0.23	1	-0.03	0.14	-0.2	0.25	-0.37	-0.26	-0.43	0.93	0.26
rhum	0.26	-0.03	1	-0.12	0.34	-0.12	0.03	-0.18	-0.17	-0.02	0.11
uwnd	0.12	0.14	-0.12	1	-0.17	0.68	-0.07	0	-0.03	0.05	0.26
vwnd	0	-0.2	0.34	-0.17	1	-0.15	0.31	0.07	0.13	-0.18	-0.12
WDIR	0.1	0.25	-0.12	0.68	-0.15	1	-0.07	-0.04	-0.09	0.17	0.2
GST	-0.18	-0.37	0.03	-0.07	0.31	-0.07	1	0.04	0.14	-0.34	-0.23
DPD	-0.34	-0.26	-0.18	0	0.07	-0.04	0.04	1	0.71	-0.27	-0.31
APD	-0.44	-0.43	-0.17	-0.03	0.13	-0.09	0.14	0.71	1	-0.44	-0.44
PRES2	0.26	0.93	-0.02	0.05	-0.18	0.17	-0.34	-0.27	-0.44	1	0.29
WTMP	0.94	0.26	0.11	0.26	-0.12	0.2	-0.23	-0.31	-0.44	0.29	1

Figura 4.4: Matriz de correlación de los atributos inicial tras eliminar ATMP

con AIR, por lo cual determinando cual de ambos se ha de eliminar, se llega a la conclusión que es mejor borrar el atributo WTMP. Al cabo de las diferentes iteraciones, debido que es posible que las correlaciones puedan verse modificadas al eliminar uno u otro atributo, se ha establecido el umbral de eliminar aquella correlación superior al 0.94, finalizando con la eliminación de los siguientes atributos:

- ATMP
- WTMP
- DPD
- PRES2

4.1.3. Detección y tratamiento de outliers

La detección de datos anormales ha sido realizada mediante el uso de un filtro no supervisado a nivel de atributo. Este añadirá a cada patrón dos atributos adicionales indicando si se trata de un outlier o no, mediante el filtro “InterquartileRange”.

En nuestro problema no hemos detectado ningún dato anómalo tras la realización de los pasos mencionados, por lo que no hemos realizado tratamiento alguno de este tipo de datos.

4.1.4. Normalización

El proceso de normalización ha sido realizado mediante el filtro “Normalize”, sin embargo, antes de realizar el proceso de normalización, se ha de normalizar el conjunto de test. Una vez se ha normalizado el conjunto de test mediante la obtención del máximo y mínimo de cada atributo, podremos realizar la normalización con el filtro indicado.

4.2. Preprocesamiento en conjunto de test

El proceso de preprocesado de los datos del conjunto de test ha sido prácticamente similar al del conjunto de train, variando únicamente el proceso de normalización de atributos y reemplazo de valores perdidos.

4.2.1. Tratamiento de atributos

Al igual que en el conjunto de train, se eliminarán los atributos que tengan más del 40 % de datos perdidos, en nuestro caso se ha eliminado el atributo DEWP, por lo que habrá que eliminar dicho atributo también en el conjunto de test. Todo atributo que hayamos borrado en el conjunto de test también deberá ser borrado en el conjunto de train. Se ha de añadir un nuevo atributo denominado Class_WVHT, que tendrá todos sus valores vacíos, de modo que nuestro objetivo será predecirlos.

Una vez finalizado el proceso de eliminar atributos, hemos de llenar todos los datos que se encuentren perdidos, esto será realizado mediante la obtención de la media/moda del atributo en el conjunto de train, por lo que todo dato perdido en el test será cambiado por la media o moda del train, usando el filtro “*ReplaceMissingWithUserConstant*”.

4.2.2. Normalización

El proceso de normalización de atributos difiere un poco respecto al procedimiento seguido en el conjunto de train. Para su correcta normalización debemos hacer uso de los valores máximo y mínimo del train, normalizando de esta forma cada atributo de forma individual mediante el uso del proceso de normalizado.

$$\text{Normalización}(u) = \frac{V(u) - V_{min}}{V_{max} - V_{min}}$$

Como nota, hay que indicar que, durante este proceso, hemos podido observar que existen datos que se encuentran o bien por debajo del valor 0 o superior al 1.

4.2.3. Construcción del modelo

A la hora de seleccionar nuestro modelo o conjunto de modelos, nos hemos basado en usar aquellos que mejores resultados nos han brindado para nuestro problema. Entre los cuales podemos encontrar el MultilayerPerceptron, SimpleLogistic y RandomTree.

Por lo que, como objetivo final, tras haber estudiado las posibles configuraciones, así como tratamiento de datos para cada clasificador por individual, nos hemos decantado por hacer uso de un ensemble, haciendo uso de clasificador de combinación “vote”, cuya configuración final ha sido la que se muestra a continuación.

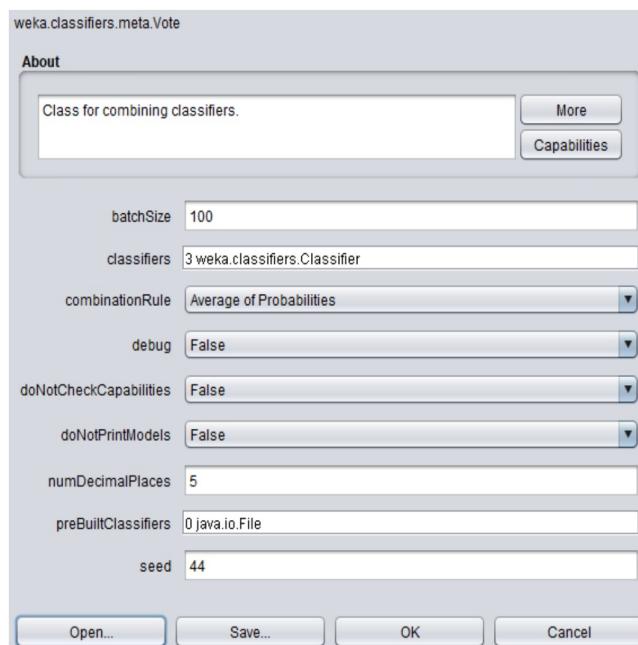


Figura 4.5: Parámetros del ensemble *vote* de clasificadores

Hemos seleccionado los tres clasificadores mencionados previamente, así como optado por utilizar la media de probabilidades como regla para combinarlos. Los

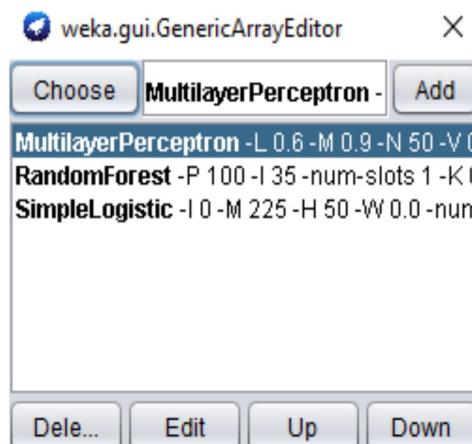


Figura 4.6: Clasificadores usados en el ensemble

clasificadores empleados, así como su respectiva configuración han sido realizados con el único fin de evitar el sobreentrenamiento, tratando de maximizar la capacidad de generalización de nuestro modelo.

MultilayerPerceptron

La configuración empleada para el MultilayerPerceptron, ha sido la siguiente, en la cual destacaremos los siguientes puntos:

decay	True
doNotCheckCapabilities	False
hiddenLayers	a
learningRate	0.6
momentum	0.9
nominalToBinaryFilter	True
normalizeAttributes	True
normalizeNumericClass	True
numDecimalPlaces	5
reset	True
seed	100
trainingTime	50
validationSetSize	0
validationThreshold	20

Figura 4.7: Parámetros del clasificador *MultilayerPerceptron*

- Decay: Al establecer un decay a True, nos estamos asegurando que el ratio de aprendizaje sea reducido, pudiendo ayudar a evitar que la red se aparte de la salida de destino, así como a mejorar el rendimiento general.

- LearningRate: Se ha establecido a un valor de 0.6, de tal forma que su aprendizaje sea algo más rápido.
- Momentum: Es el peso aplicado a las conexiones, establecido a 0.9 mediante ensayo y error.
- HiddenLayers: Será el número de capas ocultas, por defecto hemos observado que no hay diferencias significativas respecto a cambiarlo.
- NormalizeAttributes: El activar esta opción, no hará falta que normalicemos los atributos tal y como mencioné previamente, sino que será el propio clasificador el que realizará la normalización internamente.
- numDecimalPlaces: Se ha establecido que la precisión para los números decimales sea de 5.
- TrainingTime: Para evitar un sobre-entrenamiento se ha establecido que este valor a 50 iteraciones, obtenido mediante ensayo y error.

RandomForest

Continuando con el siguiente clasificador usado, la configuración establecida ha sido la siguiente:

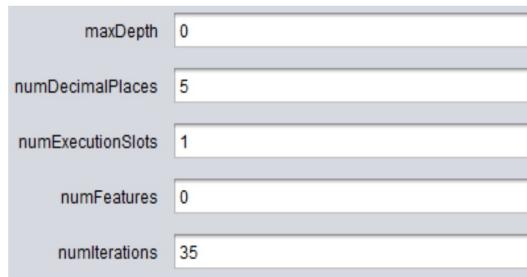


Figura 4.8: Parámetros del clasificador *RandomForest*

- MaxDepth: profundidad máxima del árbol, el cual se ha optado por dejarla ilimitada.
- NumDecimaLplaces: Número de decimales, al igual que el anterior.
- NumIterations: Tras realizar varias pruebas, se ha determinado que con 35 iteraciones obtenemos unos resultados bastante buenos en nuestro modelo.

SimpleLogistic

Por último, se incluirá el clasificador SimpleLogistic, en el cual hemos usado prácticamente la configuración por defecto, debido que no encontrábamos grandes cambios respecto a nuevas configuraciones.

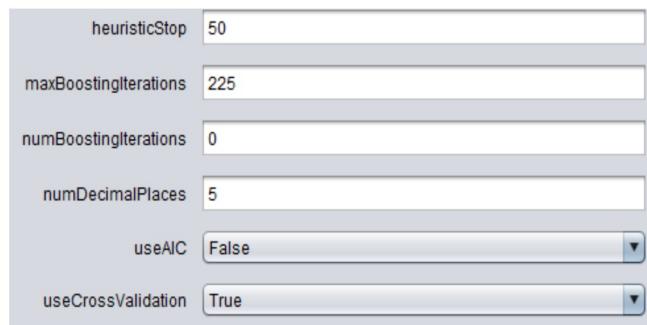


Figura 4.9: Parámetros del clasificador *SimpleLogistic*

- useCrossValidation: Se ha establecido a True, el cual determinará el número de iteraciones para la validación cruzada.
- heuristicStop: Estableciendo un valor superior a 0, estaremos haciendo uso de una heuristica greedy o voráz.

Bibliografía

- [1] Jason Brownlee. *Machine Learning Mastery*, 2019.
- [2] Stack Overflow. *Stack Overflow - Where Developers Learn, Share, & Build Careers*, 2019.
- [3] Weka. *Weka.sourceforge.net*, 2019.