Escuela Politécnica Superior de Jaén
Grado en Ingeniería Electrónica Industrial

**UNIVERSIDAD DE JAÉN**
*Escuela Politécnica Superior de Jaén*

Trabajo Fin de Grado

# IMPLEMENTACIÓN EN JAVA DE UNA HERRAMIENTA PARA LA CLASIFICACIÓN DE ACEITES SEGÚN SU CALIDAD

**Alumno: Mª Ángeles Pérez Carrasco**

Tutor: Prof. D. Diego Manuel Martínez Gila

Prof. Dña. Elisabet Estévez Estévez

Dpto: Ingeniería Electrónica y Automática

**Septiembre, 2019**


Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingenería Electrónica y Automática

Don Diego Manuel Martínez Gila , tutor del Trabajo Fin de Grado titulado: Implementación en Java de una Herramienta Para la Clasificación de Aceites Según Su Calidad, que presenta Mª Ángeles Pérez Carrasco, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2019

El alumno: Los tutores:

Mª ÁNGELES PÉREZ CARRASCO DIEGO M. MARTINEZ GILA

## RESUMEN

En el trabajo de fin de grado que aquí se expone, se ha abordado el problema de la clasificación del aceite según su calidad.

Para la consecución de éste objetivo, primero se han transformado los datos, provenientes de una nariz electrónica, para hacerlos aptos como entradas de los algoritmos clasificadores, que serán aquellos de entre todos los disponibles en Weka que tengan mejor desempeño.

Con objeto de saber ajustar adecuadamente los parámetros de los clasificadores, y de analizar en interpretar los resultados obtenidos mediante éstos, se va a realizar una investigación acerca del funcionamiento de los algoritmos clasificadores.

Finalmente, se expondrán tanto los resultados de los clasificadores, el resultado de la implementación de los mismos en Java.

## ABSTRACT

This project is about the classification of olive oil according to its quality.

To reach this objective, first of all, a data coming from an electronic nose will be transformed, according to make it available as input of the classifiers. The classifiers will be chosen over the ones offered by Weka.

After this, a research about how does the classifiers works will be done, in order to analyze the results obtained by them, and adjust its parameters properly.

Finally, the results of the classifiers will be exposed, and the tool developed in Java will be presented.

# Índice

# 1. INTRODUCCIÓN

En el presente trabajo de fin de grado (TFG), se ha realizado la implementación de una herramienta, programada en Java, que a partir de una serie de medidas realizadas con una nariz electrónica y mediante unos algoritmos de clasificación que se describirán en el punto 2, es capaz de distinguir si las muestras de aceite de oliva que se le proporcionan son de calidad virgen extra o no.

En éste documento se explicará el proceso seguido para el desarrollo de la herramienta de clasificación: la obtención de los datos de entrada de los algoritmos clasificadores, el proceso de selección de los algoritmos adecuados para la problemática planteada, una explicación de cómo trabajan éstos para asignar una clase a una muestra, y los resultados obtenidos por ellos.

## 1.1. Motivación

El presente proyecto, viene dado por la necesidad de aportar una solución mecanizada y libre de factores azarosos, que pueden introducir errores en el proceso de clasificación de la calidad en los aceites de oliva.

La distinción entre el aceite de oliva virgen y el aceite de oliva virgen extra, se realiza en la actualidad mediante un análisis sensorial llamado cata. En éste proceso, atendiendo a su color, sabor y aroma, se distingue entre aceite de oliva virgen y aceite de oliva virgen extra. Tomando en consideración su sabor y su olor, hay una serie de atributos que se considera positiva a la hora de catalogar el aceite como de la máxima calidad, y otros atributos que sirven para lo contrario. Los atributos gustativos y olfativos positivos son: frutado, amargo y picante; mientras que los negativos: quemado, madera, basto, lubricante, alpechín, salmuera, esparto, tierra, gusano y pepino. La cata la realiza un número de ocho a doce catadores, cada uno de los cuales tiene que cumplir con unas normas a fin de aislar posibles elementos que distorsionen su juicio. No obstante, pese a éstas normas, se puede apreciar que éste tipo de análisis aportan una excesiva subjetividad.

Es por esto, que es conveniente la disposición de una herramienta que sea capaz de realizar ésta clasificación sin los problemas inherentes a la metodología anteriormente expuesta.

## 1.2. Contexto y antecedentes

Éste trabajo de fin de grado se enmarca en el ámbito de la automatización industrial aplicada a la agricultura y se puede contextualizar dentro del proyecto que versa sobre el desarrollo de una nariz electrónica (E-NOSE) desarrollado por el grupo de investigación en Robótica, Automática y Visión por computador, y que tiene como objetivo el desarrollo de un dispositivo electrónico que determina la calidad del aceite de oliva a partir de sus componentes volátiles. Estos componentes volátiles son medidos por una serie de sensores de gas siendo la respuesta electrónica de estos sensores las entradas de la herramienta de clasificación desarrollada en éste TFG. Por tanto, éste proyecto tiene como punto de partida el resultado de los siguientes trabajos fin de grado:

- **Desarrollo de hardware de una nariz electrónica para la discriminación de aceites**, (Hernández Cledera, M.,2018). En éste proyecto se realizó el desarrollo de todas las placas de circuito impreso que conforman la nariz electrónica, que es la parte encargada de sensar las muestras de aceite.
- **Diseño mecánico de una nariz electrónica para la discriminación de aceites**, (Gómez García, J., 2018). En este proyecto se trabajó sobre el diseño físico de la nariz electrónica se ha realizado en éste proyecto. Esto abarca todos los mecanismos físicos que hacen posible el sensado, así como la caja que sirve de soporte a todo.
- **Desarrollo de Software de una nariz electrónica para la discriminación de aceites**, (López Riquelme, A., 2018). Con las partes de la circuitería y el soporte físico ya cubiertas, en éste proyecto se realizó el programa que controla todos los procesos de la nariz electrónica.

El resultado de todos éstos pasos es la nariz electrónica que se puede ver en la ilustración 1.

Las narices electrónicas son dispositivos que constan de una serie de sensores de gas, cada uno de los cuales mide unos compuestos en específico, mediante la variación de alguna variable física, como la conductividad. A éstos elementos de

sensado, por tanto, se les ha de proveer de un sistema de reconocimiento de patrones, para la distinción de los olores.

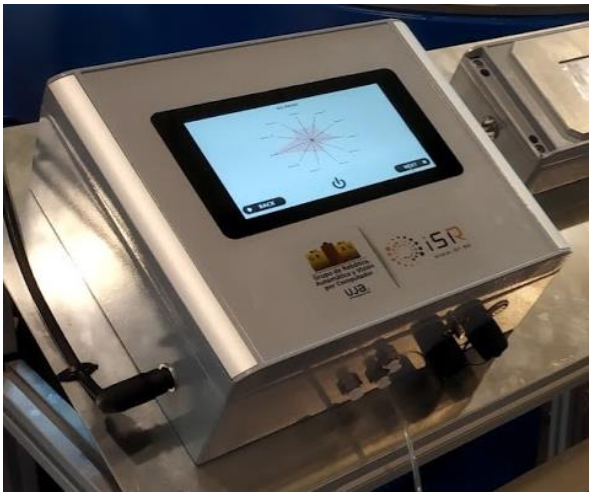


**Ilustración 1: E-NOSE**

Éste proyecto enlaza con todos los anteriores, y viene a proporcionar un software que sea capaz de procesar la respuesta sensorial de diferentes muestras de aceite de oliva, cuyos datos son recogidos por la E-NOSE, y clasificarlas según su calidad.

## 1.3. Objetivos

El objetivo de éste TFG es la implementación de las nuevas tecnologías aplicadas al control de calidad del aceite en el sector de la elaiotecnia. En concreto, mediante éste trabajo se pretende desarrollar una aplicación que, a partir de unos datos tomados de unas muestras de aceite, sea capaz de clasificar éstas según su calidad. Para la consecución de éste objetivo, se plantean los siguientes objetivos específicos:

- Realizar una aplicación para la extracción de datos, procedentes de un conjunto de sensores, y el cálculo de las características más relevantes de éstos. Éstas características son, para las medidas normalizadas de cada sensor, las siguientes: el máximo valor de la medida, el mínimo, la desviación típica, la media, área de la curva que encierran las medidas normalizadas, y las pendientes de subida y de bajada de dicha curva.

- Seleccionar los clasificadores del entorno de análisis del conocimiento WEKA que sean más relevantes para la clasificación de aceites según su calidad.
- Estudiar los fundamentos y el funcionamiento de los clasificadores seleccionados, ajustando los valores de los mismos para los cuales el error de clasificación es mínimo.
- Integrar en una aplicación de java los clasificadores seleccionados como adecuados para la problemática planteada.

## 1.4. Estado del Arte

Actualmente, existen diversos programas que permiten la aplicación de algoritmos de Machine Learning sobre los sets de datos que se pretende estudiar. El término Machine Learning, etiqueta a un campo de la ciencia de la computación que crea modelos mediante la identificación de patrones en los sets de datos de entrada que se les suministre. Así, mediante la modelización de la información de entrada, se podrá categorizar una nueva muestra nunca antes vista.

Los programas que permiten la aplicación de éstos algoritmos pueden ser softwares propietarios como RapidMiner Matlab o SPSS Modeler entre otros muchos. Pero también hay muchos softwares “open source” con un muy amplio uso por parte de la comunidad científica, como R o Weka.

A continuación, se procederá a describir éstos entornos para el Machine Learning, haciendo especial hincapié en Weka, entorno que se ha usado para el estudio de los algoritmos de clasificación que desarrolla la herramienta creada en éste TFG.

### 1.4.1. RapidMiner

Éste programa está disponible tanto en “open source”, como en una versión de pago del mismo, menos limitada. Está desarrollado en Java y dispone módulos de integración con Python y R. Éste software fue desarrollado en 2001 por la Universidad de Dormund, bajo el nombre de YALE (Yet Another Learning Environment), y es un programa muy usado en el campo del Machine Learning, debido a su facilidad de uso, dada por la interfaz gráfica que posee, y a su compatibilidad con R.

Éste programa permite la creación modelos predictivos muy variados a través de una programación visual de bloques, muy amigable y para la que no hace falta tener conocimientos profundos en el campo de la informática.

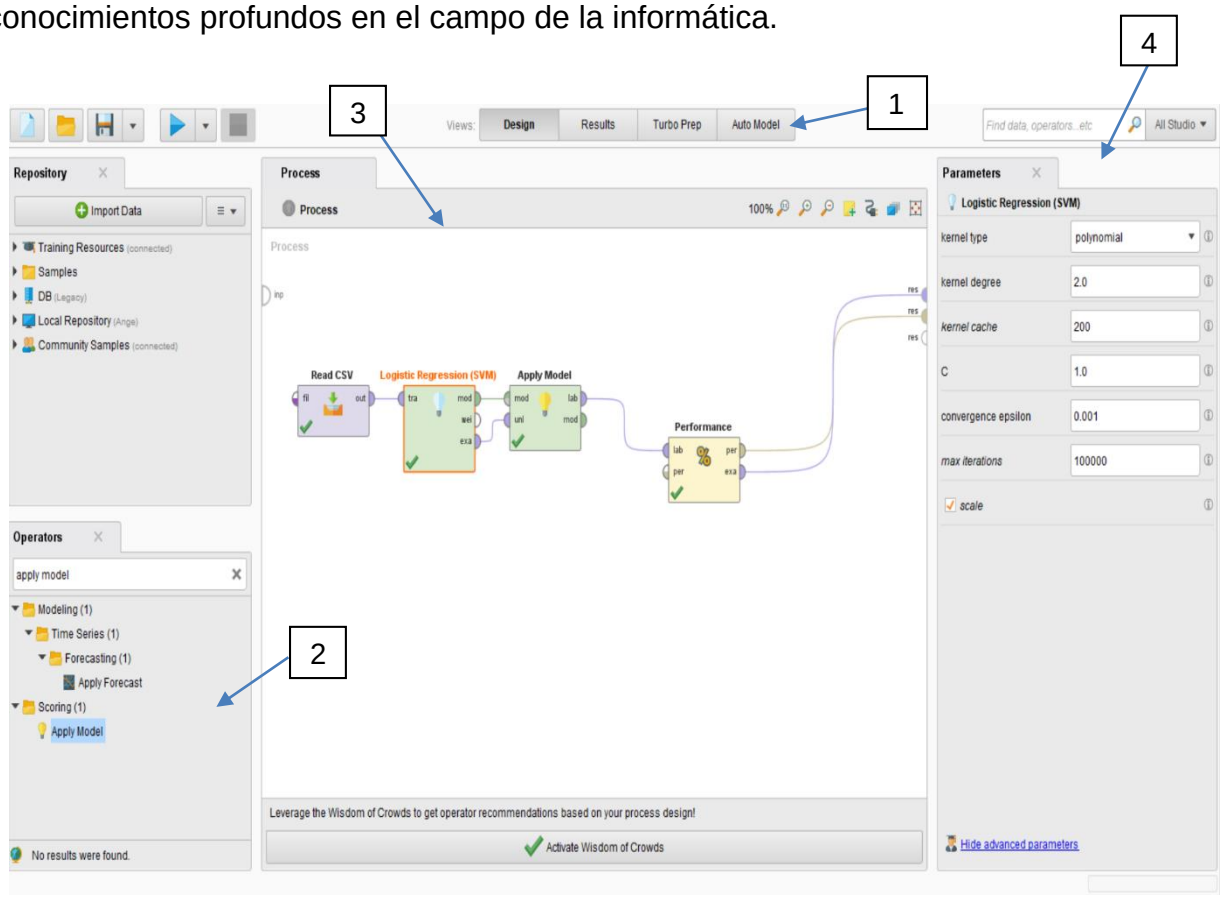



**Ilustración 2: Interfaz Diseño RapidMiner**

La ilustración 2 muestra la interfaz de RapidMiner relativa al diseño del programa. En ella, se distinguen cuatro partes debidamente etiquetadas:

- En la parte superior (1), se presenta el menú de pestañas de las que consta ésta interfaz gráfica: Design, Results, Turbo Prep y Auto Model. En éste documento se mostrarán las dos primeras mediante la creación de un programa sencillo.
- En la esquina inferior izquierda (2), se encuentran todos los bloques que podrán ser usados en la creación de un programa.
- La región central de la interfaz de diseño (3) está dedicada al área de trabajo disponible para la realización del programa. En ésta ilustración se puede ver un sencillo programa, a modo de ejemplo de uso de la herramienta. En él se lee un archivo CSV con la información de entrada del modelo, y se construye una regresión logística mediante máquinas de vector soporte. El último

bloque es el encargado de evaluar el rendimiento del modelo creado por el clasificador. Nótese que éste último bloque está enlazado a un nodo de nombre “res”, que marca lo que se podrá ver en la ventana results.

- Los bloques pueden ser configurados mediante menús que emergen en la parte derecha de la ventana (4), al hacer click izquierdo sobre ellos.

Una vez creado el programa que se ajusta a las necesidades del usuario, éste se ejecuta pudiéndose ver entonces una pantalla de resultados como la de la figura 3.

En ésta ilustración se puede ver la matriz de confusión de la clasificación, así

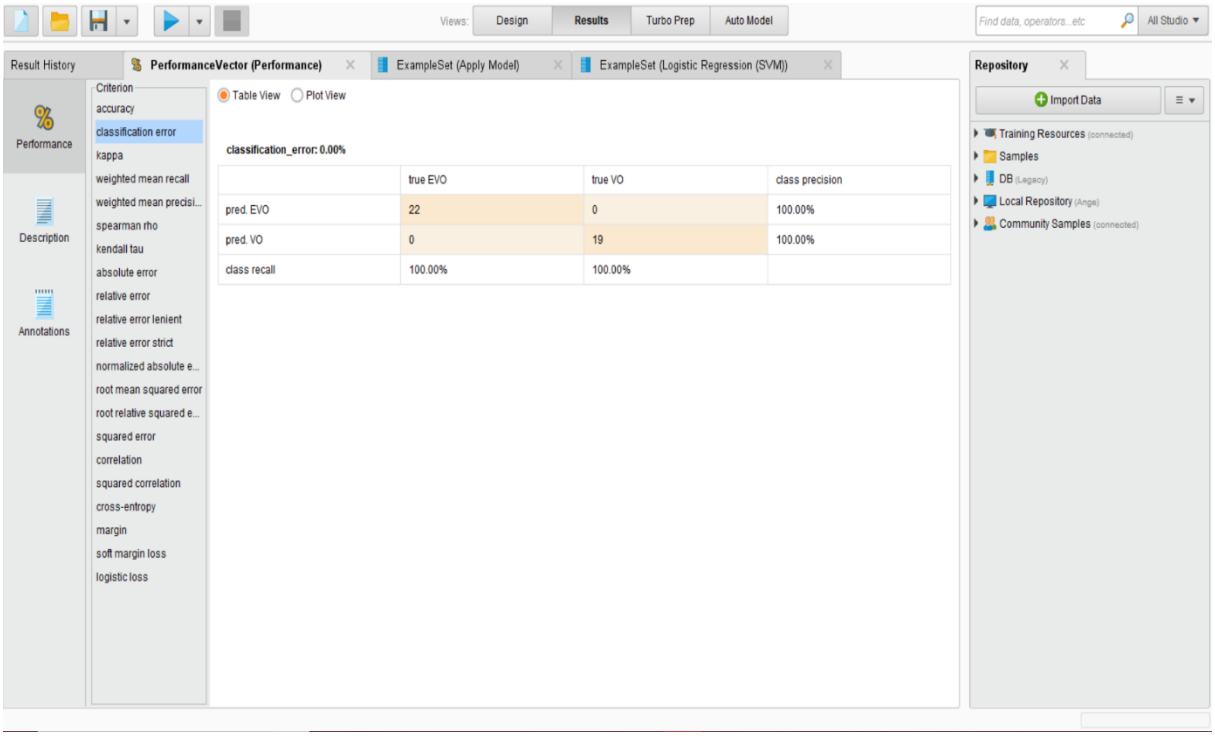



**Ilustración 3: Resultados Modelo**

como otros campos que se explicarán más adelante, como recall (sensibilidad). Nótese que ésta solo una pestaña de todas las que se pueden observar como resultado de la evaluación.

### 1.4.2. Matlab

Matlab (MATrix LABoratory) es un entorno de desarrollo integrado (IDE) que, al contrario que RapidMiner, es por completo un software propietario. Éste software tiene su propio lenguaje de programación de alto nivel (lenguaje M), siendo uno de los softwares más potentes y versátiles que se usan en el campo de la ingeniería y la investigación. Esto es posible gracias a una extensa oferta de toolboxes, o “cajas de

herramientas" que se usan en un campo concreto de aplicación: hay uno para la adquisición de imágenes, otro para la simulación de sistemas dinámicos (Simulink), etc.

En el campo de investigación del presente documento, hay varios toolboxes de interés, recogidos en la sección "Math, statics and optimization". A continuación, se va a proceder a presentar un ejemplo de uso del toolbox "Classification Learner".

Para ilustrar el funcionamiento de éste programa en materia de Machine Learning, se ha procedido a analizar un fichero de datos en formato CSV que contiene

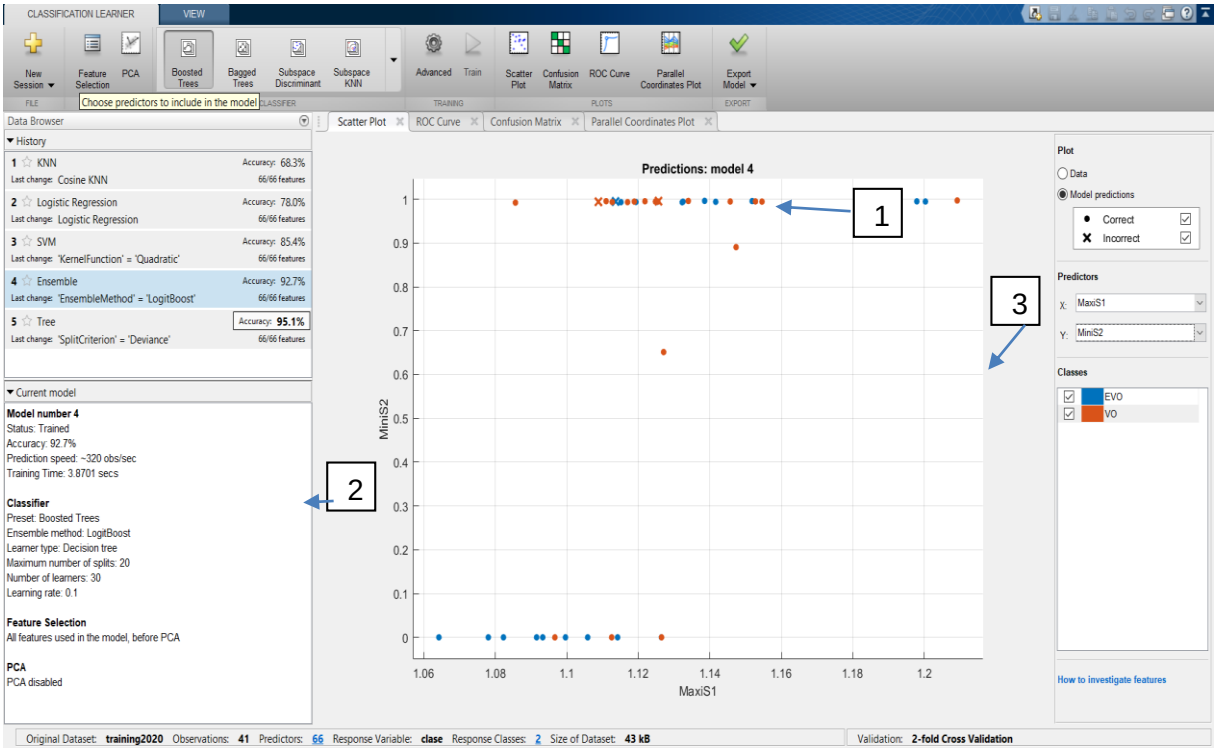



**Ilustración 4: Toolbox Classification Learner Matlab**

varias muestras de aceite. Para ello se ha inicializado el toolbox "Classification Learner".

En la interfaz gráfica de éste toolbox, se pueden distinguir varias regiones:

- La región superior (1) está presidida por las diferentes herramientas de que se dispone. En la parte izquierda de éstas, se sitúan las opciones de selección de atributos de entrada tanto manual como mediante análisis de componentes principales. Seguidamente se halla el menú de clasificadores disponibles, pudiendo elegir entre una amplia variedad de ellos. Los

clasificadores tales como perceptrón multicapa y similares no están disponibles en éste toolbox ya que poseen el suyo propio. En el caso del ejemplo, se han realizado varios entrenamientos con distintos clasificadores, siendo el modelo creado por el clasificador LogitBoost (que se explicará más adelante), el que obtiene los resultados que se pueden ver en la ilustración 4. La opción colindante a esto, advanced, permitirá configurar el clasificador con parámetros distintos de los que vienen por defecto, y las opciones siguientes son relativas a la información del entrenamiento que se desee visualizar.

- En la parte izquierda superior (2), se puede observar el historial de los clasificadores con los que se ha realizado el entrenamiento, así como de la precisión obtenida por los mismos.
- La región central (3) contiene una representación gráfica del resultado obtenido por el clasificador en el entrenamiento. En la Ilustración 4 se ha representado en el eje x los valores máximos del sensor 1, mientras que en y se tienen los valores mínimos del sensor 3. Nótese que ésta región está compuesta por varias pestañas, con lo cual se puede visualizar, como en éste ejemplo, la curva ROC o la matriz de confusión entre otros.

### 1.4.3. IBM SPSS Modeler

SPSS Modeler es un software propietario de IBM. Sus inicios se remontan hasta 1994, fecha en la que Integral Solutions Limited, crearon la primera versión de éste bajo el nombre de Clementine. Éste software fue comprado por SPSS, que más tarde sería comprado a su vez por IBM en 2010, empresa que le dio el nombre con el que se le conoce en la actualidad.

Éste software tiene características interesantes como el soporte a código abierto, como R o Python, o un modelado automático en el cual se construyen varios modelos con distintos clasificadores y se comparan los resultados a fin de escoger el mejor.

Como RapidMiner, éste software permite crear programas mediante la programación visual de bloques, lo que le aporta la simpleza de uso de la que carecen, entre otros, R.

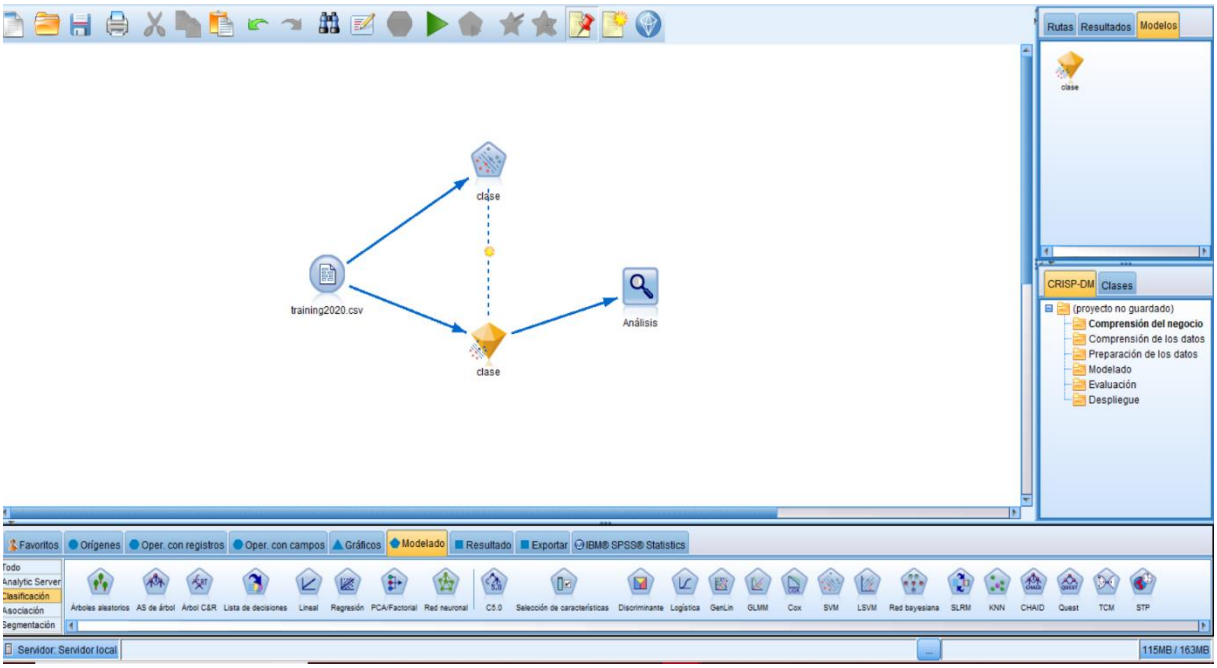



**Ilustración 5: Pantalla Proramación IBM SPSS Modeler**

Prueba de ello es el programa de ejemplo que se puede observar en la ilustración 5. El programa creado lee el fichero de muestras en formato CSV y construye un modelo usando el algoritmo de máquina de vector soporte. Una vez creado el modelo, se analiza estadísticamente su rendimiento.

Como se puede apreciar en la ilustración, la región central es la usada para la elaboración del programa, mientras que los bloques que se usarán para ello están en la parte inferior de la pantalla. Cabe asimismo la posibilidad de modificar los parámetros por defecto de cada bloque haciendo doble click izquiero sobre los mismos.

Una vez ejecutado el programa, se abrirá una ventana nueva de resultados, debido a la presencia del bloque de análisis, que evalúa el modelo creado.

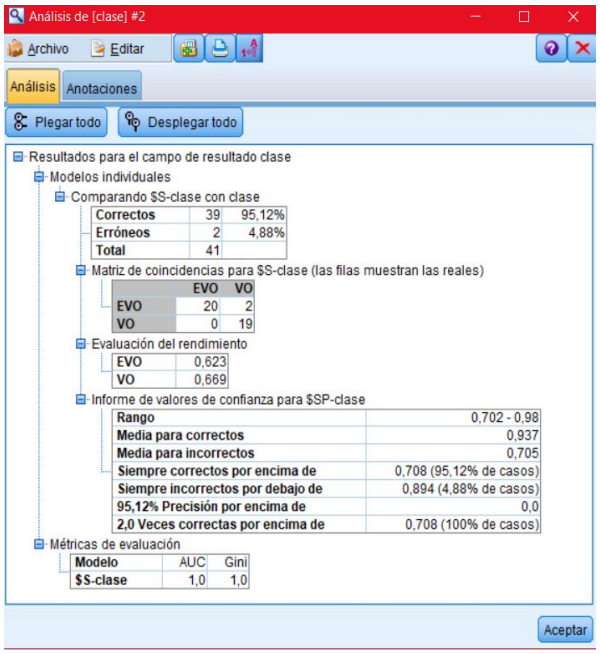



**Ilustración 6: Resultados Evaluación**

En ésta ventana, se muestran los parámetros estadísticos que evalúan la adecuación del clasificador al problema planteado. Los valores que se muestran son configurables desde el bloque de análisis.

### 1.4.4. RStudio

RStudio es el entorno de desarrollo integrado para R, un lenguaje de programación orientado a objetos enfocado en el campo de la estadística. Éste lenguaje de programación, desarrollado en la Universidad de Auckland en 1993 por Robert Gentleman y Ross Ihaka, es uno de los más importantes en el ámbito científico debido a su amplio número de herramientas estadísticas. Aunque éste lenguaje de programación posee diferentes interfaces gráficas, en éste ejemplo se ha utilizado RStudio, que es un software que, a diferencia de los anteriormente mencionados, es open source. Como contrapunto respecto a los demás cabe destacar que, al ser un entorno gráfico basado en R, es algo más dificultoso en su uso.

En la ilustración 7 se puede observar un breve programa realizado con el mismo set de datos que en los ejemplos anteriores. En él, se realiza una clasificación por medio de una máquina de vector soporte con un núcleo lineal.

Como se puede apreciar en la figura, la interfaz consta de cuatro partes diferenciadas:

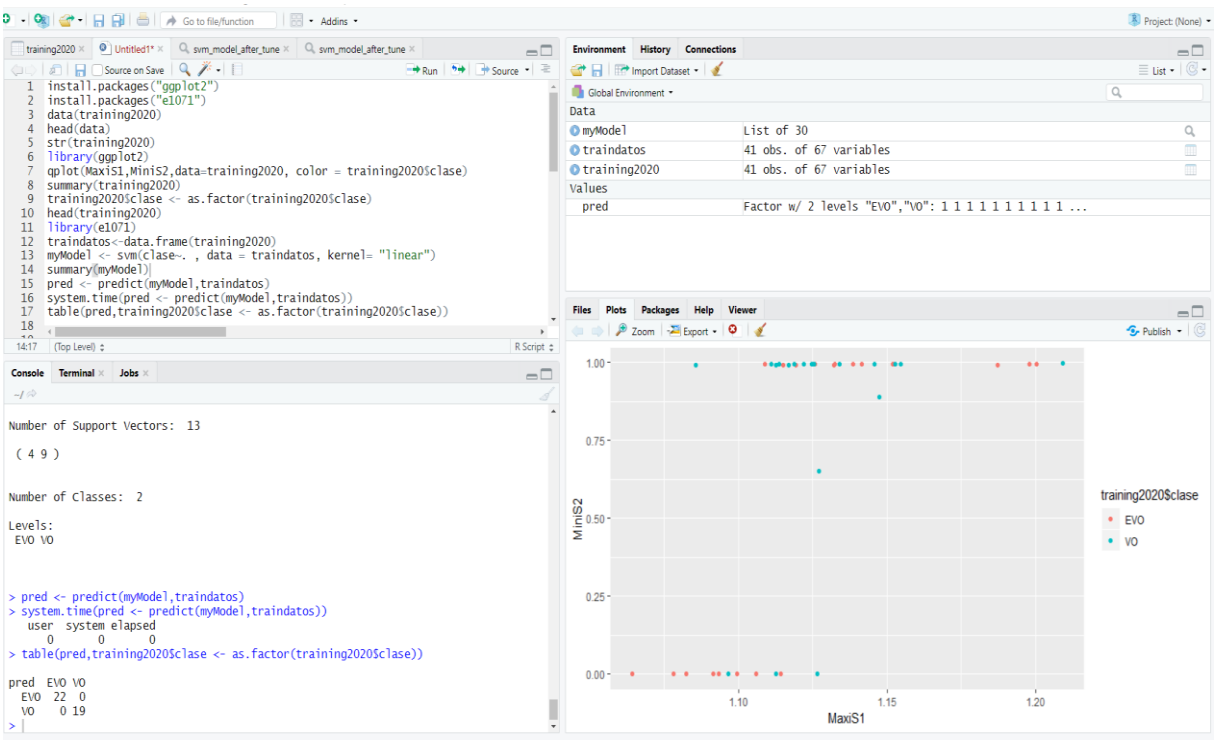



**Ilustración 7: Entorno gráfico de RStudio**

- En la esquina superior izquierda se encuentra el editor de R, que es el lugar donde se escribirán las líneas de código que conformen el programa.
- En la parte inferior del editor se encuentra la consola, que la zona donde se puede observar cómo se ejecuta el código, y sus resultados o errores.
- La esquina superior derecha almacena las variables y el historial de las líneas de código ejecutadas.
- En la parte inferior derecha, se encuentran las representaciones gráficas que se hayan programado para ejecutarse.

### 1.4.5. Weka

Weka (Waikato Environment for Knowledge Analysis), es un entorno para el análisis de datos escrito por entero en Java. Fue desarrollado inicialmente en C por la Universidad de Waikato en 1993, aunque en 1997 se empezó a reescribir en Java.

Es ésta característica la que lo hace adecuado para éste trabajo, ya que es un software open source cuyas librerías y funciones, por tanto, se pueden utilizar en otros entornos gráficos.

Éste entorno dispone de una interfaz gráfica, facilitando en gran medida su uso por parte de los usuarios más principiantes.


**Ilustración 8: Selector de interfaces Weka**

Una vez se ejecuta el programa Weka, la primera pantalla de la interfaz gráfica da a elegir entre varias interfaces:

- Explorer: En ésta interfaz, se realizan las siguientes operaciones sobre un mismo archivo: preprocesado, clasificación, asociación, selección de atributos y visualización.
- Experimenter: En ésta interfaz se podrá comparar el rendimiento de distintos clasificadores sobre el mismo conjunto de datos. Asimismo, también se permite la modificación de los datos de entrada.
- KnowledgeFlow: Como en RapidMiner o IBM SPSS Modeler, ésta interfaz de Weka también permite la programación visual mediante bloques preprogramados.
- SimpleCLI: En ésta interfaz se trabaja de forma parecida a como se hacía con RStudio: mediante código. En ella, se pueden realizar todas las tareas que se realizan en la interfaz explorer, ya que se tiene pleno acceso a todas las clases de Weka.

Para la tarea de selección de los clasificadores más adecuados para la problemática que se desea abordar, se ha utilizado la interfaz explorer y en el siguiente apartado de éste documento se detallará su uso.

# 2. SOFTWARE UTILIZADO

En éste apartado se describirá el software usado durante el trabajo. Para ello, se procederá a explicar las nociones básicas acerca de su uso, remarcando de entre todas sus funcionalidades, las que han sido de mayor utilidad en éste trabajo.

## 2.1. Weka

Como se introducía en el anterior punto, Weka es un entorno para el análisis del conocimiento escrito en Java que posee varias interfaces. Éste punto se va a centrar en describir con detalle las funcionalidades de la interfaz explorer, que permitirán estudiar sets de datos entrantes para construir modelos con los que poder realizar una clasificación verosímil de nuevas instancias.

### 2.1.1. Preprocess

Para crear un clasificador con ésta interfaz, lo primero será leer el archivo sobre el que se va a trabajar, en la pestaña Preprocess.

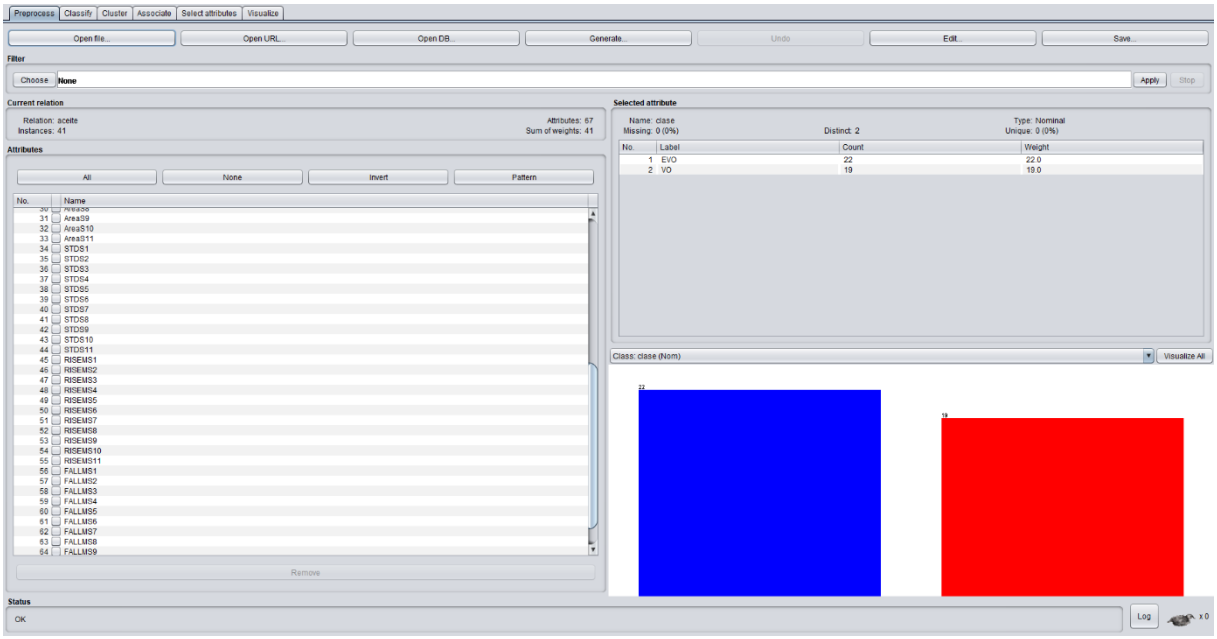



**Ilustración 9: Pestaña Preprocess de Explorer**

Los ficheros de entrada pueden estar en formato CSV, pero Weka tiene su propio formato de archivos de entrada, el ARFF. En ellos, los datos de entrada tienen la siguiente estructura:

```
@relation aceite,
@attribute MaxiS1 numeric
,@attribute MaxiS2 numeric
,@attribute MaxiS3 numeric
,@attribute MaxiS4 numeric
,@attribute MaxiS5 numeric
,@attribute MaxiS6 numeric
@attribute clase {EVO,VO},
@data,
1.15183684872,3.22703560898,2.56541295817,1.90110780870,1.00028009243,1.00914634146,1.00243309002,EVO
1.07802631578,2.46683573936,2.21826681049,1.73455316892,1.00339709031,1.01823708206,1.00242718446,VO
1.11418979409,3.15442561205,2.57304844906,1.94626544868,1.00202020202,1.01515151515,1.00242130750,VO
1.10581964348,3.86303758233,2.66544218936,2.52027027027,1.00365317229,1.03939393939,1.00241545893,EVO
1.06419382504,2.47449420715,2.25588310254,1.78076416337,1.00606729178,1.01215805471,1.0,1.58254994124,VO
```

**Ilustración 10: Formato de archivo ARFF**

Como se puede ver en la ilustración 10, la primera línea del archivo se dedica al nombre del mismo. Los atributos de los que se conforman las instancias deben ir a continuación de éste. Cada uno de ellos debe tener la siguiente estructura: “attribute” + nombre del atributo + tipo del mismo. Los tipos admisibles son: numérico (tanto de tipo entero como real), cadena de texto, fecha (cuyo formato para fecha completa es: “dd-mm-yyyy HH:mm:ss”) y enumerado (una variable de tipo enumerado es aquella que toma su valor dentro de una lista de posibilidades).

Como se puede distinguir en la ilustración 10, las variables atributo son numéricas (reales en éste caso), mientras que la variable de clase es del tipo enumerado, pudiendo tomar los valores tanto de EVO como de VO.

Una vez se han definido todos los atributos, así como la variable de clase, se habrá completado la parte de la cabecera del archivo. Así pues, lo siguiente en la creación de éste, será la relación de los datos obtenidos que se pasan como entrada al clasificador. A ésta sección del archivo se la conoce como datos y su primera línea será “@data”.

Nótese que, para la declaración de la relación, los atributos y los datos, se empieza siempre la línea con el carácter “@”. No pasa así con los datos de los que se compone el set de datos.

Cuando ya se ha leído el archivo en un formato válido de entrada (CSV o ARFF en éste caso), se pueden realizar diferentes operaciones sobre el set de datos contenido en el fichero, tales como eliminación de variables atributo, o aplicación de

filtros que realicen diferentes tareas sobre éstos. Los filtros pueden actuar sobre las variables que componen una instancia (filtros de atributo), seleccionando los más relevantes o incluso cambiando su tipología; o actuar sobre el conjunto de instancias (filtros de instancia), produciendo una sub-muestra del conjunto de entrada.

Ésta pestaña también posee un gráfico en la esquina inferior derecha, que muestra los valores que toma cada clase para cada atributo.

### 2.1.2. Classify

Una vez leído el fichero y aplicados, en caso de ser necesario, los filtros, se accede a la pestaña de clasificación (ver ilustración 11).

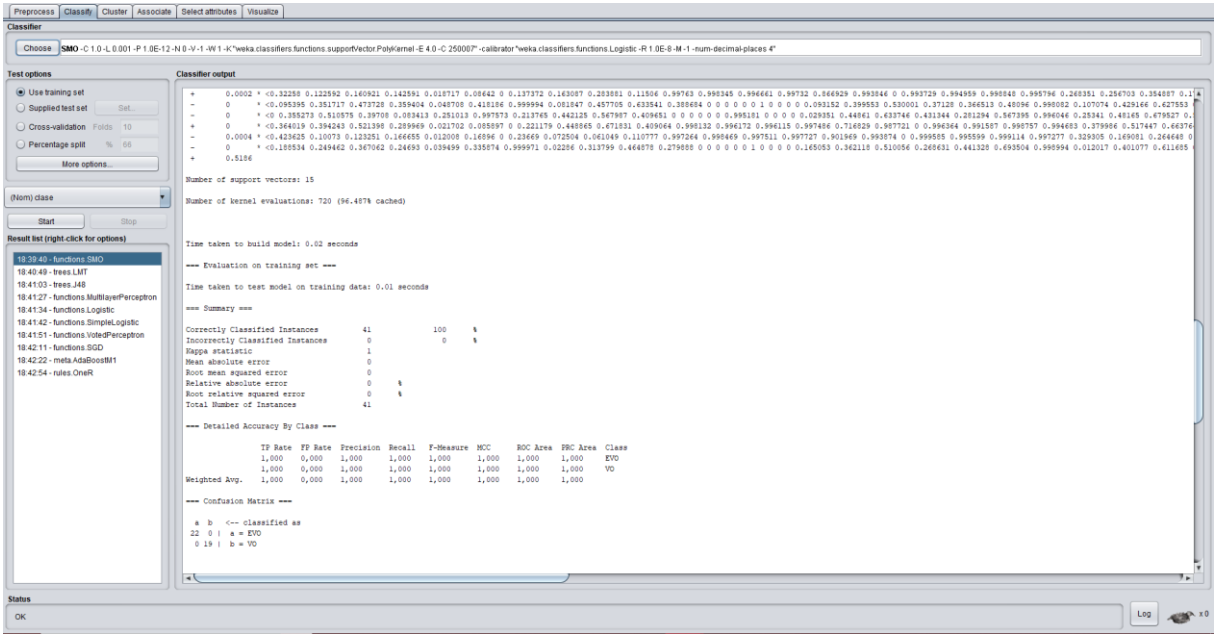



**Ilustración 11: Pestaña Clasiffy**

En ella, se selecciona el algoritmo de clasificación de entre los disponibles en Weka, y se presiona “Start” para crear el modelo. Cabe mencionar que, para la creación del modelo, pueden seleccionarse las opciones de “validación cruzada”, y “porcentaje de partición”. Éstas opciones, grosso modo, lo que hacen es crear el modelo sobre una sub-muestra del conjunto, y validar éste modelo creado, con las instancias que quedan fuera del grupo de entrenamiento. Asimismo, se pueden

modificar los parámetros de cada algoritmo mediante el área de texto que hay al lado del botón “choose”.

Los tipos de clasificadores disponibles, así como su funcionamiento interno, se detallarán más adelante.

Una vez creado el modelo, se puede introducir un fichero de datos similar con instancias nuevas, y re-evaluar el modelo para que éste haga una estimación de la clase a la que pertenecen las instancias introducidas. La exactitud de ésta estimación se evaluará en base a unas medidas estadísticas que se explicarán más adelante, tales como la Kappa de Cohen o área ROC.

De manera análoga a ésta, se procede en las pestañas de “Cluster” (que contienen algoritmos para agrupar las instancias en base a las semejanzas o diferencias entre éstas instancias), “Associate” (que contiene algoritmos que buscan relaciones existentes en los datos de la muestra), y “Select Attributes” (con los algoritmos de ésta pestaña se pueden seleccionar los atributos más relevantes de las instancias a la hora de clasificarlas).

### 2.1.3. Visualize

La pestaña visualizar, contiene una representación en dos dimensiones de la distribución de la clase de las instancias en la cual, los ejes están formados por todas

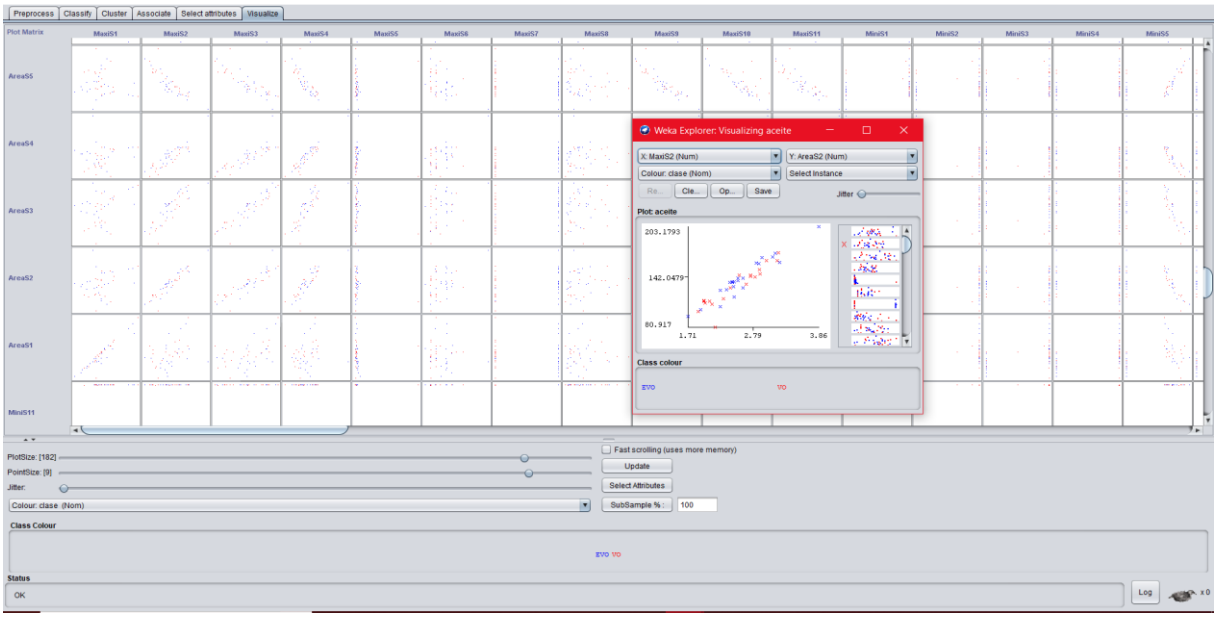



**Ilustración 12: Pestaña Visualize**

las posibles combinaciones de los atributos de entrada tomados de dos en dos (ver ilustración 12). Ésta pestaña puede ser muy útil para hallar de forma gráfica correlaciones entre atributos.

## 2.2. NetBeans

NetBeans es un entorno de desarrollo integrado (IDE) para la programación, principalmente, en Java, C/C++ y PHP, siendo Java el lenguaje más usado en éste entorno. Es un programa "open source" sin restricciones, cuya primera versión (NetBeans 3.1) vio la luz en diciembre del año 2000 de la mano de Sun Mycrosystems. Éste programa permite crear y editar código, compilarlo, ejecutarlo y depurarlo de una manera cómoda mediante puntos de ruptura, pudiendo ver el valor que toman las variables en cada momento. Asimismo, posee un potente creador de interfaces gráficas, que permite crearlas de forma fácil y rápida.

En éste trabajo, se va a realizar toda la programación en Java, ya que es un lenguaje de programación orientado a objetos muy potente y popular, y el idioma en el que están escritas todas las librerías de Weka. Por lo cual, se hace necesaria una introducción al mismo.

### 2.2.1. Java

Java es un lenguaje de programación orientado a objetos desarrollado por James Gosling, de Sun Mycrosystems, y publicado en 1995. Nació con el objetivo de ser un lenguaje que se pudiese ejecutar en cualquier dispositivo (multiplataforma), mediante el uso de lo que se llama "Java Virtual Machine" o JVM, que interpreta el código de Java y lo traduce. Es un lenguaje que deriva de C++, al cual le debe en gran parte su sintaxis, pero al contrario que éste, Java es 100% orientado a objetos: en Java, todo es un objeto, y todo está alojado en alguna clase.

Se entiende por programación orientada a objetos al paradigma de programación que utiliza objetos en la construcción del programa. Esto es, entidades formadas por propiedades (atributos) y funcionalidades (método), modelizadas mediante clases. Ésta es una manera de programar más ajustada a la realidad, cuyos pilares básicos son:

- Abstracción: Representación de las características esenciales del objeto, es decir, las que distinguen un objeto de otros.
- Encapsulamiento: Se reúnen todos los elementos que se pueden considerar pertenecientes a la misma entidad, en el mismo nivel de abstracción (clase).
- Ocultación: Los objetos están aislados del exterior, y sólo pueden cambiarse los valores de los atributos de éstos, mediante métodos (interfaz del objeto). De éste modo, se protegen las propiedades de éste contra modificaciones incorrectas o indebidas.
- Polimorfismo: Es la cualidad de los objetos de responder de forma distinta a un mensaje sintácticamente igual. De éste modo, por ejemplo, se podría acceder a métodos que tienen el mismo nombre, pero distinto cuerpo.
- Herencia: Las clases se pueden jerarquizar. Así, se pueden crear clases genéricas y subclases, que heredarán las características de las anteriores. Nótese que éste concepto está íntimamente ligado con el polimorfismo.

Cabe mencionar, que en Java una clase sólo puede heredar de otra, eliminándose así la herencia múltiple que sí permite, en cambio, C++ entre otros. No obstante, hay una excepción, ya que todas las clases de Java heredan de la "super-clase" Object, pero esto no se considera herencia múltiple. Aunque no se puede dar la herencia múltiple en clases, sí que se pueden implementar distintas interfaces para cada clase. Éstas pueden contener métodos predeterminados con el mismo nombre.

### 2.2.2. Interfaz de usuario de NetBeans

Una vez presentado el lenguaje de programación que se va a utilizar en NetBeans, se pasará a describir éste entorno.

La ilustración 13, muestra la interfaz de NetBeans. En ella, se pueden apreciar cuatro zonas diferenciadas:

- En la parte superior izquierda, se muestran todos los paquetes (package) de los que está formado el proyecto, así como las librerías que se han importado para su uso. Un paquete, es una agrupación de clases que guardan algún tipo de relación entre sí. Ésta agrupación es muy importante ya que pueden existir clases con el mismo nombre, pero no en el mismo paquete. Nótese

que, como se ve en la ilustración 13, para usar un método de una clase, hay que importar el paquete al que pertenece dicha clase.

- Debajo de ésta ventana se encuentra el Navegador. En él, aparecen las clases, constructores (métodos que construyen los objetos de la clase),

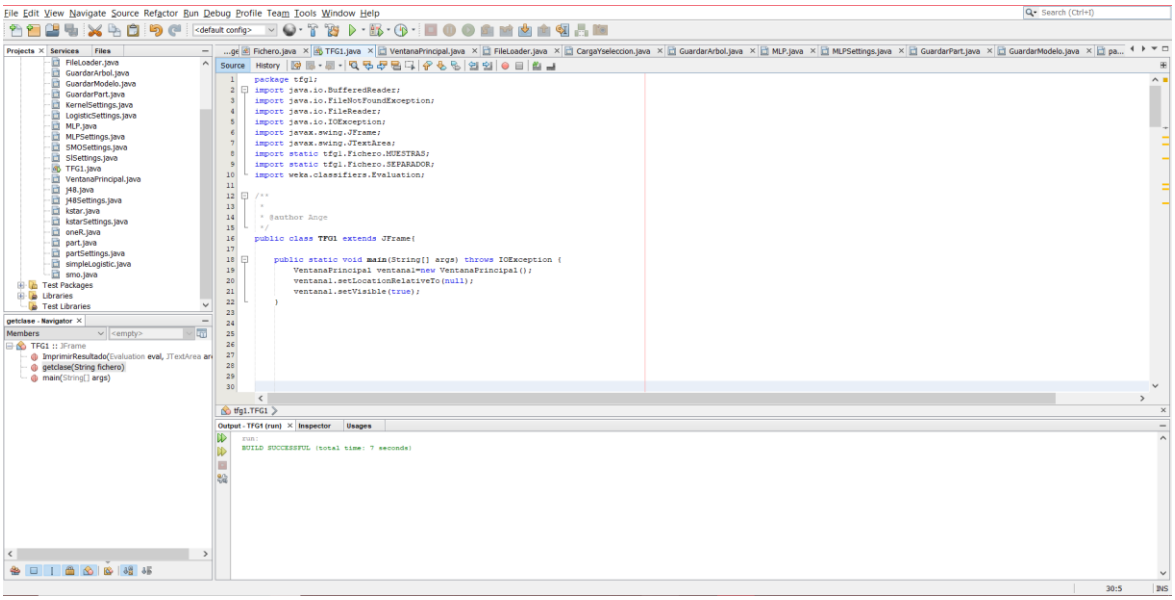


**Ilustración 13: Interfaz NetBeans**

métodos y variables que definen los objetos de la clase del paquete en el que se está en ese momento. De éste modo, se puede viajar rápidamente a la implementación de éstos métodos o a la declaración de la variable deseada.

- En la parte central superior, se encuentra la ventana de edición. Ésta es la zona donde se puede escribir el código del programa a implementar.
- La parte inferior a la ventana de edición está reservada para la consola. En ella, se muestran los mensajes que el programa imprima en consola, así como los posibles errores que pueda haber en periodo de compilación o ejecución. Otra utilidad es la de poder ver, en modo de depuración, los valores que toman las variables a cada momento.

### 2.2.3. Creación de una interfaz gráfica

NetBeans ofrece, como se ha dicho con anterioridad, una potente herramienta para la creación de interfaces gráficas de un modo rápido y muy intuitivo. Para ello, se tendrá que crear una clase dentro del paquete, eligiendo la opción de “JFrame Form” o de “JDialog Form”, ambas clases contenidas en el paquete “javax.swing”. Por norma general, un programa creado en Java tendrá un solo “JFrame Form”, y todas las demás ventanas serán “JDialog Form”. Esto se considera lo apropiado ya que,

mientras que los “JDialog” pueden jerarquizarse, los “JFrame” no. Es decir, a éstas ventanas, al crearlas, se les puede asignar una ventana “padre” y que, de éste modo, se muestren delante de ella y nunca detrás. Otro de los motivos por los que se debería limitar el uso de JFrame a uno por aplicación, es el hecho de que ésta ventana es visible en la barra de tareas de Windows, por lo que parece coherente que la aplicación

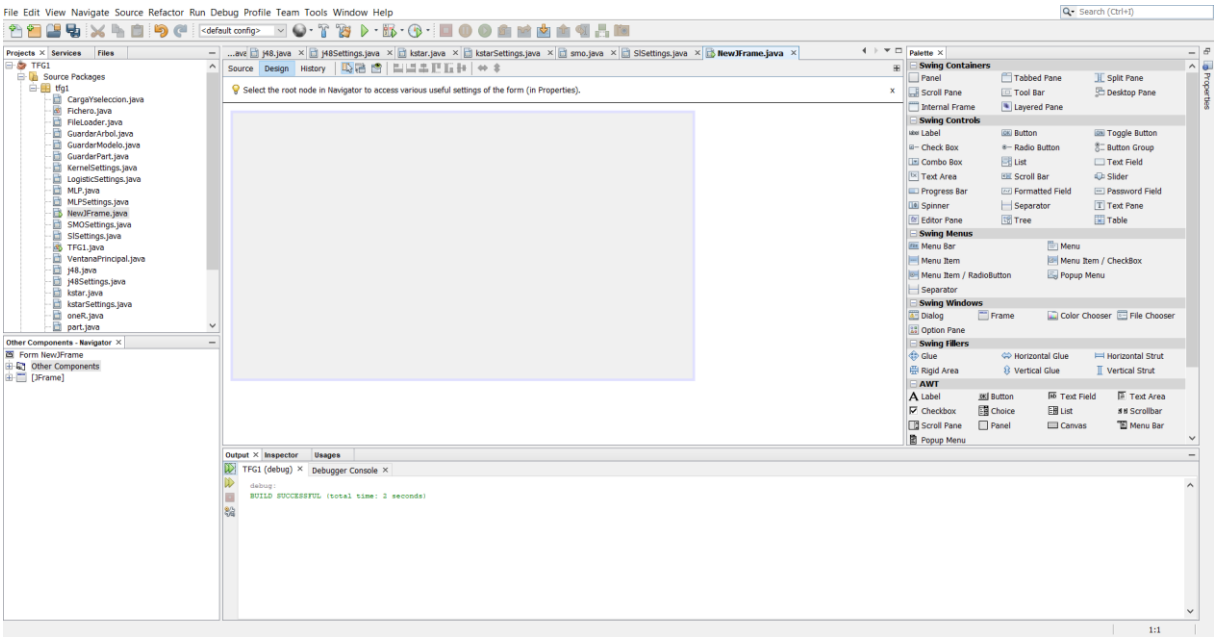


**Ilustración 14: JFrame en NetBeans**

aparezca sólo una vez ahí.

Una vez creada la ventana de acuerdo a lo anteriormente expuesto, se tiene la ventana que se muestra en la ilustración 14. En ella, se puede ver en la parte central, correspondiente al editor de código, una representación gráfica de la ventana creada, donde se podrán agregar los elementos deseados por el usuario. Éstos elementos se encuentran, como se puede observar, en la parte derecha de la interfaz gráfica. Una vez añadidos los elementos que van a componer la ventana, se les podrá dotar de una función. Por ejemplo, si se añade un botón que se quiere que abra una ventana nueva o realice cualquier otra opción, se tendrá que crear un evento de tipo “ActionPerformed”. Una vez creado éste, se podrá programar su función, y las de cualquier otro elemento de la ventana, ingresando en la pestaña “source” de la interfaz, donde se encuentran codificados tanto la ventana en sí misma como todos los elementos que se han ido añadiendo.

Como se puede observar, éste modo de crear interfaces gráficas constituye una de las principales ventajas de NetBeans con respecto a otros entornos de programación. Pero ésta no es la única ventaja de NetBeans: Al ser un programa “open source”, cuenta con un amplio soporte en la comunidad, y su editor de texto tiene un resaltador de sintaxis, algoritmos predictivos y consejos acerca de la optimización del código.

# 3. CLASIFICADORES SELECCIONADOS

En ésta sección se explicará, de manera pormenorizada, el funcionamiento de los clasificadores que se han implementado en la herramienta creada. El proceso de selección de éstos, se ha realizado atendiendo a diversas pruebas realizadas con sets de muestras, en las que se han utilizado los distintos clasificadores implementados en Weka. Éstos están agrupados en categorías según los métodos de clasificación que usan. Se han añadido a la herramienta los clasificadores, de cada categoría, que mejor desempeño han mostrado, a fin de esclarecer qué método es más eficaz para la tarea de clasificación que se aborda en el presente trabajo.

## 3.1. Entropía de Shannon

Antes de empezar a hablar del funcionamiento interno de los clasificadores, se hace necesario explicar un concepto fundamental que sirve como base para el funcionamiento de éstos: En 1948, Claude Shannon, en su publicación “Una teoría matemática de la comunicación”, fundó el campo de la teoría de la información, una rama de las matemáticas que trata de modelizar la transmisión y el procesamiento de la información.

Ésta teoría [1] aborda el problema de la construcción de un codificador, que transforma el mensaje de un emisor o fuente para que se pueda enviar de un modo eficiente y sin errores (que pueden ser debidos al ruido que introduce el canal), y un decodificador que tiene que adivinar el mensaje de la fuente, conociendo las transformaciones que el codificador aplica a éste. La ilustración 16 muestra un esquema típico del modelo matemático de la teoría de la información.

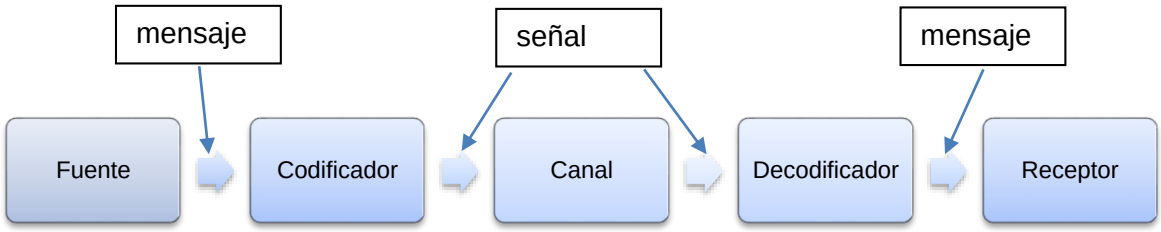



**Ilustración 16: Esquema de emisión de información**

El pilar sobre el que se apoya ésta teoría, es la llamada entropía de Shannon, cuya formulación matemática se encuentra representada en la Fórmula 3.1.1.

$$H(x) = -\sum_{x \in A} p(x) * \log_2 p(x)$$

*(Fórmula 3.1.1)*

Siendo A un conjunto de posibles valores de la variable x.

Como se puede apreciar, la entropía es una función que depende de la probabilidad de ocurrencia de los valores de una cierta variable x.

Especial atención requiere siguiente término dentro de la formulación de entropía:

$$I(x) = -\log_2 p(x)$$

*(Fórmula 3.1.2)*

Éste término representa la información y, como se puede comprobar, decrece con el aumento de la probabilidad de suceso de x. Esto lleva a una conclusión muy importante, y es que la información contenida en un evento que ocurre con una alta probabilidad, es menor que la de un evento con probabilidad baja. Esto es una idea bastante interesante ya que resulta razonable pensar que, para explicar la ocurrencia de un suceso, sería más interesante estudiar los casos en los que el valor de la variable ofrece poca probabilidad de éxito (casos extraños), ya que pueden contener un elemento diferencial: El comportamiento de un suceso se explica mejor por las

causas cuya probabilidad de ocurrir, habiendo ocurrido el suceso, es menor de un valor determinado. Esta manera de pensar, es ampliamente usada, por ejemplo, en los análisis de varianza. Para ilustrar esto, se puede pensar en un texto en español: la letra s es muy probable, por lo que daría poca información acerca de la palabra que la contiene. En cambio, la ñ, contiene más información al ser mucho menos probable.

Teniendo en cuenta lo anteriormente expuesto, la entropía de una variable no es más que su información media, expresada en bits (nótese que está presente un logaritmo de base dos en la expresión matemática de la entropía). Es decir, que la cantidad de información que contiene un mensaje es proporcional al número de bits necesarios para su representación.

Obsérvese entonces, que la entropía representa el “desorden” o incertidumbre de un mensaje.

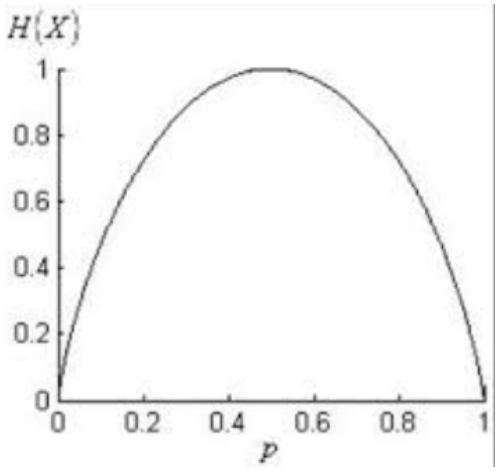



**Ilustración 17: Representación H frente a p**

En la ilustración 17, se representa la entropía de una variable binaria frente a la probabilidad de que ésta sea uno. Si se calculase el máximo de la función de la entropía, derivando respecto a la probabilidad, se comprobaría que éste se da cuando la ocurrencia de un suceso es equiprobable, como se ve en la ilustración. Es decir, la máxima incertidumbre se produce cuando todos los valores de la variable de estudio son equiprobables. Para ilustrar esto, puede pensarse en una moneda: la probabilidad de ocurrencia es la misma para las dos caras, por lo cual el experimento tiene máxima incertidumbre, al ser un experimento azaroso.

Y es que la entropía mide la aleatoriedad de una variable: si se considera una fuente que emite mensajes en idioma español, en un principio no se puede adivinar qué letra va a seguir a otra, pero, como se ha dicho antes, hay más probabilidad de que se den unas letras que otras, por lo que el proceso no es tan aleatorio como puede parecer, y la encargada de medir la aleatoriedad de la fuente será la entropía.

#### 3.1.1. Entropía Condicionada

Teniendo claro el concepto de entropía, se introduce ahora el concepto de entropía condicionada [2].

Sea una variable aleatoria bidimensional (x, y), se define la entropía condicionada de la siguiente forma:

$$H(X/Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} P(y_j) * P(x_i/y_j) * \log_2(P(x_i/y_j))$$

*(Fórmula 3.1.1.1)*

Esto es, la incertidumbre, o información media de X cuando se conoce Y. Ésta entropía se usa en procesos con varias variables aleatorias. Nótese que, si X, Y son variables independientes, entonces H(X/Y) será igual que H(X). Por lo cual, mediante la entropía condicional se puede medir la dependencia entre dos variables.

## 3.2. Árbol de decisión J48

El árbol de decisión J48 [3] es una implementación realizada en Weka del algoritmo de clasificación C4.5. Éste es un algoritmo creador de árboles de decisión implementado por Ross Quinlan en 1993, a partir del ID3. En éste clasificador cobra gran importancia el concepto de entropía introducido con anterioridad.

La base sobre la que se asienta éste algoritmo es la construcción de un árbol de decisión con el método llamado “dividir y conquistar”, que se basa en lo siguiente: si se tiene un set de instancias (datos) pertenecientes a varias clases, éste se dividirá en varios subsets que tiendan a contener instancias pertenecientes a sólo una clase.

Éstas particiones se realizarán mediante tests, realizados sobre el set de instancias, con atributos que tienen salidas mutuamente excluyentes. Éstos tests constituyen los nodos de decisión del árbol, que tendrán tantas salidas o ramas como clases distintas contenga ése atributo. El proceso sigue con esos subsets creados (ver ilustración 18) hasta que se llega a las hojas, que será la predicción de clase realizada por el clasificador. El criterio de paro se discutirá más adelante.

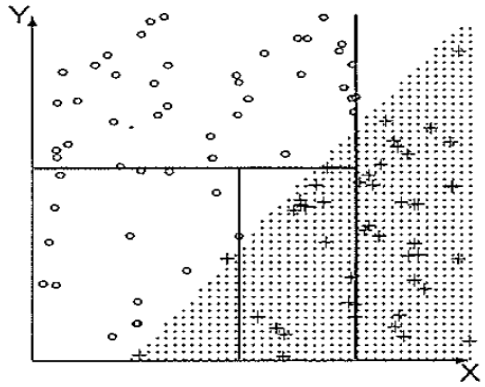



**Ilustración 18: Subdivisión Dataset**

### 3.2.1. Seleccionando los nodos del árbol

Como se ha comentado con anterioridad, éste algoritmo es una mejora del algoritmo ID3, creado por el mismo Quinlan en 1986. También se ha mencionado que la idea básica subyacente en la creación de un árbol de decisión, con el método "dividir y vencer", utiliza tests de atributos sobre el set de instancias. En éste punto se hablará sobre el criterio de elección de esos tests.

El criterio de selección de tests implementado por el algoritmo ID3, usa lo que se llama ganancia de información. Sea un set de entrenamiento T, la ganancia de información, se formulará del siguiente modo:

$$Gain(T) = H(T) - H_x(T)$$

*(Fórmula 3.2.1.1)*

Donde

$$H_x(T) = -\sum_{i=1}^{n} \frac{T_i}{T} * H(T_i)$$

*(Fórmula 3.2.1.2)*

Siendo i=1…n, los valores que toma el atributo de test X, y:

$$H(T_i) = \sum_{j=1}^{k} P(Y_j) * \log_2 P(Y_j)$$

*(Fórmula 3.2.1.3)*

Para j=1…k, clases.

Se quiere remarcar que H (T) es la información media necesaria para identificar la clase de una instancia en T, es decir, la entropía, y que esto mide la homogeneidad de clases dentro de éste grupo.

Como se puede ver en las fórmulas anteriores, la ganancia de información es, por tanto, la información que se gana si se divide T de acuerdo a un test X. Por lo que, un criterio de división de subconjuntos basado en esto, buscará que la ganancia sea máxima, ya que ésta mide cómo clasifica el atributo escogido como test a las instancias testeadas. Es decir, la partición será tanto más homogénea como mayor sea la ganancia, ya que esto conlleva una disminución de la entropía del conjunto (H(T)), o, dicho de otro modo, a un ordenamiento del conjunto de entrenamiento T.

Nótese, sin embargo, que éste método no es el más conveniente ya que, si se selecciona una variable de test que tenga una alta ganancia pero que puede tomar muchos valores, ésta generará tantas ramas de salida como rango de valores tenga. Esto puede dar lugar a particiones de un solo ítem que, como se puede comprobar fácilmente, tienen una entropía igual a cero (no dan ninguna información). Más aún, el clasificador resultante tendrá mucho **sesgo**, es decir, el clasificador se sobreentrenará, con lo que se hará muy bueno prediciendo el set de entrenamiento, pero será incapaz de clasificar correctamente una nueva instancia: el árbol memorizará, pero no aprenderá.

Para evitar esto, el algoritmo C4.5 introduce una corrección a la ganancia de información, por medio de dividir ésta entre la entropía de la división de T en i subconjuntos.

$$Gain\ Ratio(X) = \frac{Gain(X)}{-\sum_{i=1}^{n} \frac{Ti}{T} * \log_2 \frac{Ti}{T}}$$

*(Fórmula 3.2.1.4)*

Esto es lo que se llama ratio de ganancia, y representa la razón de información que da la partición, que es útil para la clasificación. Se puede observar que, si se da el caso anteriormente expuesto de divisiones de una sola instancia, la información de la división será cero con lo que la solución de la ratio de ganancia será inestable. Así que, si bien el criterio será maximizarla, se impone la restricción de que la ganancia tiene que ser grande.

### 3.2.2. Subdividiendo el conjunto

Ahora que se ha explicado el criterio de selección de los tests o nodos del árbol, se va a pasar a explicar cómo se realizan éstos en base al tipo de atributo que se escoja:

- Atributos discretos: Para los atributos discretos se puede trabajar de dos modos. El modo ortodoxo será el visto con anterioridad, de una rama por salida. Pero también se podrán dividir los distintos valores de ésta en rangos, de tal modo que haya una rama por dominio.
- Atributos continuos y numéricos: si el atributo seleccionado es continuo y numérico, se procederá seleccionando un umbral (z). Para ello, se ordenarán los valores de atributo de forma creciente y se tomarán como candidatos los valores medios formados por un punto y su siguiente, que no excedan el punto medio de todo el set de datos. La selección del umbral adecuado de entre todos los posibles, se realiza mediante el criterio de la ganancia anteriormente expuesto. Una vez hecho esto, se generará una clasificación booleana (a<=z, a>z) a partir de éste umbral.

Para la clasificación de las instancias se introduce una restricción más: para cualquier división, al menos dos de los subsets Ti deben contener un número no trivial de datos (en Weka, esto está establecido en dos por defecto).

### 3.2.3. Tratamiento de los valores perdidos

Como se puede inferir de lo anteriormente expuesto, un valor perdido puede ser un problema en varias etapas del algoritmo.

- A la hora de seleccionar los nodos del árbol, ya que la ganancia de información se puede definir como la información que se requiere para saber la clase antes de la partición candidata, menos la información requerida para saber la clase después de la partición, un nodo no puede dar información de pertenencia a una clase de casos de valores perdidos. Así que, con objeto de solventar éste problema, sea F el porcentaje de instancias de T cuyo valor de atributo del nodo de test es desconocido, se implementa la siguiente formulación para la ganancia, en la que se ha tenido en cuenta éste valor.

$$gain(X) = F(H(T) - H_X(T))$$

*(Fórmula 3.2.3.1)*

Para el divisor de la ratio de ganancia (Entropía de la división), se puede considerar que los casos con valores desconocidos son un grupo más.

- En la etapa de partición del set T, se puede dar que el test elegido tenga salidas desconocidas. Para asignar una instancia con valor de atributo desconocido a un subset Ti, se procede del siguiente modo: primero se asigna un peso a cada instancia clasificada en un subset Ti, que es la probabilidad de pertenencia a ése subset. De éste modo, si la salida es conocida, el peso será 1. Pero si el valor es desconocido, el peso es la probabilidad del valor de la variable en ese punto. Es importante recalcar que el peso de una instancia que pertenece a un subset y que es conocido, no tiene por qué ser uno, ya que éste puede ser partición anterior. Así que, en general, una instancia en T con valor de variable de test desconocido, y peso w, se asigna a cada subset Ti con peso: W=w * p (valor atributo). La

probabilidad final de que la instancia tenga ese valor de atributo, se calcula sumando los casos del set T que tienen ese valor de atributo, dividido por la suma de las instancias en T con salida conocida en éste test. Así, en las hojas ahora lo que se encuentra es una distribución de probabilidad de tipo binomial N/E, donde de N casos que llegaron a la hoja, E no pertenecen a la clase.

- Cuando el valor perdido se da cuando se está clasificando una instancia nueva, y éste está presente en variables atributo que forman nodos del árbol, la salida del nodo no puede ser determinada. Así que lo que se hace es explorar todas las posibles salidas a partir del nodo con el valor perdido. Así, de un modo análogo a lo explicado con anterioridad, lo que se tiene cuando se llega al final del árbol es una probabilidad de pertenencia a la clase. La instancia es asignada a la clase con mayor probabilidad.

### 3.2.4. Poda

Éste método continúa de forma recursiva hasta que todas las particiones que se hagan estén en la misma clase, o hasta que la partición de los subsets, no ofrezcan mejora alguna. Esto suele dar lugar a una sobreadaptación no deseada, ya que, si bien el árbol se adapta plenamente al caso muy concreto del set de entrenamiento, hará generalizaciones malas. Esto se puede ver claramente en la ilustración 19.


**Ilustración 19: Ajuste Entrenamiento**

Ha llegado el momento de decidir cuándo parar de subdividir el set de datos. Para ello se evalúa la reducción del error que tendría la partición candidata, y se compararía con un umbral. Obsérvese que lo que se propone aquí es aumentar el

error de clasificación de las instancias de entrenamiento con vistas a que el árbol tenga mayor capacidad de generalización. Así, como se ha dicho anteriormente, las hojas contienen una distribución de probabilidad.

Una vez se tiene el árbol, se realiza el proceso de poda:

Para ello, y empezando por el fondo del árbol, se evalúa cada subárbol que no dé a una hoja. Si el reemplazo de éste por una hoja o por su rama más frecuente disminuye la tasa de error medio de predicción del árbol, se elimina éste. Así que aquí el problema es el cálculo de ésta tasa de error, ya que es evidente que no se podrá realizar con el set de entrenamiento, puesto que la poda aumenta el error sobre éste. Si una hoja cubre N instancias, de las cuales E son clasificadas de forma incorrecta, esto puede verse como una distribución binomial donde E es la ocurrencia del evento (en éste caso error). Así, para un límite de confianza establecido (normalmente es de 0.25, y así lo tiene por defecto Weka), el límite superior de la probabilidad de error se da mediante los límites de confianza para la binomial de la hoja: $U_{0.25}(E, N)$.

Por tanto, la hoja que cubre N instancias produciría un total de $N^* U_{0.25}(E, N)$ errores de predicción. Así, el número de errores de predicción asociado a un subárbol es la suma de errores de predicción de sus ramas.

Por todo lo anteriormente expuesto, la tasa de error para los árboles creados, es la suma de los errores estimados en las hojas dividido entre el número de casos de entrenamiento.

Pero C4.5 no trabaja con el árbol propiamente dicho para la poda, sino con el conjunto de reglas de clasificación asociadas a éste. Por esa razón, lo que se evalúa para su eliminación, son condiciones de éstas reglas. No obstante, el método seguido para su poda es el mismo: si una regla borra una determinada condición X del conjunto de ellas, y no deriva en un incremento de la tasa de error de predicción, la condición X se elimina. Por lo que la pregunta aquí es, ¿cómo se escoge la condición a eliminar? Esto se realiza mediante lo que se llama un algoritmo voraz o “greedy”, que elige en cada etapa la condición óptima, sin tener en cuenta el resto, esperando que así se llegue a una solución óptima, aunque esto no tiene por qué darse.

La eliminación de condiciones en las reglas puede llevar a que no sean mutuamente excluyentes, habiendo instancias que pueden clasificarse con más de una regla, o con ninguna. Para esto último, se crea una regla por defecto, mientras que, para solucionar posibles ambigüedades a la hora de clasificar, se establecerá un ordenamiento de las reglas de clasificación por su número de falsos positivos.

## 3.3. Algoritmo PART

En 1998, Frank Eibe e Ian Witten, ambos de la Universidad de Waikato [4], implementaron un método de reglas de decisión a partir del algoritmo C4.5.

Éste algoritmo usa, al igual que C4.5, el método dividir y conquistar visto con anterioridad para crear sus reglas, pero de un modo peculiar. La ilustración 20 muestra cómo se realiza éste proceso:

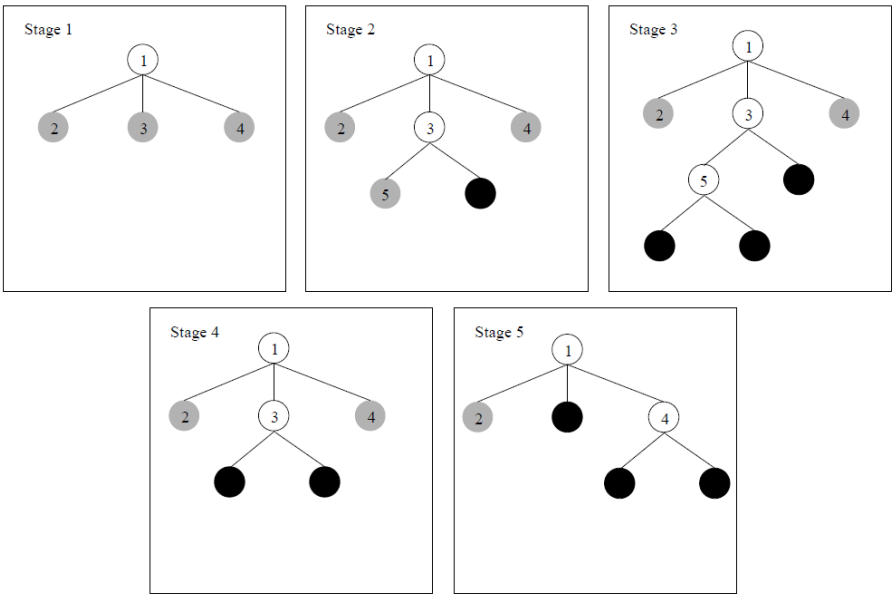



**Ilustración 20: Creación de regla en PART**

El primer paso, al igual que en C4.5, será elegir el atributo de test y dividir el set de instancias. Pero ahora, los subsets que se expanden primero son aquellos cuya entropía media es menor, al ser más probable que éstos den como resultado un árbol más pequeño (menos subdivisiones o regla más genérica). Nótese que esto es lo que ocurre en las etapas uno, dos y tres de la ilustración anterior.

Ésta metodología se repite hasta que uno de los nodos da como resultado sólo hojas (nodos de negro en la ilustración): Ese es el momento en el que se empieza a podar el árbol.

El algoritmo evalúa entonces su sustitución por una hoja, del mismo modo que hace el C4.5 y si se reemplaza, continúa haciendo lo mismo con los “hermanos” del nuevo nodo. Ésta poda se para en el momento en el que, haciendo el camino hacia atrás en el árbol, se llega a un nodo en cual en todas sus ramificaciones no lleve a hojas, quedando los subsets restantes inexplorados.

Éste procedimiento se puede ver claramente en las etapas tres y cuatro: al llegar el algoritmo al nodo cinco, se obtienen dos hojas, así que empieza la poda. De éste modo, se reemplaza el nodo cinco por una hoja, y a su vez ésta se compara con la otra del nodo tres, para su reemplazo si compete. Una vez terminado éste proceso, se continúa expandiendo el nodo 4, que es el siguiente con menos entropía. Ésta vez no ha tenido lugar el reemplazo.

Una vez se tiene construido éste “árbol parcial”, sólo se extrae una regla de él: la extraída de la hoja que cubre el mayor número de instancias.

Los valores de atributo perdidos se tratan del mismo modo que en el C4.5.

## 3.4. KSTar

Éste algoritmo fue creado en 1995 por John G. Cleary y Leonard E. Trigg [5], de la Universidad de Waikato. En él, se usa el concepto de la entropía como medida de distancia. Éste es un tipo de clasificador que está basado en instancias, es decir, dos instancias similares tienen la misma clasificación. Así que, lo que se trata de dirimir aquí será cuándo son similares dos instancias. Para ello, será necesaria una función de distancia, que mida cómo son de similares dos instancias, y una función de clasificación que discrimine éstas.

### 3.4.1. Función K*

La distancia [5] entre instancias será la complejidad para transformar una instancia en otra. Para ello, se define instancia a instancia, un conjunto finito de transformaciones que habrán de terminar con un símbolo de paro.

La distancia K* será la suma de todas las posibles transformaciones que se pueden dar para convertir una instancia en otra. Para poder sumar éstas transformaciones, cada una de éstas secuencias están asociadas con una probabilidad de suceso. En concreto, si “c” es la complejidad de una transformación, o camino, medida en bits, su probabilidad asociada será $2^{-C}$. Es decir, habrá una probabilidad de $2^{-C}$ de que una instancia se convierta en otra. La suma de éstas probabilidades será la distancia K*. Una vez realizado esto, si se toma el logaritmo en base dos, se tendrá la complejidad de transformación medida en bits.

$$P^*\left(b/a\right) = \sum_{t \in P : t(a) = b} p(t)$$

*(Fórmula 3.4.1.1)*

Esto es: la probabilidad de que un atributo “a” se convierta en uno “b”, es igual a la suma de las probabilidades de todas las transformaciones, o caminos que llevan a “a” hasta “b”. Así, K*, la función de distancia que se busca, queda definida como sigue:

$$K^*\left(b/a\right) = -\log_2 P^*\left(b/a\right)$$

*(Fórmula 3.4.1.2)*

### 3.4.2. Cálculo de la función P*

Éste algoritmo admite atributos numéricos y categóricos, así que será necesario el cálculo de la función de probabilidad de un modo distinto para cada uno.

Para los atributos numéricos reales, la distancia es proporcional a la diferencia, en valor absoluto, entre los dos valores de atributo.

$$K^*\left(b/a\right) = |a - b|$$

Y la función P*:

$$P^*(x) = \frac{1}{2x_0} * e^{-x/x_0} * dx$$

*(Fórmula 3.4.1.3)*

Esto es una función de densidad de probabilidad. La selección del parámetro $x_0$ es algo que se tiene que hacer con especial cuidado, ya que actúa como una medida de longitud de escala.

Del mismo modo, si el atributo es categórico, cosa que no se va a dar en éste trabajo, se procede del siguiente modo: si se tiene un set de valores de instancia "n", con probabilidades $p_i$, es obvio que la transformación de una instancia a otra será la conversión de un valor de atributo a otro. Si se llama "s" a la probabilidad de que ese valor de atributo de una instancia se mantenga igual (esto es el final de la cadena de transformaciones que se mencionaba anteriormente), se tiene:

$$P^*\left(j/i\right) = \begin{cases} (1-s) * p_j, & si\ i \neq j \\ s + (1-s) * p_i & si\ i = j \end{cases}$$

*(Fórmula 3.4.1.4)*

Éste valor s, al igual que $x_0$, se habrán de seleccionar de un modo apropiado.

Hasta ahora, se ha supuesto que las instancias tenían un solo atributo, pero si tienen más de uno, como parece ser razonable, el procedimiento es simple. Una vez se ha calculado la función P* de cada atributo, éstas se combinan entre sí. Es decir, la distancia entre dos instancias con múltiples atributos, será la unión de las distancias existentes entre cada atributo de éstas, o, expresado en términos probabilísticos:

$$P^*\left(I_2/I_1\right) = \prod_{i=1}^{m} P^*\left(V_{2i}/V_{1i}\right)$$

*(Fórmula 3.4.1.5)*

Esto es, la probabilidad de que una instancia $I_1$, se transforme en otra $I_2$, es igual a la probabilidad de que cada valor de atributo de una $V_{1i}$, se transforme en el valor de atributo de la otra $V_{2i}$.

### 3.4.3. Valores Perdidos

En éste caso, se asume que los valores perdidos del set de datos, puede tener cualquier valor, por tanto, la función de probabilidad de la distancia será la media de la probabilidad de ese valor perdido de ser transformado en cualquier valor de atributo del set de datos.

$$P^*\left(¿\,?/a\right) = \sum_b \frac{P^*\left(b/a\right)}{N}$$

*(Fórmula 3.4.3.1)*

Siendo N el número de instancias del set de datos.

### 3.4.4. Selección de $x_0$ y s

La selección adecuada de éstos atributos es, como se ha mencionado con anterioridad, de vital importancia para el desempeño de éste algoritmo.

Nótese que, al ser P* una probabilidad de transformación de unas instancias en otras, se tiene que determinar cuántas instancias contiguas a la actual se toman en consideración en ésta comparación. Así, éstos dos parámetros hacen que la distribución P* pueda ser una distribución en que sólo se tiene en cuenta el vecino más cercano (con los parámetros cercanos a 1), o en la que todas las instancias se tengan en cuenta (con éstos parámetros valiendo 0).

$$n_0 \leq \frac{\left(\sum_b P^*\left(b/a\right)\right)^2}{\sum_b P^*\left(b/a\right)^2} \leq N$$

*(Fórmula 3.4.4.1)*

Siendo $n_0$ el número de instancias en la menor distancia del atributo considerado, y N el número de instancias total. Así, para los parámetros que estamos estudiando, se selecciona un valor de entre $n_0$ y N, y se invierte. Éste proceso da como resultado lo que en Weka se llama “global blend” o parámetro de mezclado.

#### 3.4.5. Clasificación de las Instancias

Para calcular la probabilidad de una instancia “a” de pertenecer a una categoría “C”, de acuerdo a lo expuesto anteriormente, se sumarán las probabilidades de que los valores de atributos de la instancia “a” se transformen en los valores de atributo de las instancias que pertenecen a la clase “C”:

$$P^*(C/a) = \sum_{b \in C} P^*(b/a)$$

*(Fórmula 3.4.5.1)*

Así, se calcula esto para cada clase, y la que dé mayor probabilidad será la que se asigne a la instancia.

## 3.5. Simple Logistic

Niels Landwehr, Mark Hall y Frank Eibe, publicaron en 2005 un documento en el que hablaban de árboles con una estructura usual, pero con la diferencia de que la predicción de clase que daban sus hojas se modelizaba mediante una implementación especial de la regresión logística, que combinaba ésta técnica estadística con un algoritmo llamado LogitBoost. Así pues, para explicar en qué consiste ésta implementación, se requiere la introducción de ambos conceptos.

#### 3.5.1. Regresión Logística

La regresión logística, al igual que la lineal, [6] es un análisis estadístico que trata de predecir el valor de una variable, en función de otras llamadas variables predictoras. La regresión lineal se encarga de hacerlo para variables numéricas mediante una función lineal (recta de regresión), y la logística hace lo propio si la variable a predecir es categórica. Es especialmente útil cuando la variable a predecir es dicotómica, es decir, toma sólo dos valores.

#### 3.5.1.1. Modelo de Regresión Logística Binaria

Para realizar ésta clasificación, se requiere de una función discriminante. En éste caso, se usa una función llamada Logit.

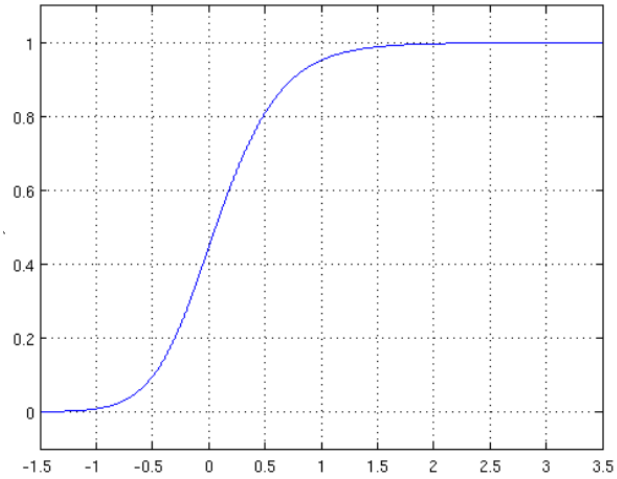



**Ilustración 21: Función Logit**

En la ilustración 21, se puede apreciar que, debido a la forma de la función, es perfecta para discriminar entre valores que tomen valor 0 o 1. Así, en el eje x se encuentra el valor de la variable predictora, y en el eje "y", la probabilidad de pertenencia a la clase, expresada mediante la función ya mencionada. Siendo la probabilidad de pertenencia a la clase:

$$P(Y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\cdots\beta_n x_n)}}$$

*(Fórmula 3.5.1.1.1)*

Así, se define la función Logit:

$$Logit(P) = \ln(P) = \beta_0 + \beta_1 x_1 + \cdots \beta_n x_n$$

*(Fórmula 3.5.1.1.2)*

Como se puede ver, la probabilidad de pertenencia a la clase, pertenece de los valores que toman las variables predictoras, y de unos ciertos valores β.

Éstos valores son lo que se llama LogOdd, que no es más que el logaritmo neperiano de los odds.

$$Odd = \frac{P(Y=1)}{1-P(Y=1)}$$

*(Fórmula 3.5.1.1.3)*

Como se puede ver en la fórmula 3.5.1.1.3, los odds [6] reflejan cuántas veces es más probable la pertenencia a la clase respecto a la no pertenencia. Éstos son muy usados, por ejemplo, en el mundo de las apuestas deportivas, cuando se da una cuota de, por ejemplo 2 a 1 en un partido de fútbol, que quiere decir que hay dos veces más probabilidad de que un cierto equipo gane un partido que de que lo pierda. O, dicho de otro modo, el equipo tiene una probabilidad de un 66% de ganar el partido.

Se puede ver fácilmente que, si la probabilidad de no pertenencia es mayor que la de pertenencia, éste odd o ratio, estará contenido en un intervalo entre 0 y 1, mientras que, si la probabilidad de pertenencia es mayor, el valor del odd está entre 1 e infinito. Es por esto que, para la función, se toma logaritmo neperiano. De ese modo, ahora los odds son simétricos: para probabilidades de no pertenencia mayores, éste LogOdd irá de 0 a menos infinito, y viceversa. Así que, lo que representan éstas β es la relación que las variables predictoras tendrán sobre la variable de estudio. De éste modo, si las β son positivas, esto quiere decir que, si la variable predictora aumenta, aumenta también la probabilidad de pertenencia a la clase (Y=1 es el grupo de interés), y viceversa.

Éstos odds también se usan para comparar dos atributos. Es decir, se puede expresar cuántas veces aumenta la probabilidad de pertenencia a la clase (Y=1) el atributo B si lo comparo con otro cierto atributo A, o, dicho de otro modo, qué tan grande es la probabilidad de B respecto a la de A. Cuando la variable es numérica, representa la probabilidad de pertenencia a la clase cuando se pasa de un valor de atributo a otro, permaneciendo constantes el resto de las variables.

Una duda razonable es acerca [7] de cómo calcular éstos valores, ya que son éstos los que ajustan la posición del modelo logístico. Para ello, se utiliza el criterio de

máxima verosimilitud. Ésta posición será la que maximice la probabilidad de observación de los casos de entrenamiento.

Para ello, se siguen los siguientes pasos:

1- Usando la escala logarítmica, se proyectan los datos originales sobre la recta candidata (logit). Esto dará a cada instancia un LogOdd.
2- Se calculan las probabilidades asociadas a éstos LogOdd para la representación de la función discriminante, similar a la vista en la ilustración 3.5.1.1.1.
3- Ahora, éstas probabilidades se comparan con los datos reales para hallar su verosimilitud. Ésta será la probabilidad hallada en el paso 2, para los ejemplos positivos, y uno menos ésa probabilidad para los ejemplos negativos, ya que la verosimilitud (Likelihood o L a partir de ahora en las fórmulas) es el valor del eje “y” donde el punto interseca la curva.

$$L(casos\ positivos) = \prod_{Y=1} p_i$$

*(Fórmula 3.5.1.1.4)*

$$L(casos\ negativos) = \prod_{Y=0} (1 - p_j)$$

*(Fórmula 3.5.1.1.5)*

$$L(total) = \prod_{Y=1} p_i * \prod_{Y=0} (1 - p_j)$$

*(Fórmula 3.5.1.1.6)*

En éstos casos, es más común trabajar con el logaritmo de la verosimilitud, con lo que quedaría la siguiente expresión:

$$Log(L) = \sum_{Y=1} \log(p_i) + \sum_{Y=0} \log(1 - p_j)$$

*(Fórmula 3.5.1.1.7)*

O, de otra manera:

$$Log(L) = \sum_{i=1}^{n} y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)$$

*(Fórmula 3.5.1.1.8)*

Siendo $y_i$, los resultados conocidos, y $p_i$ las estimaciones, desde 1 hasta n muestras del set de datos.

Una vez realizado esto, se rota la línea y se empieza de nuevo. Esto se hace de forma recursiva hasta que se encuentra la línea que hace que la verosimilitud sea máxima. En Weka, ésta búsqueda se hace mediante un método llamado “Quasi-Newton”, que es un algoritmo que no necesita el uso de la matriz jacobiana para calcular máximos y mínimos, disminuyendo en gran medida el coste computacional de la operación.

Si se busca información en la página de Weka acerca de la regresión logística, se verá que la fórmula del logaritmo de la verosimilitud está en negativo. Esto es porque éste estimador se está usando como función de pérdida, donde los valores más pequeños representan los modelos que mejor se ajustan a los datos. Nótese, que si antes, con signo positivo, lo que se busca es que el logaritmo de la verosimilitud sea máximo, si se pone un menos delante, se buscará que sea mínimo.

$$Log(L) = -\sum_{i=1}^{n} y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)$$

*(Fórmula 3.5.1.1.9)*

La implementación de la regresión logística en Weka, incluye un estimador llamado “ridge” [8]. El efecto de éste es desajustar la función discriminante con respecto a los datos de entrenamiento, para evitar un indeseado sobreajuste.

$$Log(L)^R = \sum_{i=1}^{n} y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i) + Ridge * \sum_{j=1}^{p} \beta_j^2$$

*(Fórmula 3.5.1.1.10)*

Siendo j=1…p el número de atributos. Nótese que, como se ha dicho con anterioridad, $p_i$ es $x_i * \beta_i$, la predicción realizada por el modelo.

Así, si el parámetro vale 0, la estimación es la ordinaria. Éste parámetro influye en la sensibilidad a la hora de clasificar una instancia para la clase perteneciente a Y=1, ya que lo que hace es aumentar o disminuir la pendiente de la función discriminante.

### 3.5.2. LogitBoost

Un algoritmo booster es aquel [9] que combina un conjunto de lo que se llama "clasificadores débiles" (algoritmo que produce una clasificación, con mayor probabilidad de ser significativamente mejor que el azar), para construir un clasificador fuerte. Cada clasificador débil creado tiene en cuenta el error de su predecesor, y no todos tienen el mismo peso a la hora de decidir la clasificación de una instancia. El caso más conocido de algoritmo booster es el AdaBoost (ver ilustración 22). Éste algoritmo, utiliza usualmente "stumps" (árboles de un nodo y dos hojas) como clasificadores débiles. Éstos stumps están ponderados según su importancia a la hora de clasificar una instancia nueva, ya que se decide la clase mediante la "votación" de todos éstos clasificadores (cada uno hace una predicción, que contará para la decisión final en base a su peso).

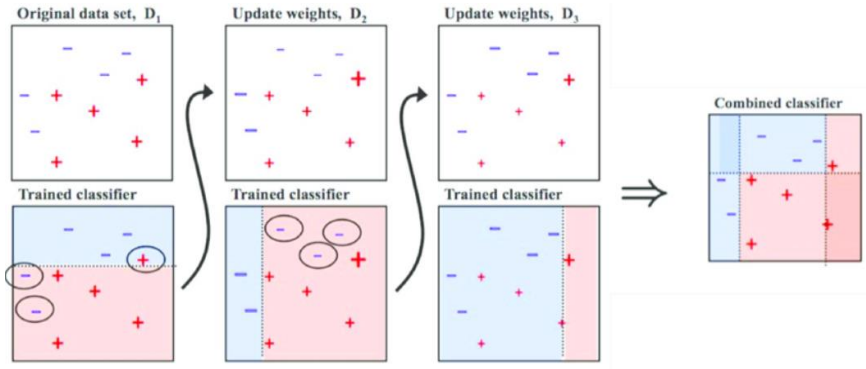



**Ilustración 22: AdaBoost**

LogitBoost sigue los pasos de Adaboost, pero aplicando la función de pérdida de la regresión logística. Los pasos que sigue el algoritmo LogitBoost [7] se indican a continuación.

#### 3.5.2.1. Ponderación del set de datos

Se empieza, al igual que en AdaBoost, dando pesos iguales a las instancias del set de datos. Y estableciendo los LogOdds iniciales a 0. Esto es análogo a decir que todos los sucesos del set de datos son equiprobables.

$$w_i = \frac{1}{N} \quad para\ i = 1 \dots N$$

*(Fórmula 3.5.2.1.1)*

$$F_0(x_i) = 0 = p(x_i) = \frac{1}{2}$$

*(Fórmula 3.5.2.1.2)*

#### 3.5.2.2. Creación de los M modelos

La segunda parte de éste algoritmo consta de un bucle, que va desde m (modelo actual) =1 hasta M (número de modelos creados, o vueltas que da el algoritmo LogitBoost). Nótese que, el bucle quedaría del siguiente modo: for (m=1; m=M; m++).

Lo primero será calcular las respuestas del modelo actual.

$$z_j = \frac{y_i - p(x_i)}{p(x_i) * (1 - p(x_i))}$$

*(Fórmula 3.5.2.2.1)*

Y los nuevos pesos:

$$w_i = p(x_i) * (1 - p(x_i))$$

*(Fórmula 3.5.2.2.2)*

Antes de continuar, se hace necesario explicar de dónde ha venido ésta fórmula. Esto es resultado de optimizar la función de pérdidas (logaritmo de la verosimilitud) mediante gradient boosting:

$$r_{im} = -\left[\frac{\delta L\left(y_i, F(x_i)\right)}{\delta F(x_i)}\right]_{F(x)=F_{m-1}(x)} \qquad para\ i = 1 \dots n$$

*(Fórmula 3.5.2.2.3)*

Donde $r_{im}$ son pseudo-residuos de cada muestra, así que se calculan como la diferencia entre el valor real y el de la predicción. El siguiente paso es construir un árbol de regresión para predecir éstos $r_{im}$ para, acto seguido, crear las regiones clasificatorias del modelo actual (habitualmente, LogitBoost trabaja con stumps así que éstas regiones serán las hojas). Una vez se ha llegado a éste paso, se procede a calcular la salida de cada región clasificadora, optimizando la función de pérdida, es decir, la salida de cada hoja en un árbol creado por ésta técnica, será el valor de salida que minimice la función de pérdidas (para cada hoja, se tiene en cuenta las instancias que forman parte de ella):

$$z_{jm} = \underset{z}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + z)$$

*(Fórmula 3.5.2.2.4)*

Y sustituyendo en la fórmula 3.5.2.2.3 la función de pérdidas usada:

$$z_{jm} = \underset{z}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} -y_i[F_{m-1}(x_i) + z] + \ln(1 + e^{F_{m-1}(x_i)+z})$$

*(Fórmula 3.5.2.2.5)*

Como se puede ver, la función de pérdidas tiene en cuenta el resultado del modelo anterior. Así, se tiene que optimizar el resultado actual mediante el valor de z que haga mínima la función.

Para abordar el problema de la optimización de la fórmula 3.5.2.2.5, en vez de derivar respecto a “z”, se aproxima mediante el polinomio de Taylor. Esto daría como resultado la siguiente expresión:

$$z = \frac{-\frac{d}{dF}(y_i, F_{m-1}(x_i))}{\frac{d^2(y_i, F_{m-1}(x_i))}{dF}}$$

*(Fórmula 3.5.2.2.6)*

Nótese que esto ya se ha calculado antes, y es que el dividendo es el residuo como dice la ecuación 3.5.2.2.3. Así que, de manera análoga, sabiendo que, al fin y al cabo, derivar con respecto a F es derivar respecto a LogOdds, y que las probabilidades se pueden expresar en su forma neperiana, se llegaría a la fórmula 3.5.2.2.1.

Una vez se tiene ajustado el modelo $f_m(x)$ como se ha dicho con anterioridad usando los pesos y las variables independientes, se actualiza el modelo aditivo:

$$F(x) \leftarrow F(x) + \frac{1}{2} * f_m(x)$$

*(Fórmula 3.5.2.2.7)*

$$p(x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}$$

*(Fórmula 3.5.2.2.8)*

#### 3.5.2.3. Clasificador de Salida

Una vez se ha completado el bucle, cuando se han creado ya todos los modelos, el clasificador de salida es el siguiente:

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

*(Fórmula 3.5.2.3.1)*

Así, una instancia se calcula calculando la probabilidad de pertenencia a la clase Y=1 con la siguiente fórmula:

$$p(Y=1)=\frac{1}{1+e^{-\sum_{m=1}^{M} f_m(x)}}$$

*(Fórmula 3.5.2.3.2)*

Como se puede apreciar, lo que se calcula con éste algoritmo, es el logit de una regresión logística, de ahí su nombre.

### 3.5.3. Simple Logistic

La idea subyacente de éste método [10] es seleccionar los atributos más relevantes en la regresión, para evitar los atributos que no incrementen la capacidad predictiva de la misma. Para esto se usa el algoritmo LogitBoost, de forma que a cada iteración introduce un nuevo atributo.

Para ello, se construye un modelo de regresión para cada atributo y se ajusta ésta al set de datos. Se seleccionará a cada vuelta el atributo que de menor error. En ésta selección se puede dar que el mismo atributo se incluya varias veces.

Así, este algoritmo puede tener la función de seleccionar los atributos más relevantes si se hace que pare antes de converger, o de clasificador si se deja ejecutar hasta que no mejore la precisión predictiva sobre instancias sin clasificar. Para saber cuándo parar en tiempo de entrenamiento, se suele utilizar la técnica llamada validación cruzada. En ella, el set de entrenamiento se divide en K subconjuntos en el que un subconjunto se utiliza como set de test y el resto como set de entrenamiento, repitiéndose ésta operación K veces cambiándose el set de prueba cada vez.

Una ventaja respecto al modelo logístico tradicional es que en éste algoritmo se escogen los atributos más relevantes, con lo cual habrá menos ruido y menos probabilidad de sobreajuste.

#### 3.5.3.1. Atributos Nominales

Como LogitBoost sólo puede actuar con valores de atributo numérico, los atributos nominales o categóricos se transforman en un código binario con “n” dígitos, tantos como valores distintos del atributo hay, donde todos serán cero menos la posición que pertenezca al valor de atributo actual.

#### 3.5.3.2. Valores Perdidos

Como LogitBoost no puede tratar con valores de atributos perdidos, al no poder ajustar la función de regresión, éste valor perdido se sustituye por la media (atributos numéricos) o la moda (atributos nominales) de éstos atributos.

## 3.6. Máquinas de vector Soporte

El objetivo de éste algoritmo es construir un hiperplano para separar las clases del set de datos. Un hiperplano es un subespacio de una dimensión menor que el espacio que lo contiene, por ejemplo, en el plano ($V^2$) un hiperplano es un subespacio de dimensión uno, es decir, una recta.

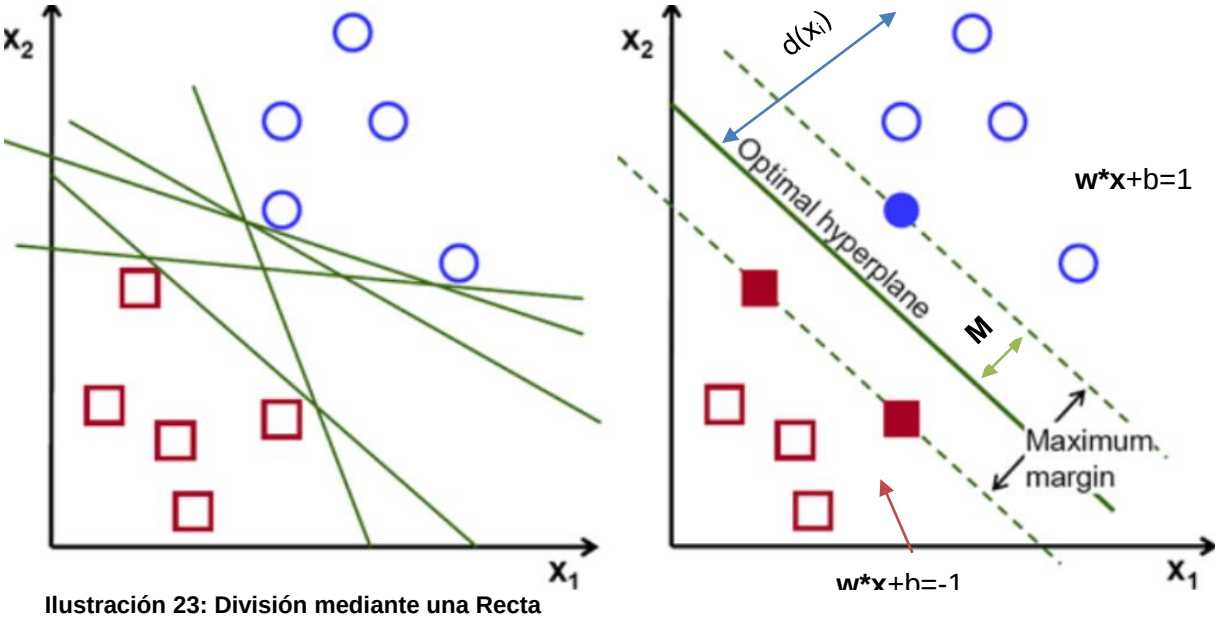



**Ilustración 23: División mediante una Recta**

Con objetivo de que la división sea lo más clara posible, y que al clasificar una instancia nueva no se produzcan falsos positivos, la distancia de éste hiperplano a los puntos más cercanos del set de datos se maximizará, lo que dará lugar al hiperplano óptimo.

Como se puede ver en la ilustración 23, hay dos instancias dentro del margen (líneas discontinuas): Esos son los vectores soporte del hiperplano óptimo.

### 3.6.1. Clases linealmente separables

Para el cálculo del hiperplano óptimo, lo primero que se habrá de realizar es calcular la región mínima que contiene al conjunto. Esta región será la envolvente convexa.

$$f(x) = \bar{w} * \bar{x} + b,$$

$$Restricciones: \begin{cases} \bar{w} * \bar{x} + b \geq 1 \ , & x \in clase \ + \\ \bar{w} * \bar{x} + b \leq -1, & x \in clase \ - \end{cases}$$

*(Fórmula 3.6.1.2)*

Las restricciones se pueden agrupar en:

$$\bar{y} * (\bar{w} * \bar{x} + b) - 1 \geq 0$$

*(Fórmula 3.6.1.2)*

Siendo **w** el vector de pesos, y **x,** los vectores de atributos de las instancias. Nótese que las restricciones son las envolventes convexas que engloban los casos de las clases por separado. Lo que se busca el segmento más corto que une ambas envolventes con objetivo de hallar los casos más cercanos a la región factible. El hiperplano óptimo será perpendicular a éste y equidistante a ambos extremos (el hiperplano será perpendicular a **w**). Así, la distancia entre un cierto caso próximo y el hiperplano será:

Siendo la distancia a cualquier instancia $x_i$:

$$d = \frac{f(x_i)}{||\boldsymbol{w}||}$$

*(Fórmula 3.6.1.3)*

Se concluye que para aumentar el margen se tiene que disminuir la norma de éste vector **w**:

$$M = \frac{1}{||\boldsymbol{w}||}$$

$$\min f(w) = \frac{1}{2} * \left|\left|\boldsymbol{w}\right|\right|^2$$

*(Fórmula 3.6.1.4)*

Por tanto, habrá que minimizar ||**w**|| con las restricciones previamente vistas para encontrar el hiperplano. Éste problema se resuelve en la práctica mediante la teoría de la optimización, con la Lagrangiana, para formular un problema de optimización dual a partir del problema de optimización primal, que es el que se ha planteado en la fórmula 3.6.1.3. Por tanto, ahora el problema queda formulado de la siguiente manera:

$$\max L(\alpha) = \sum_{i=1}^{n} \alpha_i \ - \frac{1}{2} * \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j$$

$$Restricción: \sum_{i=1}^{n} \alpha_i y_i = 0$$

*(Fórmula 3.6.1.5)*

Siendo α los multiplicadores de Lagrange, que tienen que maximizar la función. Una vez resuelto esto, se puede solucionar el problema inicial, sabiendo que:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

*(Fórmula 3.6.1.6)*

Junto a todo lo anteriormente expuesto, a saber, condiciones de margen y restricciones. Así, la función buscada queda de la siguiente manera:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i * (\overline{\mathrm{w}} * \bar{\mathrm{x}}) + b$$

*(Fórmula 3.6.1.7)*

Así pues, el último paso será calcular el parámetro b (el sesgo). Para ello, y sabiendo que sólo las instancias en las que $\alpha_i > 0$ son los vectores soporte (sólo éstos son los usados para la construcción del hiperplano), éste parámetro se construye calculando la media sobre éstos vectores soporte:

$$b = \frac{1}{k} * \sum_{i=1}^{k} \left( y_k - (\overline{w} * \overline{x_k}) \right) \quad para\ k\ vectores\ soporte$$

*(Fórmula 3.6.1.8)*

Pero esto sólo se puede realizar, como se puede ver, cuando el problema es linealmente separable. Cuando esto no es así, se tiene que recurrir a los kernels.

### 3.6.2. Corrección para casos Cuasi Linealmente Separables

Nótese que, hasta ahora, se ha supuesto que las clases son linealmente separables, bien en el espacio original, o bien en el espacio de alta dimensionalidad que el método de los kernels ofrece. Pero, cuando esto no ocurre, se introduce un parámetro de complejidad c.

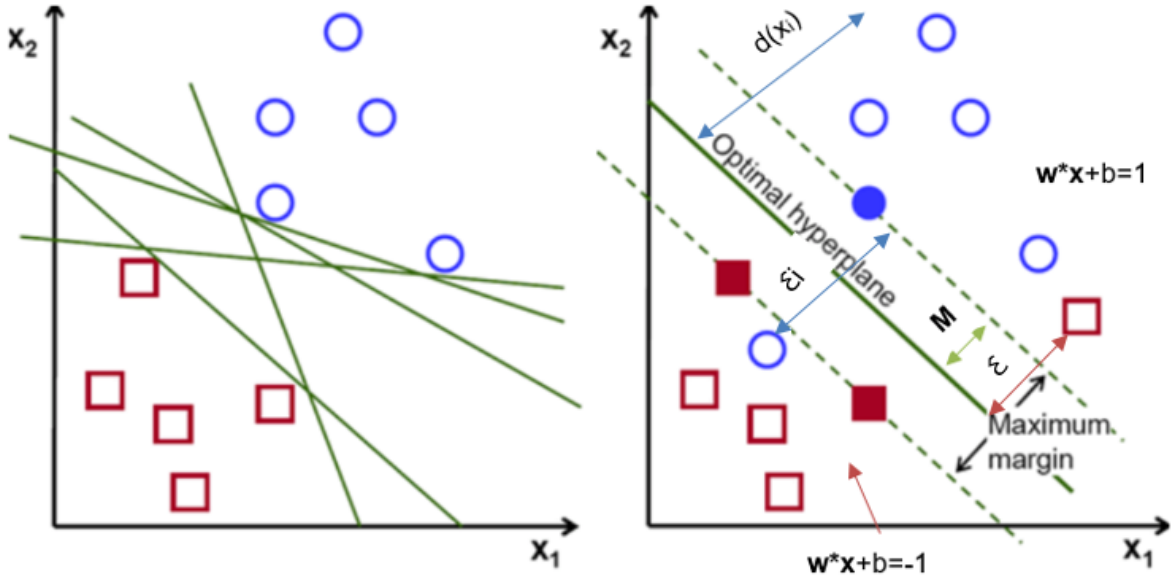



**Ilustración 24: casos linealmente no separables**

Como se puede observar en la ilustración 24, hay instancias que no han sido debidamente separadas por clases, y éstas tienen asociado un parámetro de holgura, que es la distancia de éstas a la función lineal de separación de su clase (la holgura será 0 para los casos separables). Así el problema de optimización debe ser un

compromiso entre el máximo margen y los menores errores de clasificación, dando como resultado lo que se llama hiperplano de margen suave:

$$\min f(w) = \frac{1}{2} * \left|\left|\boldsymbol{w}\right|\right|^2 + C * \sum_{i=1}^{n} \xi_i$$

$$Restricción: \bar{y} * (\bar{\mathrm{w}} * \bar{\mathrm{x}} + \mathrm{b}) - 1 + \bar{\xi} \geq 0$$

*(Fórmula 3.6.3.1)*

El parámetro C de complejidad será el encargado de permitir mayores o menores errores de clasificación, por lo que tendrá relación directa sobre un posible sobreajuste o su contrario. Así, para valores altos de C, se permiten pocos errores mientras que, para valores bajos, se estarán permitiendo muchos errores de clasificación.

Una vez formulado el problema de optimización primal, el problema de optimización dual queda de la siguiente manera:

$$\max L(\alpha) = \sum_{i=1}^{n} \alpha_i \ - \frac{1}{2} * \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_j, x_i)$$

$$Restricción: \sum_{i=1}^{n} \alpha_i y_i = 0 \ , con\ 0 < \alpha_i < C$$

*(Fórmula 3.6.3.2)*

El cálculo del parámetro de sesgo se realiza de manera análoga al caso en el que el set de datos es linealmente separable, pero teniendo ahora en cuenta que las instancias que se elijan como vectores soporte tienen que cumplir con la restricción dada en la fórmula 3.6.3.2. A éstos se les llama vectores soporte acotados.

### 3.6.3. Clases no separables Linealmente

La idea clave aquí es que, si bien las clases no son linealmente separables, existe un espacio de alta dimensionalidad en el cual sí lo son.

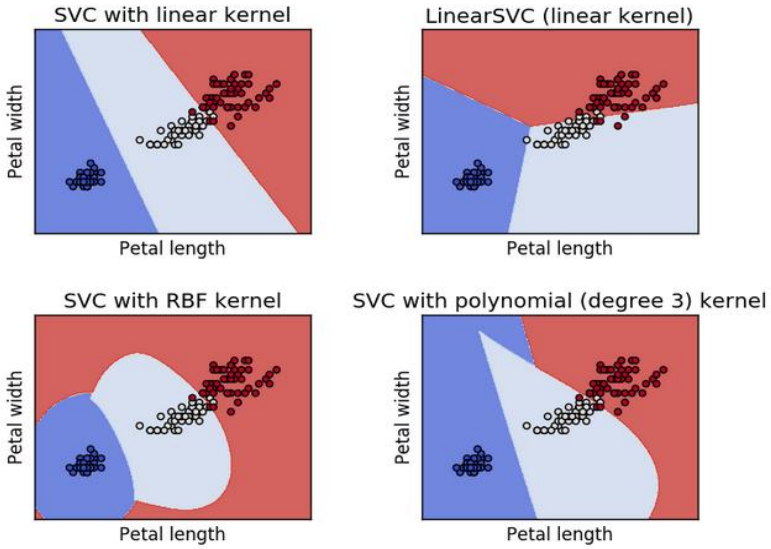



**Ilustración 25: Clases separadas por distintos kernels**

Para ello, lo primero que se habrá de hacer es realizar una transformación sobre las instancias que las lleve a ése plano de alta dimensionalidad. Esto se realiza mediante una función llamada kernel (ver ilustración 25). Así, sea Φ el conjunto de funciones de transformación que lleva a los vectores de atributos $\mathbf{x_i}$ al espacio de alta dimensionalidad, la ecuación del hiperplano quedará como sigue:

$$f(x) = (w_1 * \Phi_1(x) + \cdots + w_n \Phi_n(x))$$

*(Fórmula 3.6.2.1)*

Se define la función de Kernel como aquella que asigna a cada par de vectores de atributos de entrada, un número real, que es el que resulta del producto escalar de la transformada de éstos vectores.

$$K(x_1, x_2) = (\Phi_1(x_1) * \Phi_1(x_2) + \cdots \Phi_m(x_1) * \Phi_m(x_2))$$

*(Fórmula 3.6.2.2)*

Por lo que, una vez transformados los vectores, se puede proceder de manera análoga a lo visto con anterioridad, y así, el problema de optimización quedaría como sigue:

$$\max L(\alpha) = \sum_{i=1}^{n} \alpha_i \ - \frac{1}{2} * \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_j, x_i)$$

$$Restricción: \sum_{i=1}^{n} \alpha_i y_i = 0 \, , con \ \alpha_i \geq 0$$

*(Fórmula 3.6.2.3)*

Estos kernels tienen que ser funciones semidefinidas positivas y simétricas, para cumplir el teorema de Mercer y así existirá una función Φ: $R^n \rightarrow \Omega$, siendo Ω un espacio de Hilbert (espacio de n dimensiones en el que se cumplen las reglas de la geometría euclídea), que será el espacio de dimensionalidad alta en el que las instancias serán separables, tal que cumplirá lo siguiente:

$$K(x_i, x_j) = \phi(x_i) * \phi(x_j)$$

*(Fórmula 3.6.2.4)*

De ello se deduce que no se necesita conocer Φ, sólo se necesitará conocer ésta función kernel que cumpla con el teorema de Mercer.

Un ejemplo de éste tipo de funciones será el kernel polinomial:

$$K(x_i, x_j) = (x_i * x_j + 1)^k$$

*(Fórmula 3.6.2.4)*

### 3.6.4. Mínima Optimización Secuencial (SMO)

El cálculo del hiperplano requiere programación cuadrática, al ser fórmula 3.6.2.3 una función cuadrática. Pero esto se complica si el problema clasificatorio tiene alta dimensionalidad. Para ello, John Platt en 1998 inventó un algoritmo que resuelve éste problema.

Éste algoritmo [11] divide el problema original en subconjuntos, cogiendo dos multiplicadores de Lagrange ($\alpha_i$) a cada etapa para su optimización, lo que simplifica enormemente el problema de programación cuadrática inicial.

Ésta optimización se realiza de forma analítica con las restricciones anteriormente vistas, calculándose primero un multiplicador y con éste, calculándose el otro:

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2 * ((f^{old}(x_1) - y_1) - (f^{old}(x_2) - y_2))}{2 * K(\overline{x_1}, \overline{x_2}) - K(\overline{x_1}, \overline{x_1}) - K(\overline{x_2}, \overline{x_2})}$$

$$E_1 - E_1 = error = (f^{old}(x_1) - y_1) - (f^{old}(x_2) - y_2)$$

*(Fórmula 3.6.4.1)*

Una vez hallado esto, se aplican las restricciones:

$$\alpha_2^{new} = \begin{cases} H & si\ \alpha_2^{new} \geq H \\ \alpha_2^{new} & si\ L < \alpha_2^{new} < H \\ L & si\ \alpha_2^{new} \leq L \end{cases}$$

*(Fórmula 3.6.4.2)*

Donde, H y L son:

$$H = \begin{cases} \min(C, C + \alpha_2^{old} - \alpha_1^{old}) & si\ y_1 \neq y_2 \\ \min(C, \alpha_2^{old} + \alpha_1^{old}) & si\ y_1 = y_2 \end{cases}$$

$$L = \begin{cases} \max(0, \alpha_2^{old} - \alpha_1^{old}) & si\ y_1 \neq y_2 \\ \max(0, \alpha_2^{old} + \alpha_1^{old} + C) & si\ y_1 = y_2 \end{cases}$$

*(Fórmula 3.6.4.3)*

Calculado ya uno de los multiplicadores, el otro se halla con la siguiente fórmula:

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 * (\alpha_2^{old} - \alpha_2^{new})$$

*(Fórmula 3.6.4.3)*

Los multiplicadores se seleccionan de la siguiente manera: para el primer multiplicador, el algoritmo itera sobre el set de datos, quitando los ejemplos frontera. Todo multiplicador que viole las restricciones serán candidatos para su optimización. Una vez que se elija éste primer multiplicador, el segundo se elige de tal manera que el error sea máximo, y se aplica la optimización.

Acabada la optimización de éstos dos primeros multiplicadores, se actualizan sobre el set de datos y empieza la segunda iteración del algoritmo. Esto se repite hasta que todos los multiplicadores cumplen las restricciones, momento en el cual se miran los ejemplos de frontera para ver si ahora que todos los demás han sido optimizados, éstos no se han quedado fuera de las restricciones.

Una vez todos los multiplicadores cumplen las restricciones, el algoritmo termina, habiendo creado ya el clasificador.

#### 3.6.5. Consideraciones Finales

Éste algoritmo trabaja con variables numéricas, por lo que, para trabajar con variables categóricas, se tendrá que hacer una conversión binaria de éstas. Pese a ello, éste método es muy eficaz, aunque el set de datos posea pocos atributos, y, al contrario del perceptrón multicapa, el mínimo optimizado es único al tratar con una función convexa. A su vez, éstas son muy resistentes al sobreajuste, por lo que su capacidad de generalización es muy buena.

## 3.7. Perceptrón multicapa

Las redes neuronales son [12] algoritmos inspirados en el funcionamiento del cerebro humano. Su unidad básica de procesamiento es la neurona, representada en la ilustración 26.

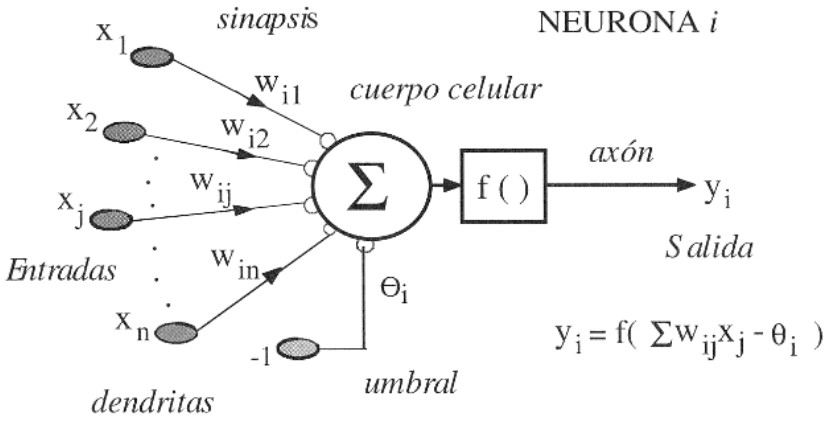



**Ilustración 26: Modelo Neurona**

La neurona [12] consta de las siguientes partes:

- Entradas: Las entradas a la neurona correspondería a la sinapsis en el símil biológico, y como se puede ver, éstas están ponderadas con unos ciertos pesos $w_{ij}$, que representan la fuerza de la conexión simpática.
- Regla de propagación: el cuerpo celular de la neurona será el sumatorio de las entradas ponderadas a ésta, menos una cierta cantidad llamada umbral.
- Función de activación: ésta es la salida o estado de activación de la neurona, es decir, su axón.

### 3.7.1. Perceptrón Simple

Éste modelo [12][13] fue ideado por Rosenblatt en 1962. Éste estaba formado por una capa de neuronas de entrada y otra de salida, y da solución a problemas de clasificación lineales mediante la creación de un hiperplano divisor.

La capa de entrada está formada por los atributos del set de datos, cada uno de ellos ponderados mediante su correspondiente peso, y la capa de salida conforma la decisión de clasificación del algoritmo, mediante una función de tipo escalón (ver iliustración 27).

Para la creación de éste hiperplano divisor, formado por la función de salida de

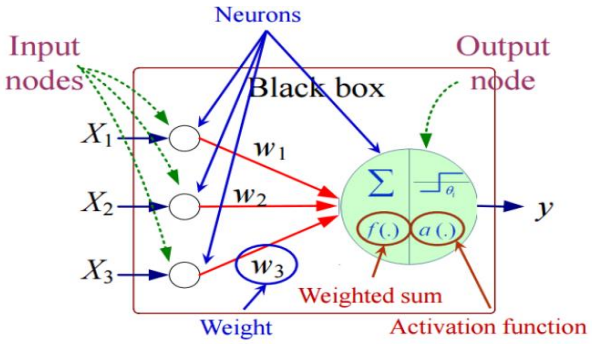



**Ilustración 27: Perceptrón**

cada neurona de la última capa, se habrá de ponderar cada entrada de tal forma que el set de datos quede dividido según su clase. Por tanto, siendo $y_j$ la señal de una de las m neuronas de salida:

$$y_j = a(\sum_{i=1}^{n} w_i x_i - u) = a(f(w) - u)$$

*(Fórmula 3.7.1.1)*

El hiperplano separador para una neurona tendrá la siguiente forma:

$$0 = w_1 x_1 + \cdots w_i x_i - u$$

*(Fórmula 3.7.1.2)*

A modo de ejemplo, si el perceptrón constara de dos entradas, y se tratase de discernir entre dos clases, el hiperplano divisor tendría la forma que se puede ver en la ilustración 28.

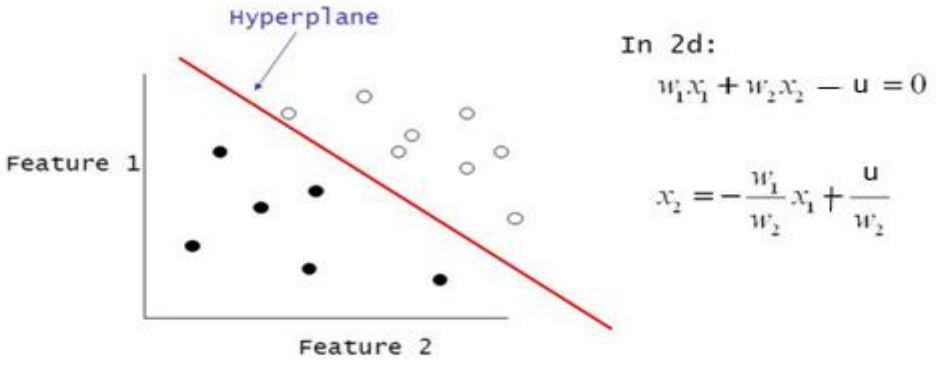



**Ilustración 28: Hiperplano en 2D**

Ésta superficie divisora se ajusta del siguiente modo:

- Primero se establecen de forma aleatoria los pesos y el umbral. Éste puede ser representado como una entrada más, pero con un valor constante sin ponderar.
- Una vez realizado esto, se comprueban las salidas que darían las instancias de entrenamiento con ese ajuste. Ésta salida se calcula con la función escalón correspondiente:

$$y(w,u) = \begin{cases} 1, & si\ w_1x_1 + \cdots w_ix_i \geq u \\ 0, & si\ w_1x_1 + \cdots w_ix_i < u \end{cases}$$

*(Fórmula 3.7.1.3)*

- Una vez realizada la clasificación, se comprueba si ésta es correcta, y si no lo es, se modifican los pesos del modo siguiente:

$$w_{i,j}^{new} = \alpha(Cl(x_i) - y_j) * x_i + w_{i,j}^{new}$$

$$u_{new} = (Cl(x_i) - y_j) + u_{old}$$

*(Fórmula 3.7.1.4)*

  El cálculo del peso, por tanto, se realiza teniendo en cuenta el error cometido, y un cierto parámetro α que representa la razón de aprendizaje, y hace que el algoritmo aprenda de forma más lenta, si el valor de éste es bajo, por lo que podría implicar largos periodos de aprendizaje; pero también, si el valor es alto, puede hacer que el algoritmo oscile. Éste va entre 0 y 1.

El entrenamiento se realiza hasta que, si el set de entrenamiento es linealmente separable, no hay errores de clasificación, o si no lo es, éste oscilará de forma que no convergerá.

Como se puede deducir a partir de todo lo anteriormente expuesto, éste algoritmo tiene una importante desventaja en el hecho de limitarse únicamente a clasificaciones donde las clases son linealmente separables. Para salvar ésta desventaja, en 1965 se implementó el perceptrón multicapa.

### 3.7.2. Perceptrón Multicapa

Éste algoritmo [12] [13] es el resultado de generalizar el perceptrón. Éste consta de una capa de entrada, que serán los atributos del set de datos, tantas capas ocultas como sea considerado por el usuario, cuya entrada son las salidas de las neuronas anteriores, y una capa de salida que tendrá tantas neuronas como clases se desee clasificar (ver ilustración 29).

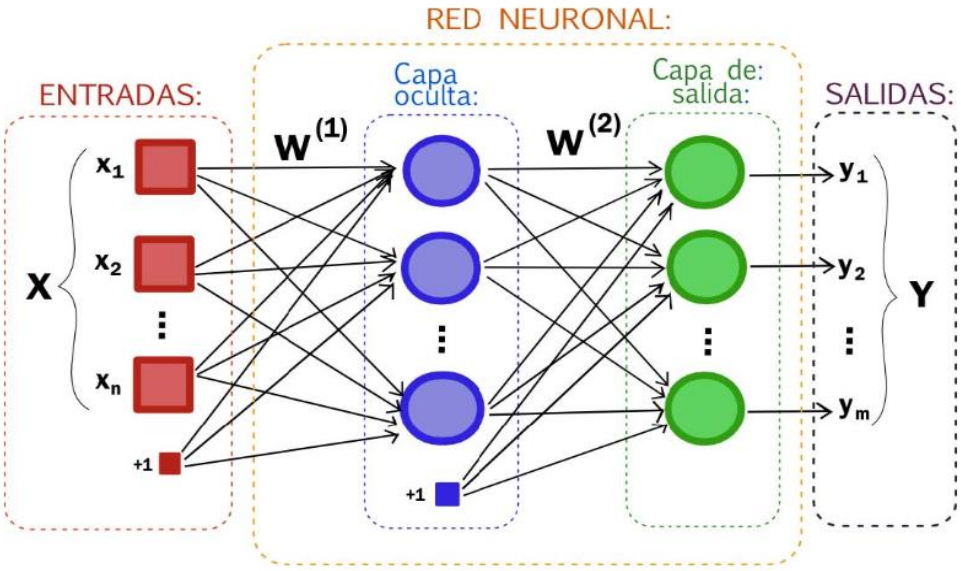



**Ilustración 29: Perceptrón Multicapa**

A través de la inclusión de éstas capas ocultas, junto con el uso como función de activación de la sigmoide, se consigue evitar el hándicap del perceptrón, pudiendo ahora resolver problemas clasificatorios fuera del ámbito lineal (ver ilustración 30).

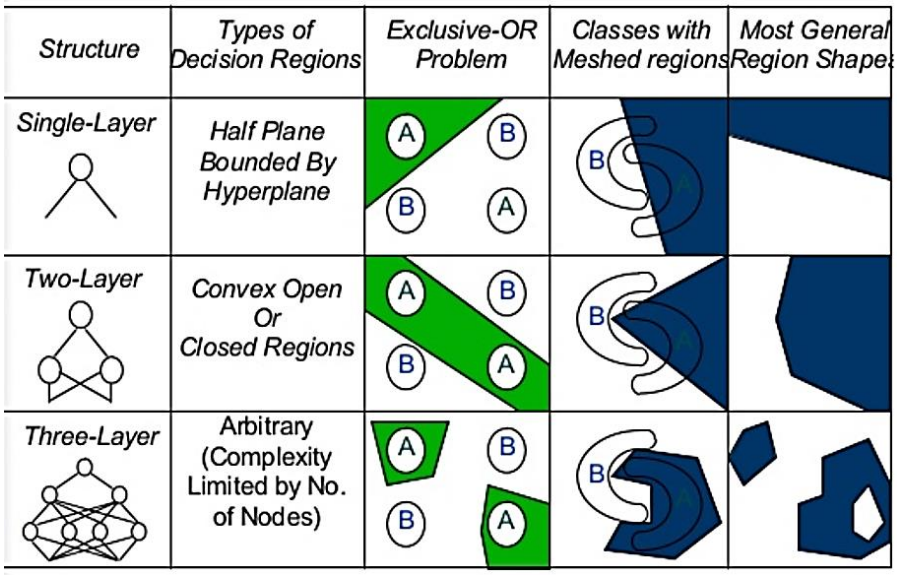



**Ilustración 30: Regiones clasificación en función de las capas**

Como se ha mencionado antes, ahora la función de activación de las neuronas de la capa oculta es una sigmoide, que tiene la forma reflejada en la ilustración 31 (es

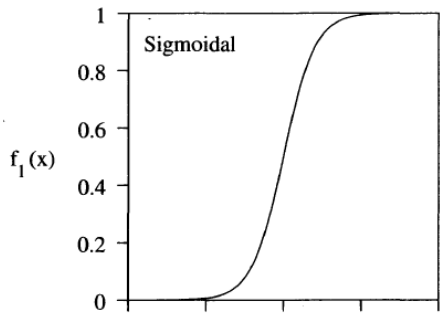



**Ilustración 31: sigmoide**

la función que también usa SimpleLogistic, como se puede ver por su forma), y cuya ecuación, que se puede ver en la fórmula 3.7.2.1, tiene una peculiaridad muy importante en el algoritmo de entrenamiento, y es que su derivada es de fácil cálculo, como se puede ver en la fórmula 3.7.2.2.

$$f(x) = \frac{1}{1 + e^{-x}}$$

*(Fórmula 3.7.2.1)*

$$f'(x) = f(x) * (1 - f(x))$$

*(Fórmula 3.7.2.2)*

Esto es muy útil ya que el algoritmo de aprendizaje usado, backpropagation, utiliza el método del descenso del gradiente para el cálculo de los nuevos pesos. Esto se basa en que la variación de los pesos se hará de la siguiente forma:

$$\Delta w = -\alpha \frac{\delta e(w_1 \dots w_n)}{\delta(w_i)}$$

*(Fórmula 3.7.2.3)*

Es decir, la variación del peso se hace en contra de la pendiente de la función (que será el error), para encontrar los mínimos de ésta. Éste método no proporciona un mínimo global, como sí que hacían las máquinas de vector soporte, sino que proporciona el mínimo local más cercano. El parámetro de aprendizaje hará que, bien

calibrado, el método no se quede oscilando entorno a un mínimo sin encontrarlo nunca.

#### 3.7.2.1. Algoritmo para el aprendizaje de la red (Backpropagation)

El aprendizaje [13] [14] de un perceptrón multicapa consta de las siguientes fases:

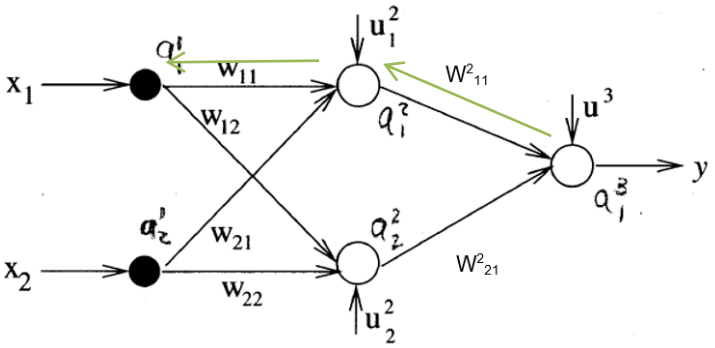



**Ilustración 32: perceptrón multicapa de ejemplo**

El primer paso, al igual que en el perceptrón, será establecer los pesos y los umbrales de forma aleatoria.

Una vez realizado esto, se calculará la respuesta del sistema a cada entrada, es decir, se realizará cada camino de la red que lleve a cada salida. Por ejemplo, la salida "y" para el perceptrón de la ilustración 32, sería:

$$y = a_1^3 = f(u^3 + a_1^2 * w_{11}^2 + a_2^2 * w_{21}^2)$$

*(Fórmula 3.7.2.1.1)*

Siendo "a", las funciones de activación de cada neurona. Las subsiguientes funciones de activación se calculan de manera análoga, y se sustituyen en la ecuación anterior. Así, calculada ya la respuesta para cada clase, se está en disposición de calcular los nuevos pesos umbrales de la red, minimizando el error cometido sobre la función de salida.

$$\frac{\delta e}{\delta y_i} = \sum_{i=1}^{n} -(Cl_i - y_i) * \frac{\delta y_i}{\delta(w,u)}$$

*(Fórmula 3.7.2.1.2)*

Para el cálculo del error, por tanto, se debe de calcular la derivada parcial de la función de salida respecto a cada peso y umbral, para ver cómo varía ésta en función de ellos y realizar la respectiva corrección. Ésta tarea se ve facilitada al ser la función de activación de las capas ocultas una sigmoide, por lo que el cálculo de las derivadas parciales se puede realizar de manera directa. Para el cálculo de la derivada parcial respecto a un peso, se sigue el camino desde la salida de la red hasta el nodo que éste pondera. Por ejemplo, para derivar respecto a $w^1_{11}$, los caminos posibles son los marcados en verde en la ilustración 32, derivando la función de activación de cada neurona en base a lo expuesto con anterioridad.

$$\frac{\delta y_i}{\delta w^1_{11}} = y_1(1-y_1) * w^2_{11} * a^2_1(1-a^2_1) * x_1$$

*(Fórmula 3.7.2.1.3)*

Así, cada derivada de [13] [15]pesos de la primera capa de una red, se puede escribir de forma genérica de la siguiente manera:

$$\frac{\delta y_i}{\delta w^1_{jk}} = x_j(a^2_k) * (1-a^2_k) * [\sum_{i=1}^{n} w_{ki} * y_i(1-y_i)]$$

*(Fórmula 3.7.2.1.4)*

Y, por tanto, el error, para cualquier peso de ésta capa $w_{jk}$, se calcula con la fórmula 3.7.2.1.5.

$$\frac{\delta e}{\delta w^1_{jk}} = x_j(a^2_k) * (1-a^2_k) * \sum_{i=1}^{outs} \left(w_{ki} y_i(1-y_i)\right) * (-(Cl_i - y_i))$$

*(Fórmula 3.7.2.1.5)*

Esto aparece en la literatura sobre el tema contraído, de la siguiente manera:

$$\frac{\delta e}{\delta w^1_{jk}} = x_j * \delta^2_k$$

*(Fórmula 3.7.2.1.5)*

La correspondiente δ de la capa subsiguiente está contenida en la capa anterior. Si se llama δ al error, esto quiere decir que el error de la neurona de la capa siguiente se tiene en cuenta en la neurona anterior, o, dicho de otro modo, el error se retropropaga a través de la red, lo que da nombre al algoritmo.

Los nuevos pesos, por tanto, se calculan con la fórmula 3.7.2.3, y los nuevos umbrales con la fórmula 3.7.2.1.6

$$\Delta u = -\alpha \frac{\delta e(w_1 \dots w_n)}{\delta u}$$

*(Fórmula 3.7.2.1.6)*

Éste proceso de cálculo de pesos y error se realiza para todas las instancias del set de entrenamiento. Una vez calculados los errores de todas las instancias, se calcula el error absoluto de la red, que será la media de los errores de las instancias. El proceso termina cuando se alcanza un error mínimo, habiéndose realizado un determinado número de ciclos.

Éste algoritmo trabaja con los valores de atributos normalizados, por lo que, si algún valor de atributo está fuera del rango entre 0 y 1, se calculará el máximo y el mínimo de ese atributo en el set de datos, y se escalará del siguiente modo:

$$Valor(x_i) = \frac{Valor(x_i) - Valor_{min}(x)}{Valor_{max}(x) - Valor_{min}(x)}$$

*(Fórmula 3.7.2.1.7)*

En Weka, a parte del parámetro de aprendizaje, se puede configurar el momento (η), que es una corrección sobre la razón de aprendizaje para evitar posibles inestabilidades debidas a ésta:

$$w_p = w_{p-1} - \alpha \frac{\delta e(w_1 \dots w_n)}{\delta(w_i)} + \eta * \Delta w_{p-1}$$

*(Fórmula 3.7.2.1.8)*

Por tanto, el momento pondera la importancia que tiene el incremento de pesos anterior.

### 3.7.3. Limitaciones

Éste algoritmo es uno de los más usados en la actualidad, aunque tiene algunas desventajas.

Una de ellas se ha comentado con anterioridad, y es la incapacidad en algunos casos de encontrar un mínimo global, cayendo en un mínimo local. Éste proceso se puede arreglar aumentando el número de neuronas de las capas ocultas, es decir, aumentando la complejidad de representación interna.

Otro problema es lo que se llama saturación, y se da cuando las entradas a una neurona tienen valores excesivamente altos o bajos. Ello lleva a que los parámetros de la red no varíen, y con ello, el error. Esto se soluciona normalizando las entradas al perceptrón.

## 3.8. OneR

Éste método [16] [17] clasificatorio fue expuesto por Holte en 1993. Se basa en la creación de una regla (OneRule), que clasifique las instancias del set de datos evaluando un solo atributo, esto es, la creación de un stump.

La metodología que se sigue para la creación de la regla es la que se va a explicar a contrinuación: Para cada atributo, se crea una regla de clasificación para cada valor de ése atributo; esto se hace, si éste es categórico, primero encontrando la clase que es más frecuente para ese valor de atributo, y asignar esa clase a la regla. Acto seguido, se calcula la tasa de error de la regla creada. Las reglas que menor tasa de error tengan serán las escogidas.

Para los atributos continuos, lo que se hace es discretizarlos, agrupando sus valores por intervalos. Esto se hace, primero ordenando los valores de atributo; acto seguido se parte el set en aquellos puntos en los que hay cambios de clase.

Con objeto de que, si hay un amplio rango de valores para un atributo, se creen subgrupos con pocas instancias, con el consiguiente sobreajuste asociado, se busca

que cada partición ahora tenga un número mínimo de instancias de la clase mayoritaria. Esto se puede ajustar en Weka, pero por defecto está establecido en 6.

Los valores perdidos se tratan como si de un valor de atributo más se tratase.

# 4. CREACIÓN DE LA HERRAMIENTA

En éste apartado, núcleo del presente trabajo, se procederá a explicar el proceso a seguir a la hora de crear una herramienta en java que, dados unos datos en bruto extraídos de unas muestras, sea capaz de clasificar éstas mediante clasificadores implementados usando las librerías de Weka.

## 4.1. Problemática Planteada

La herramienta creada tiene que dar solución a dos puntos:

- Los datos de los que se parte son mediciones de los distintos sensores que componen la nariz electrónica creada por el GRAV. En concreto, se tienen once sensores de gas resistivos y sensibles a diferentes concentraciones de

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 148 | clean | 1,5254E+12 | 21931.0 | 4186.0 | 8728.0 | 3318.0 | 13707.0 | 333.0 | | 2761.0 | 3705.0 | 22591.0 | 4922.0 | 4.582.055.390.249.480 | 5.513.084.611.276.410 | |
| 149 | clean | 1,5254E+12 | 21921.0 | 4181.0 | 8716.0 | 3313.0 | 13711.0 | 334.0 | | 2760.0 | 3693.0 | 22588.0 | 4912.0 | 4.582.055.390.249.480 | 5.572.594.796.673.530 | |
| 150 | clean | 1,5254E+12 | 21908.0 | 4175.0 | 8704.0 | 3313.0 | 13718.0 | 333.0 | | 2759.0 | 3681.0 | 22592.0 | 4909.0 | 4.582.055.390.249.480 | 5.552.758.068.207.820 | |
| 151 | clean | 1,5254E+12 | 21905.0 | 4169.0 | 8692.0 | 3310.0 | 13725.0 | 335.0 | | 2759.0 | 3669.0 | 22592.0 | 4902.0 | 45.833.905.546.654.400 | 5.571.068.894.483.860 | |
| 152 | clean | 1,5254E+12 | 21895.0 | 4164.0 | 8680.0 | 3308.0 | 13730.0 | 335.0 | | 2758.0 | 3658.0 | 22592.0 | 4896.0 | 4.582.055.390.249.480 | 5.534.447.241.931.790 | |
| 153 | clean | 1,5254E+12 | 21885.0 | 4158.0 | 8669.0 | 3306.0 | 13737.0 | 333.0 | | 2757.0 | 3647.0 | 22592.0 | 4890.0 | 45.833.905.546.654.400 | 5.534.447.241.931.790 | |
| 154 | clean | 1,5254E+12 | 21878.0 | 4154.0 | 8658.0 | 3304.0 | 13743.0 | 332.0 | | 2756.0 | 3636.0 | 22591.0 | 4884.0 | 45.833.905.546.654.400 | 5.572.594.796.673.530 | |
| 155 | clean | 1,5254E+12 | 21868.0 | 4148.0 | 8645.0 | 3302.0 | 13748.0 | 333.0 | | 2755.0 | 3625.0 | 22591.0 | 4878.0 | 4.582.055.390.249.480 | 5.513.084.611.276.410 | |
| 156 | clean | 1,5254E+12 | 21860.0 | 4143.0 | 8635.0 | 3300.0 | 13753.0 | 334.0 | | 2755.0 | 3615.0 | 22591.0 | 4872.0 | 458.632.791.638.056 | 5.476.462.958.724.340 | |
| 157 | clean | 1,5254E+12 | 21850.0 | 4138.0 | 8624.0 | 3297.0 | 13759.0 | 334.0 | | 2754.0 | 3604.0 | 22592.0 | 4866.0 | 4.584.725.719.081.400 | 5.552.758.068.207.820 | |
| 158 | clean | 1,5254E+12 | 21842.0 | 4133.0 | 8613.0 | 3295.0 | 13765.0 | 333.0 | | 2753.0 | 3593.0 | 22591.0 | 4861.0 | 4.584.725.719.081.400 | 5.534.447.241.931.790 | |
| 159 | clean | 1,5254E+12 | 21833.0 | 4128.0 | 8602.0 | 3293.0 | 13769.0 | 333.0 | | 2753.0 | 3583.0 | 22591.0 | 4855.0 | 458.632.791.638.056 | 5.476.462.958.724.340 | |
| 160 | clean | 1,5254E+12 | 21824.0 | 4123.0 | 8592.0 | 3291.0 | 13774.0 | 336.0 | | 2752.0 | 3573.0 | 22591.0 | 4849.0 | 45.833.905.546.654.400 | 5.497.825.589.379.720 | |
| 161 | clean | 1,5254E+12 | 21816.0 | 4118.0 | 8581.0 | 3289.0 | 13780.0 | 333.0 | | 2751.0 | 3563.0 | 22593.0 | 4843.0 | 4.582.055.390.249.480 | 5.383.382.925.154.490 | |
| 162 | clean | 1,5254E+12 | 21808.0 | 4114.0 | 8571.0 | 3287.0 | 13786.0 | 332.0 | | 2751.0 | 3552.0 | 22590.0 | 4838.0 | 45.833.905.546.654.400 | 545.815.213.244.831 | |
| 163 | clean | 1,5254E+12 | 21800.0 | 4109.0 | 8561.0 | 3285.0 | 13789.0 | 332.0 | | 2750.0 | 3542.0 | 22590.0 | 4832.0 | 45.833.905.546.654.400 | 545.815.213.244.831 | |
| 164 | clean | 1,5254E+12 | 21791.0 | 4105.0 | 8551.0 | 3283.0 | 13795.0 | 336.0 | | 2749.0 | 3532.0 | 22593.0 | 4827.0 | 4.584.725.719.081.400 | 5.497.825.589.379.720 | |
| 165 | clean | 1,5254E+12 | 21782.0 | 4100.0 | 8541.0 | 3281.0 | 13801.0 | 334.0 | | 2748.0 | 3522.0 | 22592.0 | 4822.0 | 458.632.791.638.056 | 5.383.382.925.154.490 | |
| 166 | meas | 1,5254E+12 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 | 0.0 | 0.0 | 45.833.905.546.654.400 | 5.421.530.479.896.230 | |
| 167 | meas | 1,5254E+12 | 21708.0 | 4083.0 | 8509.0 | 3257.0 | 13794.0 | 334.0 | | 2735.0 | 3493.0 | 22576.0 | 4791.0 | 4.584.725.719.081.400 | 5.438.315.403.982.600 | |
| 168 | meas | 1,5254E+12 | 21736.0 | 4077.0 | 8498.0 | 3258.0 | 13800.0 | 335.0 | | 2732.0 | 3485.0 | 22574.0 | 4789.0 | 4.588.998.245.212.480 | 545.815.213.244.831 | |
| 169 | meas | 1,5254E+12 | 21709.0 | 4080.0 | 8488.0 | 3255.0 | 13814.0 | 334.0 | | 2730.0 | 3477.0 | 22572.0 | 4786.0 | 4.587.663.080.796.520 | 5.403.219.653.620.200 | |
| 170 | meas | 1,5254E+12 | 21697.0 | 4257.0 | 8489.0 | 3255.0 | 13801.0 | 334.0 | | 2731.0 | 3475.0 | 22554.0 | 4780.0 | 458.632.791.638.056 | 5.438.315.403.982.600 | |
| 171 | meas | 1,5254E+12 | 21689.0 | 5474.0 | 8579.0 | 3263.0 | 13587.0 | 334.0 | | 2732.0 | 3577.0 | 22570.0 | 4775.0 | 4.587.663.080.796.520 | 5.403.219.653.620.200 | |
| 172 | meas | 1,5254E+12 | 21707.0 | 7019.0 | 8774.0 | 3285.0 | 12729.0 | 334.0 | | 2740.0 | 4004.0 | 22586.0 | 4811.0 | 458.632.791.638.056 | 5.421.530.479.896.230 | |
| 173 | meas | 1,5254E+12 | 21701.0 | 8126.0 | 9147.0 | 3314.0 | 11401.0 | 331.0 | | 2775.0 | 5055.0 | 22609.0 | 5010.0 | 4.587.663.080.796.520 | 5.403.219.653.620.200 | |
| 174 | meas | 1,5254E+12 | 21671.0 | 8822.0 | 9918.0 | 3355.0 | 10163.0 | 335.0 | | 2828.0 | 7272.0 | 22657.0 | 5635.0 | 4.584.725.719.081.400 | 5.534.447.241.931.790 | |
| 175 | meas | 1,5254E+12 | 21680.0 | 9188.0 | 11137.0 | 3430.0 | 9203.0 | 334.0 | | 2896.0 | 9555.0 | 22703.0 | 6480.0 | 458.632.791.638.056 | 5.552.758.068.207.820 | |
| 176 | meas | 1,5254E+12 | 21715.0 | 9375.0 | 12365.0 | 3574.0 | 8563.0 | 336.0 | | 2957.0 | 11695.0 | 22795.0 | 7335.0 | 4.584.725.719.081.400 | 5.740.444.037.537.190 | |
| 177 | meas | 1,5254E+12 | 21874.0 | 9503.0 | 13518.0 | 3811.0 | 8050.0 | 333.0 | | 3013.0 | 13534.0 | 22898.0 | 8098.0 | 4.584.725.719.081.400 | 5.947.966.735.332.260 | |

**Ilustración 33: Ejemplo de fichero de datos**

volátiles, y otro que mide humedad y temperatura. Éstos sensores miden las muestras durante 60 segundos con un periodo de muestreo de 0.4s. A partir de éstos datos, que no son adecuados como entradas para un posible clasificador, la herramienta creará un archivo que contenga las entradas y el formato adecuados para los clasificadores, calculando de cada curva de medición proporcionada por cada sensor el máximo, el mínimo, la desviación típica, el área, la pendiente de subida y la de bajada. A su vez, éstos datos

se escribirán en un archivo que será la entrada al clasificador. Para ésta operación no se tendrán en cuenta todas las mediciones de los sensores, ya que hay un periodo de limpieza del sensor de un minuto. El último valor de éste periodo se tomará para normalizar los datos medidos, y con éstos valores se procederá a realizar todo lo anteriormente expuesto.

- La segunda cuestión a resolver es la inclusión en la herramienta de los clasificadores descritos en el punto 3, utilizando para ello las librerías de Weka. Asimismo, el programa deberá ser capaz de permitir al usuario la modificación de los parámetros de los clasificadores, así como la visualización y portabilidad de los resultados obtenidos.

## 4.2. Solución Adoptada

A continuación, se va a pasar a explicar paso a paso la solución que se ha dado a los puntos planteados en el anterior apartado.

### 4.2.1. Creación del fichero de entrada de los clasificadores

El proceso de creación del fichero de entrada para el clasificador está representado de forma genérica en la ilustración 34, en el que se pueden ver los pasos

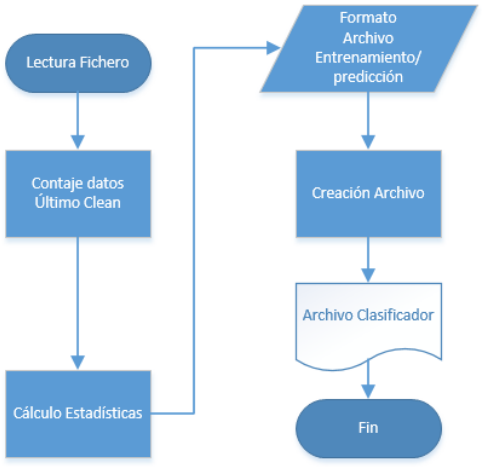



**Ilustración 34: Diagrama de Flujo creación fichero**

seguidos para la creación de un fichero apto para servir de entrada a un clasificador. Éste proceso se inicia con la lectura de los ficheros, con formato CSV, que contienen los datos en bruto proporcionados por los sensores. Así pues, lo primero será calcular el número de muestras proporcionadas por los sensores, que realmente son válidas.

Para ello, se contarán las líneas que conforman el fichero, y una función, llamada "DatosMedidos", dará como salida los datos sobre los que se va a sacar la información de entrada del clasificador en forma de matriz, donde las filas están compuestas por las medidas, y las columnas, por la etiqueta que dice si los sensores están en periodo de limpieza o de medida, el tiempo del reloj interno, y los datos de los sensores.

Nótese que, como se puede ver en la ilustración 35, se realizan dos procesos de rellenado de matrices. Esto es porque Java es un lenguaje de programación que no

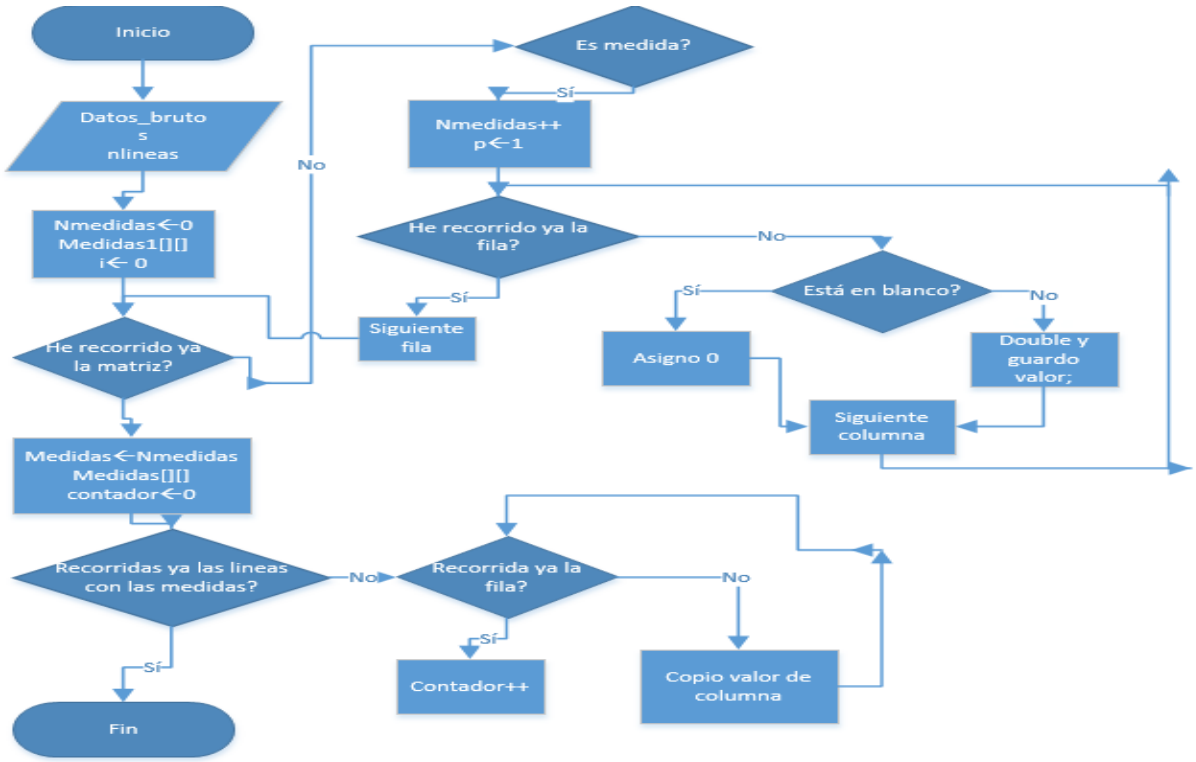



**Ilustración 35: Flujograma "DatosMedidos"**

opera con tamaños de matrices variables y, al no tener de inicio el número de medidas que el fichero contiene, sólo el número de líneas, la primera matriz contiene ceros donde la nariz electrónica está en proceso de limpieza, y los valores cuando la nariz está en proceso de medida. De éste modo se puede contabilizar de cuántas medidas dispone el fichero, y así saber el tamaño exacto de la matriz que contendrá los datos. Una vez conocido esto, se procede al llenado de la matriz ajustada a los parámetros del fichero.

El siguiente paso es normalizar los datos de las medidas (ver ilustración 36). La función “UltimoClean”, busca en la primera columna de la matriz de datos del fichero, cuándo cambia la palabra “clean” por la palabra “meas”. La última fila en la que el valor de la primera columna es “clean”, será la fila de valores que servirán para normalizar los datos.

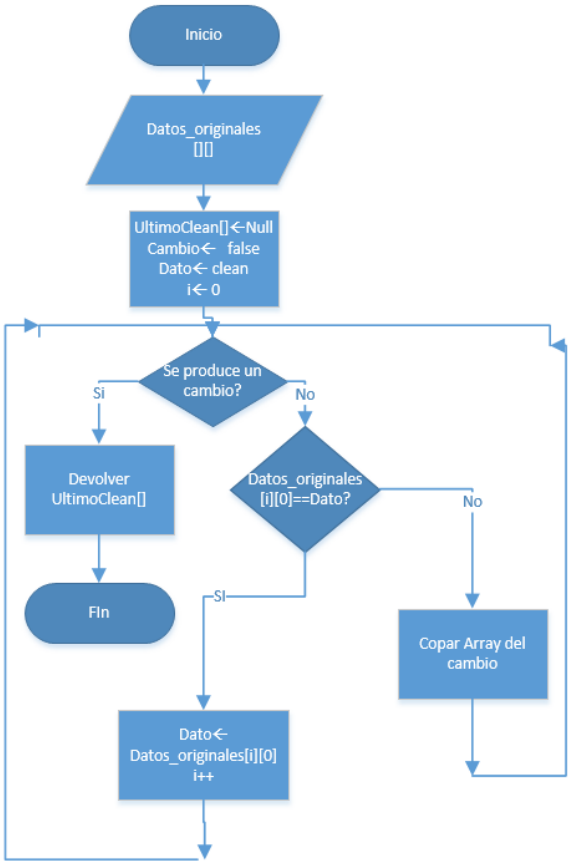



**Ilustración 36: Flujograma UltimoClean**

Así, calculada ya la fila normalizadora, el método “Normalizar” dará como salida una matriz con los datos del fichero normalizados, en la cual ya solo queda en las columnas los datos de los sensores. Esto se realizará dividiendo las columnas de la matriz de datos por su respectivo valor normalizador, contenido en el vector calculado con anterioridad.

Con los datos ya normalizados se puede proceder al cálculo de las estadísticas de la curva de medida del sensor: máximo, mínimo, desviación típica, área y pendientes. Para ello se han realizado los correspondientes métodos que las calculan, que se explicaran sin mayor detalle a continuación, ya que son problemas genéricos que no tienen ninguna especificidad que sea de interés remarcar para éste proyecto.

Para el hallazgo del valor máximo y mínimo de las medidas de los sensores normalizadas, se ha utilizado el algoritmo típico que se suele usar en estos casos, consistente en suponer que el valor máximo, o mínimo, se encuentra en la primera fila de la matriz, e ir recorriendo la columna comparando el valor preestablecido con el de la posición actual. Si el valor de la posición actual es mayor, en el caso de calcular el máximo, o menor en caso de estar hallando el mínimo, se actualiza la suposición inicial y se sigue recorriendo la columna y comparando.

El cálculo del área que encierra la curva formada por las mediciones normalizadas de cada sensor, se realizado mediante la utilización de una aproximación trapezoidal de la integral, es decir, se ha ido sumando el área que forman los trapecios formados por una medición y su siguiente.

Mención aparte merece el cálculo de la pendiente de subida y de bajada de las curvas de medida de los sensores, ya que para su cálculo, se han tomado una serie

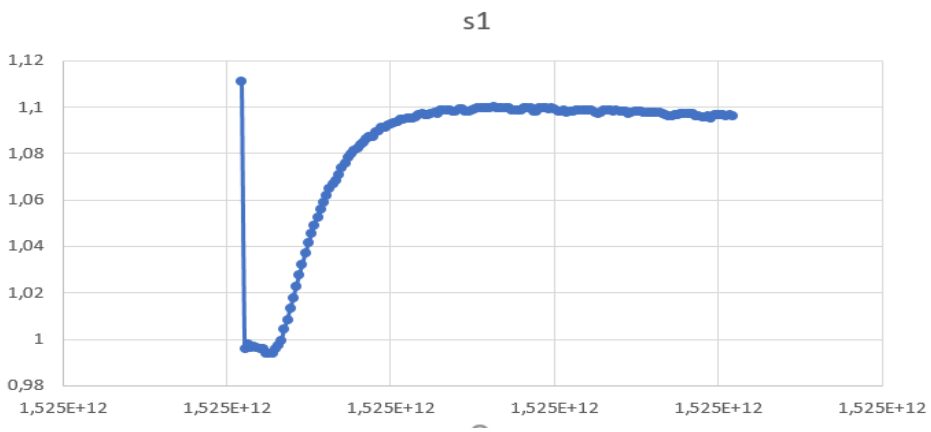



**Ilustración 37: Curva de Medida normalizada Sensor 1, muestra 53**

de consideraciones previas, basadas en el estudio de la forma de las funciones.

Las ilustraciones 37 y 38, muestran la función referenciada definida por el sensor 1y 5 para una muestra. La visión conjunta de ambas funciones sugiere que para el

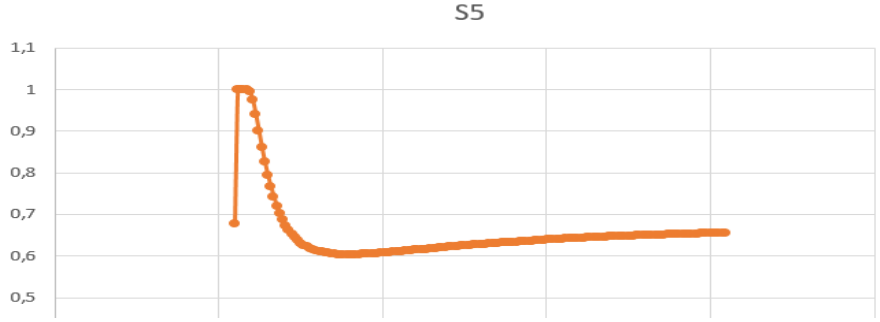



**Ilustración 38: Curva de Medida Normalizada Sensor 5, muestra 53**

cálculo de la pendiente de subida y de bajada se va a proceder de dos modos, dependiendo de si se tiene una gráfica del tipo del sensor 1 o una del tipo del sensor 5. Así, para finalmente calcular las pendientes, se va a discriminar entre los dos casos, notando que en el primero el máximo de la función está más cerca del valor final de esta, mientras que, en una función del segundo tipo, es el mínimo el más cercano.

Teniendo en cuenta éstas consideraciones, el cálculo de la pendiente de subida, en una función de la forma de la ilustración 37, se ha realizado teniendo en cuenta sólo el punto mínimo y el máximo de la función, y el cálculo de la pendiente de bajada se ha realizado teniendo en cuenta el punto máximo y el punto final de la función. Para una función con la forma de la ilustración 38 se ha procedido a la inversa. Sólo se han tenido en cuenta estos dos puntos, en lugar de realizar un proceso iterativo, ya que las distancias entre los datos sucesivos son tan cortas que el cálculo del modo descrito no difiere de forma significativa con un cálculo más exhaustivo como sería el iterativo y, por tanto, se ha escogido la opción más simple y con menos coste computacional.

Una vez hecho esto, lo siguiente será la escritura del fichero en el formato correspondiente. La clase Fichero consta de cuatro métodos para la creación de éste archivo de entrada al clasificador: “WriteDataAtOnce”, “WriteDataAtOnceT”, “WriteArffTraining” y “WriteArffPrediction”, los dos primeros crean el archivo con formato CSV, uno para entrenamiento y otro para predicción (sin la clase asociada), y los dos últimos hacen lo propio en formato ARFF.

### 4.2.2. Implementación de los Clasificadores

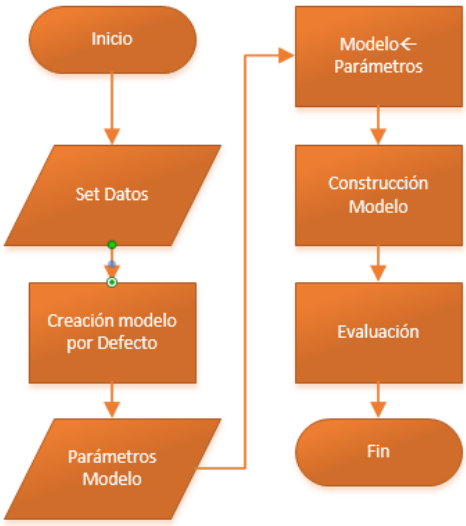



**Ilustración 39: Algoritmo genérico creación Modelo**

En la ilustración 39, se puede ver el algoritmo que se ha seguido para la implementación y evaluación de los distintos clasificadores que se han escogido. Por tanto, en éste apartado se va a explicar los pasos a seguir para la implementación en Java de los clasificadores escogidos, usando para ello las librerías de Weka. Para ello, primero se van a describir las funciones usadas para la lectura de los datos de entrada y la creación del set de datos, y después se hará lo propio con aquellas funciones que construyen los clasificadores, que están enunciadas en la clase abstracta AbstractClassifier, e implementadas en todas las clases que generan distintos clasificadores, según los parámetros propios de cada algoritmo.

#### 4.2.2.1. Lectura datos de Entrada

Para la lectura de los datos de entrada de un clasificador, contenidos en un fichero, se han usado las clases y métodos que se describen a continuación.

CSVLoader [18]: Ésta clase está contenida en el paquete de convertidores (“converters”) dentro de la librería de Weka. Contiene los métodos que se encargarán de convertir un fichero CSV cuyo separador es una coma, en un fichero de entrada ARFF para los clasificadores, con la misma estructura que el fichero original. El método utilizado de ésta clase es “getDataSet” [18]. Éste método recibe como entrada una variable de tipo cadena, con la ruta completa del archivo en cuestión, y da como salida un objeto de la clase “Instances”, que contiene todos los datos contenidos en el

archivo, conservando su estructura. En ésta clase también hay métodos que permiten conocer la estructura del fichero y el número de valores perdidos del set de datos, entre otros.

ArffLoader [19]: Ésta clase es análoga a la anteriormente descrita, y usa métodos similares para leer los datos de un archivo en formato ARFF y transformarlos en un objeto de la clase “Instances”.

Instances [20]: Es la clase que permite el manejo y modificación de las instancias del set de datos. Así, en Instances hay métodos para borrar el set de datos, añadir instancias del set de datos, o dar información sobre el número de atributos del set de datos, así como la media o la moda de los atributos sobre los que se deseen calcular dichos parámetros. Una vez cargados los datos del archivo de entrada, se usará el método “setClassIndex” para establecer qué atributo es la clase de la instancia. Éste método recibirá como entrada un número entero, que será el índice del atributo que se quiere que sea la clase. La clase usualmente ocupa el último lugar de los atributos (como es el caso ahora), por lo que, para su establecimiento, se habrá de saber cuántos atributos contiene el set de datos, con el método “numAttributes”, que se aplica sobre un objeto del tipo Instances.

Remove [21]: La herramienta implementada permite la eliminación, por parte del usuario, de los atributos que se desee. Para aplicar un filtro de eliminación y reordenar los atributos restantes, se utilizan los métodos de la clase Remove. Para ello, se crea un nuevo objeto de ésta clase, al que se le pasa, mediante “setAttributeIndicesArray”, y “setInputFormat”, los índices de los atributos a eliminar, y el formato del set de datos respectivamente. Asimismo, cabe la posibilidad de invertir la selección de atributos a eliminar, y así conservar, en vez de eliminar, los índices de atributos que se pasen con la función anteriormente descrita.

Evaluation [22]: Ésta clase es la encargada de evaluar el rendimiento de un modelo creado a partir de un clasificador, sobre un set de datos de entrenamiento o de predicción. Para ello, se usa la función “evaluateModel”, que se aplica sobre un objeto del mismo tipo, y que tiene como parámetros de entrada el modelo creado, y el set de datos sobre el que se desea evaluar el desempeño del modelo. Si se desea realizar una evaluación con validación cruzada, se usará el método

"crossValidateModel", que pedirá además el número de divisiones con las que se evaluará el modelo, y un número aleatorio. Ésta clase también contiene los métodos para poder conocer todos los descriptores estadísticos que evalúan el rendimiento del modelo (Kappa de Cohen, errores, falsos positivos y negativos…) en la predicción de la clase de la que se deseen conocer dichos parámetros, que tienen como entrada el índice de la clase: en caso de ser binaria, como es el caso, los índices serán 0 (VO) o 1 (EVO).

Para cargar y salvar los modelos creados, se hace uso de la clase auxiliar SerializationHelper [23], que contiene métodos que permite la serialización y des-serialización de objetos en archivos. En java, se entiende por serialización a la conversión de cualquier objeto en una serie de bytes que podrán ser reconstruidos más tarde, siempre y cuando la clase a la que pertenece el objeto, implemente la interfaz Serializable. De éste modo se pueden guardar los modelos creados, con el método write: Weka.core.SerializationHelper.write ("RutaArchivo" + "NombreArchivó" + ".model", Modelo), siendo Modelo el objeto de la correspondiente clase de clasificador a la que corresponda. De un modo análogo se realiza la lectura de éstos ficheros para extraer el objeto, con el método read: (Tipo Clasificador) Weka.core.SerializationHelper.read (Ruta fichero). Como se puede ver, se realiza un cast con el resultado de la des-serialización, ubicando el objeto recuperado en la clase correspondiente. Así, si, por ejemplo, se desea recuperar de un archivo, un objeto dentro de la clase perceptrón multicapa, (Tipo Clasificador) se sustituye por (MultilayerPerceptron).

#### 4.2.2.2. Implementación J48

Para la implementación del algoritmo clasificador J48, Weka posee sólo un constructor [27], el que se usa por defecto sin ningún parámetro, por lo que, para la creación del modelo con unos parámetros escogidos por el usuario, se dispone de los correspondientes métodos getter y setter. Éstos métodos permiten establecer el factor de confianza (de tipo float), el número de objetos mínimos por división (entero), el número de divisiones del set de datos que se usa para calcular el error de poda (entero), el número mínimo de instancias por hoja (entero), si el usuario habilita la poda (boolean) o si se habilita o no el error reducido de poda (boolean). Esto también se puede realizar mediante un método llamado "setOptions", que establece todos los

parámetros recibiendo una cierta cadena con los valores establecidos por el usuario. A modo de ejemplo, si se quiere establecer un factor de confianza de 0.25, y un número mínimo de instancias por hoja de 5, la llamada a la función para un cierto objeto llamado árbol, será: arbol.setOptions(new String [ ] { “-C”, “0.25”, “-M”, “5”} ).

Una vez configurados todos los parámetros, y teniendo cargados los datos de un fichero de texto como se ha indicado previamente, se construye el clasificador con el método “buildClassifier”. Éste método actúa sobre un objeto de la clase J48, y recibe otro objeto tipo Instances que contiene el set de datos, para construir el modelo, con los parámetros seleccionados, que se ajusta al set de datos. Éste método ajusta los parámetros internos del modelo, tales como la creación del árbol, pero no evalúa su rendimiento sobre el set de datos. Dicha función recae sobre la clase Evaluation, como se ha visto con anterioridad.

#### 4.2.2.3. Implementación PART

Para la implementación de éste clasificador [28], se sigue el mismo algoritmo descrito en el J48. Como las funciones que gobiernan la implementación de los clasificadores están enunciadas en la clase abstracta AbstractClassifier, el modo de construcción es análogo al descrito en el algoritmo J48, y como éste es un algoritmo que puede ser considerado como una construcción parcial de dicho árbol, los parámetros a configurar serán los mismos, con la excepción de la ausencia de reglajes como la reducción de ruido y error de Laplace.

#### 4.2.2.4. Implementación KStar

El algoritmo KStar tiene algunos aspectos diferenciadores a la hora de su programación en Java, relativos al tratamiento de los valores perdidos del set de datos. Cuando se va a instanciar un objeto de ésta clase, se debe de establecer qué tratamiento se les da a los posibles valores perdidos presentes en el set de datos. Para ello, se usa el método setter setMissingMode(), que recibe con parámetro de entrada un objeto de la clase SelectedTag. Ésta clase se usa para representar un valor seleccionado de un conjunto de valores de la clase Tag (clase [20] que asocia un número identificativo a una descripción) disponibles. Los tags disponibles se encuentran en el array “TAGS_MISSING”, en la clase KStar, y consta de cuatro valores descriptivos [21]: M_AVERAGE, para usar la curva de entropía media del

atributo, M_DELETE, para borrar las instancias con valores perdidos, M_MAXDIFF, para que los valores perdidos se traten como si tuviesen máxima diferencia con respecto a lo que se compara, M_NORMAL, normaliza los atributos. Por tanto, para seleccionar el tratamiento que se da a los valores perdidos escogido por el usuario, se usará el constructor de la clase SelectedTag, que recibirá como parámetro de entrada un string con uno de éstos cuatros valores descriptivos, y el array "TAGS_MISSING", donde se encuentran todos los valores aplicables. Y una vez hecho eso, se establecerá como tratamiento para los valores perdidos del modelo mediante el método setMissingModel(). A modo de ejemplo que clarifique lo expuesto, se va a escribir cómo sería el establecimiento de un tratamiento para los valores perdidos de eliminación para un cierto objeto, modelo, de la clase KStar llamado KStrella:

"KStrella.setMissingMode(newSelectedTag(M_DELETE, KStar.TAGS_MISSING))."

El tratamiento de los valores perdidos no es el único parámetro que se puede ajustar, ya que también se podrán establecer, con los correspondientes métodos setter, el parámetro de mezclado (entero), y si se permite o no un mezclado automático basado en la entropía (boolean).

Una vez que se hayan establecido los parámetros, la implementación del clasificador se realiza de manera análoga a lo anteriormente expuesto, con la construcción del modelo y la evaluación del mismo con el set de datos escogido.

#### 4.2.2.5. Implementación SimpleLogistic

La implementación del algoritmo clasificador Simple Logistic [22] se realiza de acuerdo al flujograma de la ilustración 38, y con las funciones relativas al común de los clasificadores, enunciadas en su clase abstracta, expuestas en los anteriores apartados. Así, los únicos métodos diferentes a los demás, usados en la implementación de éste clasificador, son los métodos setter usados para el establecimiento de los parámetros que configuran la creación del modelo: el número de iteraciones del algoritmo booster (entero), el máximo de éstas (entero), y la habilitación y des-habilitación de los distintos métodos que simple logistic implementa para la parada del algoritmo LogitBoost (boolean): la parada heurística, si la validación

cruzada está habilitada, la validación cruzada, el error en las probabilidades, o el criterio de información de Akaike (AIC).

#### 4.2.2.6. Implementación SMO

El algoritmo SMO es quizá el más complejo en su implementación, ya que posee el más extenso número de parámetros a establecer.

El más importante de ellos es la creación del núcleo. Para ello, se usará la clase relativa a éste, que para el problema que se aborda en el presente trabajo, será la clase PolyKernel [23]. Ésta clase tiene dos tipos de constructores: por un lado, está el constructor por defecto, y por otro está el que se va a usar en la implementación de éste clasificador. Éste último constructor tiene como parámetros de entrada el set de datos (Instances), el tamaño del caché, que normalmente se establece en 250007 o en 0 para caché completo, el exponente del polykernel (entero), y la habilitación o deshabilitación del uso de términos de menor orden (boolean).

Por otro lado, el algoritmo SMO [24] tiene la opción de usar otro método clasificador a modo de calibrador, en éste caso se podrá escoger de entre todos los clasificadores implementados en la herramienta, por lo que se tendrá que construir el correspondiente clasificador con sus ajustes establecidos por el usuario, de un modo análogo a como se expone en su correspondiente sección.

Otra opción a configurar es la posible aplicación de un filtro que modifique los datos del set de entrenamiento, pudiendo normalizar los valores, estandarizarlos, o no hacer ninguna de éstas dos cosas. Para el establecimiento de éste filtro, se usa un método análogo al expuesto en la implementación del algoritmo KStar: el método setter setFilterType(), que recibe como parámetro de entrada un objeto de tipo Selectedtag, establece el tipo de filtro a usar. Éstos están contenidos dentro del array TAGS_FILTER dentro de la clase SMO. Si se tuviera un objeto de la clase SMO de nombre ModeloSMO, y se quisiera establecer para la construcción del modelo un filtro normalizador sobre el set de datos, se realizaría la siguiente llamada al método: ModeloSMO.setFilterType (new SelectedTag (1, ModeloSMO. TAGS_FILTER)).

Las restantes opciones a configurar se hacen de un modo menos complejo, con los métodos setter adecuados: el parámetro de complejidad (double), el número de

subdivisiones del set de datos para la validación cruzada que se usa con el calibrador (-1 si se quiere usar el set de entrenamiento completo), la semilla aleatoria (entero), si se construye o no el calibrador (boolean), o si se chequea o no la capacidad del clasificador antes de construirse (boolean).

#### 4.2.2.7. Implementación Perceptrón Multicapa

La implementación del perceptrón multicapa sigue la forma canónica descrita en los anteriores apartados.

Primero se configuran los parámetros del clasificador con los correspondientes métodos setter [25]: el ratio de aprendizaje (double), el momento (double), la semilla (entero), el tamaño del set de validación (entero), el umbral de validación (entero), las épocas de entrenamiento (entero), si se normalizan los atributos (boolean), y la configuración de las capas ocultas (cadena).

Con respecto al parámetro de configuración de las capas ocultas, se establece una lista de números naturales separados por comas, tantos como número de capas ocultas se quiera, y el número en cuestión será el número de neuronas en la capa oculta.

Además, para la configuración de las capas ocultas también se dispone de unos valores preestablecidos: a (0.5*(atributos + clases)), t (atributos + clases), i (atributos), o (clases).

#### 4.2.2.8. Implementación OneR

Éste algoritmo es el más simple de los implementados. Se usan para ello las funciones propias de la construcción de los clasificadores, y el método setter referente al parámetro del tamaño de cubo (entero) [26], para ajustar éste parámetro propio del algoritmo clasificador.

# 5. RENDIMIENTO DE LOS CLASIFICADORES ESCOGIDOS

En éste punto del presente trabajo, se van a presentar los resultados arrojados por los algoritmos seleccionados para la tarea de clasificación que se aborda, analizando su desempeño mediante los parámetros estadísticos disponibles en la evaluación.

## 5.1. Parámetros estadísticos para la evaluación del rendimiento de los clasificadores

Para la correcta evaluación del rendimiento de los modelos creados por los clasificadores, se presta especial atención a una serie de parámetros estadísticos que se van a definir en éste punto. La mayor parte de ellos se pueden calcular de forma sencilla a partir de la matriz de confusión asociada a la clasificación realizada por el algoritmo clasificador. Ésta consiste en una tabla, en la cual filas y columnas están compuestas por las clases existentes en el set de datos, en las filas se representan los valores reales de la clasificación de las instancias, mientras que en las columnas se emplazan los resultados de la clasificación realizada. La matriz de confusión representa los errores cometidos en la clasificación (Falsos Negativos y Falsos positivos). La ilustración 40, muestra un ejemplo de matriz de confusión para una clasificación dicotómica.

| | | Predichos P | Predichos N |
|---|---|---|---|
| Reales | *P* | Verdaderos Positivos(TP) | Falsos Negativos (FN) |
| | *N* | Falsos Positivos (FP) | Verdaderos Negativos (TN) |

**Ilustración 40: Matriz Confusión**

- La Kappa De Cohen: Éste coeficiente mide la concordancia entre dos observadores, restando de esto la concordancia que se podría dar entre ellos debido al azar. En el ejemplo de la ilustración 40, el primer observador serían las predicciones, y el segundo serían los valores reales de las clasificaciones de las instancias.

$$K = \frac{P(o) - P(a)}{1 - P(a)}$$

*(Fórmula 5.1.1)*

Donde P(o) es la probabilidad de concordancia observada entre ambos observadores:

$$P(o) = \frac{TP + TN}{Total}$$

*(Fórmula 5.1.2)*

Y donde P(a) es la probabilidad de concordancia debida al azar, es decir, la probabilidad de que cada observador clasifique como positivo o negativo.

$$P(a) = \frac{TP + FN}{Total} * \frac{FP + TN}{Total} + \frac{TP + FP}{Total} * \frac{FN + TN}{Total}$$

*(Fórmula 5.1.3)*

Así, un resultado igual a 0 es equivalente a que el acuerdo entre los observadores no va más allá del esperado por azar, mientras que un resultado igual a 1, indicaría el completo acuerdo entre ellos. Como se puede observar, éste coeficiente puede tomar valores negativos, que indicarían un total desacuerdo entre los observadores.

Para el caso de una clasificación binaria, la Kappa de Cohen puede tomar cualquier valor dentro de un intervalo que va desde el -1 hasta el 1. Landis y Koch crearon una escala para la interpretación de los valores de la Kappa de Cohen, representada en la tabla 5.1.1.

| Kappa Cohen | Acuerdo |
|---|---|
| **K<0** | No acuerdo |
| **0<K<0,2** | Insignificante |
| **0,21<K<0,4** | Discreto |
| **0,41<K<0,6** | Moderado |
| **0,61<K<0,8** | Sustancial |
| **0,81<K<1** | Casi Perfecto |

*(Tabla 5.1.1)*

- Precisión: Hace referencia a la exactitud que el clasificador ha tenido a la hora de clasificar las instancias de una clase, referenciando las instancias que se han clasificado como pertenecientes a una clase con respecto a las que realmente pertenecen a la clase. Una alta precisión requiere un número muy pequeño de falsos positivos:

$$Precisión = \frac{TP}{TP + FP}$$

*(Fórmula 5.1.4)*

- Sensibilidad: La sensibilidad, o recall en Weka, hace referencia a la "facilidad" que tiene un clasificador para adjudicar la clase asociada a ésta sensibilidad a una instancia. Una gran sensibilidad conlleva un número muy pequeño de falsos negativos.

$$Sensibilidad = \frac{TP}{TP + FN}$$

*(Fórmula 5.1.5)*

- Medida F: éste parámetro es una medida de la precisión y la sensibilidad de un clasificador para una clase:

$$Medida\ F = \frac{2 * Precisión * Sensibilidad}{Precisión * Sensibilidad}$$

*(Fórmula 5.1.6)*

- MCC: El MCC o coeficiente de correlación de Mathews, mide la concordancia entre la clasificación binaria observada y la predicha. Al igual que la Kappa de Cohen, éste valor puede tomar cualquier valor entre -1 y 1, donde -1 expresa total desacuerdo entre los observadores, 0 será un acuerdo que no será distinto del que se puede producir por mero azar, y 1 es la concordancia perfecta.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

*(Fórmula 5.1.7)*

- Área PRC: Es el área que encierra la curva que relaciona la precisión, eje Y, y la sensibilidad, eje X. Un área grande representa gran precisión y sensibilidad. De ella se obtiene la precisión media del clasificador con respecto a una determinada clase evaluada.
- Área ROC (ver ilustración 41): La curva ROC es una representación de la tasa de verdaderos positivos (sensibilidad), en función a la de falsos positivos (1-precisión), en la que la clasificación perfecta de la clase correspondería al punto (0,1). Como se puede ver en la ilustración 41, la región queda dividida en dos mediante la línea diagonal que une los puntos (0,0) y (1,1). Ésta es la línea de no-discriminación, y cualquier punto sobre ésta representa una clasificación aleatoria. Por tanto, las clasificaciones de las instancias (puntos) que se ubiquen por encima de ella, serán clasificaciones mejores que el azar, y viceversa. El valor de área de ROC de un buen clasificador deberá por tanto estar por encima de 0.5.

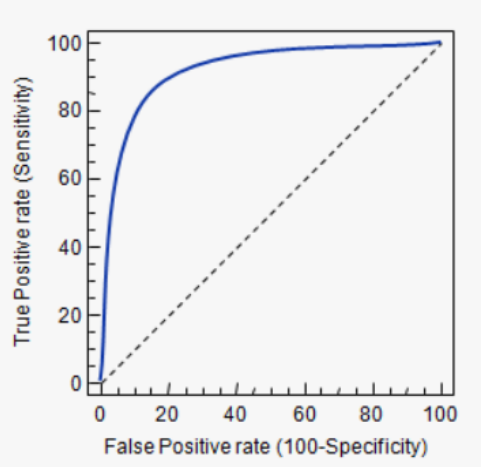



**Ilustración 41: Área ROC**

El uso del área ROC está indicado cuando el número de instancias de cada clase es parecido, ya que ésta medida podría ser demasiado optimista en sets de datos en los que esto no se cumple, mientras que el área PRC sirve

para aquellos casos en los que el número de instancias de cada clase está desequilibrado.

## 5.2. Resultado de la evaluación de los modelos creados por los clasificadores

Una vez creado el modelo a partir de un set de datos, se va a evaluar el rendimiento, o la adecuación al problema de clasificación, que da éste. Esto se puede hacer de distintos modos. Una opción poco práctica por su escaso valor a la hora de definir cómo se comportaría el clasificador con instancias nuevas, sería evaluar el modelo sobre el mismo set de datos sobre el que se construyó. Es obvio que esto dará unos resultados muy buenos que pueden distar de la realidad. Así, para evaluar el modelo sobre el mismo set de datos sobre el que se ha construido el modelo, se puede usar la validación cruzada, técnica en la cual se hacen un número “k” establecido de subdivisiones del set de datos, construyendo un modelo con las “k-1” subdivisiones, y validando éste con la subdivisión restante, a cada vuelta del algoritmo (esto es, si el número de subdivisiones son “k”, el algoritmo dará “k” vueltas). El resultado final del clasificador será la media del resultado obtenido en cada iteración.

También se puede usar como método, para evitar unos resultados demasiado optimistas del clasificador, una división del set de datos en la que, un cierto tanto por ciento de las instancias se usen como construcción del modelo, y otro como validación.

Mientas que la técnica de validación cruzada se puede utilizar en la mayoría de los casos sin menoscabo a la hora de la creación del modelo, aunque el coste computacional es alto si las subdivisiones son numerosas, el método de la división del set de datos está desaconsejado en casos en los que las instancias de los sets de datos no son muy numerosas, ya que el modelo generado por éstos sets de datos poco numerosos podría tener problemas a la hora de generalizar.

A continuación, se van a presentar las pruebas de clasificación realizadas sobre el set de datos usado para construir el modelo, usando el método de validación cruzada para que los resultados de la evaluación del mismo se adecúen a lo esperado si éste tuviese que clasificar instancias nuevas. Se ha elegido éste método por sobre una división entre set de entrenamiento y set de validación, al contarse con pocas instancias para realizar tal división.

### 5.2.1. Árbol J48

Los parámetros a configurar para la creación del árbol J48 son los siguientes:

- Factor de Confianza: como se ha mencionado en el punto en el que se explicaba dicho algoritmo, el factor de confianza se utiliza para calcular la estimación del error de predicción, que se usará para el proceso de poda del árbol. Éste valor se puede configurar dentro del rango (0, 0.5], donde para un valor pequeño, se poda más el árbol mientras que para un valor más grande, el árbol se poda menos. Para un valor más allá de 0.5, el árbol no se podará.
- Mínimo número de objetos: éste parámetro impone un número mínimo de instancias: para un nodo padre, al menos dos de los nodos hijos tienen que tener un número mínimo de instancias para que la división de éste nodo padre se implemente, impidiendo así que el árbol se sobreajuste si el valor está correctamente ajustado.
- Poda de error reducido, semilla y número de subdivisiones (folds o pliegues), hacen referencia al uso del método del error reducido para la poda, en lugar del método por defecto de poda basada en el error (la que usa el factor de confianza para la estimación de éste). Éste método de poda evalúa la sustitución de cada hoja por su clase mayoritaria, y si el árbol que resulta de esto no rinde peor sobre el set de validación (cuyo tamaño se va a configurar con el número de subdivisiones) que el árbol original, ésta sustitución se hace permanente. Antes de realizar la división del set de datos en entrenamiento y validación, los datos se barajan. Para ello se generan números aleatorios cuya semilla se puede configurar también. La poda de error reducido será útil en sets de datos grandes, y, ya que divide los datos existentes para el entrenamiento, muy desaconsejada en sets de datos pequeños.
- Crecimiento de subárbol: es un método de poda basado en la sustitución de un subárbol por su rama mayoritaria, en caso de que ésta sustitución no aumente el error estimado.
- Colapsar árbol: Se eliminan los nodos que, aunque mejora la entropía del set de entrenamiento, no mejoran su clasificación. Esto se realiza antes de empezar la poda basada en el error.

- Corrección MDL: éste es un método que calcula la ganancia de información de atributos numéricos. Tiene la peculiaridad de producir resultados negativos, los cuales conllevan que el atributo numérico asociado no se use en la división.
- No podar: ésta opción deshabilita la poda.
- Corrección Laplace: Para calcular la probabilidad de que una instancia pertenezca a una clase dentro de una hoja, se usa la ecuación 5.2.1.1.

$$P\left({}^{Clase}/_{Instancia}\right) = \frac{Nclase + 1}{N + C}$$

*(Fórmula 5.2.1.1)*

Donde Nclase es el número de instancias clasificadas en una determinada clase dentro de la hoja, N es el número de instancias clasificadas en la hoja, y C es el número de clases que la hoja clasifica.

Una vez explicados los parámetros a configurar, se presentan los resultados obtenidos para las distintas configuraciones de archivos de entrada, con la configuración óptima obtenida de sus parámetros. Antes de empezar con el entrenamiento y la evaluación de los resultados, se ha aplicado un filtro sobre las instancias, que las cambia de orden, para impedir que haya demasiadas instancias seguidas de la misma clase. Éste filtro está disponible también en la herramienta desarrollada.

A través de la experimentación con éstos ficheros de entrada, se ha observado que el filtro de suavizado de Laplace, y el método de poda de error reducido (éste especialmente), empeoran el rendimiento del clasificador. Con todos estos parámetros

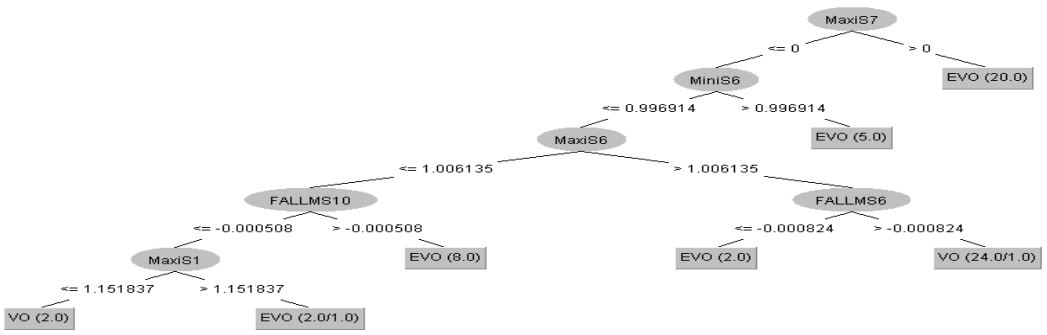



**Ilustración 42: Árbol J48 óptimo Resultante**

deshabilitados, la construcción del árbol apenas varía con la modificación de los demás resultados, con excepción de la corrección MDL, que mejora los resultados. Así, el árbol construido con los parámetros configurados de forma óptima, está representado en la ilustración 42.

Así, para un factor de confianza igual a 0.25, un número mínimo de instancias por nodo igual a 2, el método de poda basado en el error, el crecimiento del árbol, la corrección MDL y el colapso del árbol activados, el filtro de Laplace y el método de error reducido desactivados, los resultados para un conjunto de pruebas con instancias de la clase EVO predominante, son los de la ilustración 43.

```
Time taken to build model: 0 seconds

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

Correctly Classified Instances          50               79.3651 %
Incorrectly Classified Instances        13               20.6349 %
Kappa statistic                          0.5815
Mean absolute error                      0.2467
Root mean squared error                  0.4073
Relative absolute error                 50.7449 %
Root relative squared error             82.5657 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,784     0,192     0,853       0,784   0,817      0,584    0,849     0,906     EVO
               0,808     0,216     0,724       0,808   0,764      0,584    0,849     0,719     VO
Weighted Avg.  0,794     0,202     0,800       0,794   0,795      0,584    0,849     0,829

=== Confusion Matrix ===

  a  b   <-- classified as
 29  8 |  a = EVO
  5 21 |  b = VO
```

**Ilustración 43: Resultados sobre set de entrenamiento con validación cruzada J48**

Como se puede apreciar, el coste computacional del clasificador es muy escaso, y el árbol creado es muy sencillo y poco profundo.

La ilustración 43, muestra el resultado de la evaluación del modelo creado, sobre el mismo set, pero usando el método de validación cruzada, para eliminar de ese modo el problema que hay al evaluar el modelo sobre el mismo set sobre el que se ha creado, que daría unos resultados demasiados optimistas del desempeño del mismo. Con el método de validación cruzada, se obtienen unos resultados más cercanos a los que de verdad tendría el modelo al enfrentarse a instancias nuevas.

Así, se puede observar que el área ROC está por encima de la línea de no discriminación, al estar su valor por encima de 0.5. Esto indica que el modelo creado es un buen clasificador. Algo parecido vienen a expresar los parámetros Kappa de Cohen y el MCC, que al estar por encima de 0 indican que hay concordancia entre predicción y realidad y que ésta es mayor que la que se produciría por mero azar, siendo ésta, según la escala de Landis y Koch, moderada.

Más interesante que el área de la región de convergencia, resulta en éste caso el área PRC, al ser un set de datos desbalanceado en el número de instancias por clase (al disponerse de 37 instancias de la clase EVO y 26 de tipo VO). Éste muestra un desempeño mejor del clasificador catalogando instancias de la clase VO que de la EVO. Esto es consecuencia de la menor sensibilidad mostrada por éste con respecto a la clase EVO, dado por el resultado de dos falsos negativos que se puede ver en la matriz de confusión. La menor sensibilidad de la clase EVO se refleja también en la Medida F, que depende de los parámetros de sensibilidad y precisión, y es precisamente ésta sensibilidad menor la que hace que la precisión para la clase EVO sea mayor que la VO.

### 5.2.2. PART

Los parámetros disponibles para su configuración en éste clasificador son idénticos a los del algoritmo j48, con la única ausencia del filtro de Laplace.

```
PART decision list
------------------

MaxiS7 <= 0 AND
MiniS6 <= 0.996914 AND
MaxiS6 > 1.006135 AND
FALLMS6 > -0.000824: VO (24.0/1.0)

FALLMS10 > -0.000539: EVO (35.0)

: VO (4.0/1.0)

Number of Rules  :      3
```

**Ilustración 44: reglas part**

Como se puede apreciar (ver ilustración 44), las reglas generadas por éste algoritmo, son parecidas al árbol obtenido por el algoritmo j48 (por ejemplo, la raíz es la misma para ambos algoritmos), pero los resultados obtenidos sobre el set de datos, mediante la validación cruzada, son ligeramente mejores.

```
Time taken to build model: 0 seconds

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

Correctly Classified Instances          52               82.5397 %
Incorrectly Classified Instances        11               17.4603 %
Kappa statistic                          0.6419
Mean absolute error                      0.1842
Root mean squared error                  0.4184
Relative absolute error                 37.8804 %
Root relative squared error             84.8145 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,838    0,192    0,861      0,838    0,849      0,642    0,783     0,794     EVO
                 0,808    0,162    0,778      0,808    0,792      0,642    0,783     0,660     VO
Weighted Avg.    0,825    0,180    0,827      0,825    0,826      0,642    0,783     0,738

=== Confusion Matrix ===

  a  b   <-- classified as
 31  6 |  a = EVO
  5 21 |  b = VO
```

**Ilustración 45: Resultados sobre set de entrenamiento con validación cruzada PART**

Como se puede comprobar en la imagen 45, hay ligeras desviaciones en los resultados de la evaluación, obteniendo un área ROC y PRC algo menores, no obstante, el estadístico Kappa de Cohen, indica que la concordancia entre predicción y realidad es moderada.

Una observación más simple es que, con un tamaño menor de set de reglas con respecto al algoritmo j48, se han dado menos errores de clasificación, por lo que la relación complejidad-desempeño de éste clasificador es mucho mejor que la que se da para el algoritmo J48.

### 5.2.3. Resultados K*

Los parámetros con los que se puede buscar el modelo óptimo de éste algoritmo para el problema de clasificación que se aborda, son:

- Parámetro de mezclado: es el parámetro que se usa para el cálculo de $x_0$ y s, si no está activado el automezclado entrópico. Esto configura el número de instancias relevantes para la clasificación de una determinada instancia, es decir, el número de “vecinos” que influyen en la clasificación de una instancia.

- Automezclado entrópico: los parámetros $x_0$ y s se calculan automáticamente mediante la entropía.
- Tratamiento de los valores perdidos: Para sets de datos en los que haya instancias con valores perdidos, se dispone de cuatro maneras distintas de tratarlos: eliminar las instancias con valores perdidos, tratarlos como si la diferencia con el valor de atributo de la instancia actual fuese máxima, o reemplazarlos por el valor medio del atributo en el set de datos.

Para el fichero de entrada de los anteriores casos, se han probado diferentes configuraciones del algoritmo, siendo la óptima aquella en la que el automezclado entrópico está desactivado y el parámetro de mezclado está establecido en un 35%. Los resultados obtenidos por éste clasificador son, como se puede ver en la ilustración 46, algo peores que los obtenidos mediante la creación de árboles de decisión.

```
Time taken to build model: 0 seconds

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

Correctly Classified Instances          48               76.1905 %
Incorrectly Classified Instances        15               23.8095 %
Kappa statistic                          0.506
Mean absolute error                      0.2411
Root mean squared error                  0.4727
Relative absolute error                 49.5917 %
Root relative squared error             95.8244 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,811     0,308     0,789       0,811   0,800      0,506    0,792     0,794     EVO
               0,692     0,189     0,720       0,692   0,706      0,506    0,794     0,738     VO
Weighted Avg.  0,762     0,259     0,761       0,762   0,761      0,506    0,792     0,771

=== Confusion Matrix ===

  a  b   <-- classified as
 30  7 |  a = EVO
  8 18 |  b = VO
```

**Ilustración 46: Resultados sobre set de entrenamiento con validación cruzada PART**

El modelo creado por éste algoritmo, clasifica algo más fácilmente instancias de la clase EVO (tiene más sensibilidad para ésa clase en concreto), y su precisión a la hora de clasificar éstas como EVO también es mayor.

Los índices que marcan la concordancia entre los observadores teniendo en cuenta la concordancia que habría por azar (Kappa de Cohen y MCC), son peores que en los dos anteriores clasificadores, estando en el rango de concordancia moderada.

Por otro lado, el área bajo la curva de precisión-sensibilidad (PRC) es un valor alto, por encima de 0.5, esto indica que el modelo creado por el algoritmo K* con los parámetros que se han establecido, es adecuado para el problema de clasificación que se aborda.

#### 5.2.4. Resultados SimpleLogistic

La implementación de éste algoritmo permite ajustar los siguientes parámetros, para adaptarlo a los requerimientos de cada problema clasificatorio que se aborde.

- Usar error en probabilidades: se usa ésta técnica para calcular el error, que servirá para la determinación del número óptimo de iteraciones del algoritmo LogitBoost
- Número de iteraciones de boosting: establece un número fijo de iteraciones del algoritmo LogitBoost.
- Máximo de iteraciones de boosting: establece un valor máximo de iteraciones que LogitBoost realiza.
- Parada heurística: Es un criterio de parada de LogitBoost. La forma [9] de cierta una curva en la que se representa el error con el número de iteraciones del algoritmo, empieza en un cierto valor y va decreciendo, hasta que llega a un mínimo. Alcanzado éste, el error aumentará otra vez. Así pues, LogitBoost tendría que parar en las iteraciones en las que el error es mínimo, pero para evitar posibles irregularidades en la función que den lugar a una parada en un falso mínimo, se realiza un número establecido de iteraciones más y se para si éste mínimo no ha cambiado. Éste algoritmo puede ser usado en concordancia con el que se va a explicar a continuación de validación cruzada, ayudando a encontrar las iteraciones óptimas en cada vuelta del algoritmo de validación cruzada.
- Validación Cruzada [9]: se trata de otro criterio de parada del algoritmo LogitBoost, parándose cuando el algoritmo no consigue más precisión sobre

instancias que no se han visto. Así, se divide el set de datos, y se implementa LogitBoost con un número máximo de iteraciones (parámetro anterior, aunque se suele establecer por defecto en 500), y calcula el error sobre la respectiva división de prueba. Una vez realizado esto, se aplica LogitBoost al set de datos completo, que dará el número de vueltas óptimo calculado con anterioridad (el que ha dado menor error medio en el método de validación cruzada).

- Criterio de información de Akaike: se utiliza como criterio de parada de LogitBoost. Éste método se basa en la comparación de modelos, seleccionando el que tenga éste parámetro menor. Para ello, se basa en un balance entre ajuste y complejidad del modelo creado, penalizando de esta manera el sobreajuste. El AIC se puede calcular como:

$$AIC = 2k - 2 * \mathrm{Log}(\mathrm{Likelihood})$$

*(Fórmula 5.2.4.1)*

Donde Log(Likelihood) es el logaritmo neperiano de la verosimilitud máxima, y k es el número de parámetros escogidos. No obstante, para sets de datos más pequeños, se usa la siguiente corrección:

$$AICc = AIC + \frac{2k^2 + 2k}{n - k - 1}$$

*(Fórmula 5.2.4.2)*

Donde n es el tamaño del set de datos.

- Ponderación β: Sólo las instancias que tengan un peso de (1-β) son usadas en la iteración siguiente de LogitBoost.

La configuración del modelo óptimo que se ajusta al problema de clasificación del aceite de oliva, es indiferente al criterio de parada escogido, siendo los resultados los mismos para cada criterio. No obstante, pese a que las diferencias entre criterios de parada son mínimas, el modelo creado con el método de parada basado en el error sobre las probabilidades, es el que mejor desempeño tiene.

```
Class EVO :
182.41 +
[MaxiS7] * 0.61 +
[MaxiS8] * -1.55 +
[MaxiS9] * 0.78 +
[MaxiS10] * -86.64 +
[AreaS1] * -0.1 +
[AreaS4] * -0.04 +
[AreaS6] * -1.48 +
[AreaS7] * 0.06 +
[STDS2] * -2.59 +
[STDS5] * 50.76 +
[STDS9] * 2.26 +
[STDS10] * -15.45 +
[RISEMS1] * 220.31 +
[RISEMS6] * -178.39 +
[RISEMS8] * -14.06 +
[RISEMS11] * 4.06 +
[FALLMS1] * 1056.37 +
[FALLMS6] * 59.15
```

```
Class VO :
-182.41 +
[MaxiS7] * -0.61 +
[MaxiS8] * 1.55 +
[MaxiS9] * -0.78 +
[MaxiS10] * 86.64 +
[AreaS1] * 0.1 +
[AreaS4] * 0.04 +
[AreaS6] * 1.48 +
[AreaS7] * -0.06 +
[STDS2] * 2.59 +
[STDS5] * -50.76 +
[STDS9] * -2.26 +
[STDS10] * 15.45 +
[RISEMS1] * -220.31 +
[RISEMS6] * 178.39 +
[RISEMS8] * 14.06 +
[RISEMS11] * -4.06 +
[FALLMS1] * -1056.37 +
[FALLMS6] * -59.15
```

**Ilustración 47: Odds**

Algo interesante acerca de la construcción de éste modelo, es que se puede saber, mediante el análisis de los odds(ver ilustración 47), los parámetros que son más decisivos que otros a la hora de catalogar una instancia en cada clase. Así, los valores más altos representan una mayor probabilidad de pertenencia a la clase cuando se

```
Time taken to build model: 0.02 seconds

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

Correctly Classified Instances          53               84.127  %
Incorrectly Classified Instances        10               15.873  %
Kappa statistic                          0.6609
Mean absolute error                      0.1573
Root mean squared error                  0.3455
Relative absolute error                 32.3616 %
Root relative squared error             70.0329 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0,946     0,308     0,814       0,946   0,875      0,675    0,906     0,873     EVO
               0,692     0,054     0,900       0,692   0,783      0,675    0,905     0,913     VO
Weighted Avg.  0,841     0,203     0,849       0,841   0,837      0,675    0,906     0,890

=== Confusion Matrix ===

  a  b   <-- classified as
 35  2 |  a = EVO
  8 18 |  b = VO
```

**Ilustración 48: Resultados sobre set de entrenamiento con validación cruzada SimpleLogistic**

sube el valor atributo en una unidad, manteniendo el resto constante (RISEMS1 para la clase EVO).

Al ser éste algoritmo especializado en la clasificación de variables dicotómicas, por la propia forma de la función discriminante Logit, el resultado de la predicción de éste modelo (ver ilustración 45) es algo mejor que los anteriores algoritmos evaluados. Éste método sólo será superado por el algoritmo SMO y el perceptrón multicapa.

Ésta adecuación del algoritmo a la clasificación binaria se hace obvia en la Kappa de Cohen obtenida, donde según la escala de Landis y Koch, la concordancia es casi moderada.

Las puntuaciones obtenidas en los demás estadísticos de la evaluación, como el área PRC, son igualmente buenos, notándose, no obstante, una menor precisión a la hora de clasificar la clase EVO con respecto a la exactitud total obtenida por la clase VO (esto se hace evidente en la matriz de confusión, pudiéndose observar un mayor número de falsos positivos). Esto se traduce, como suele pasar, en una mayor sensibilidad a la hora de clasificar una instancia como EVO. Es por ello que el área PRC para la clase EVO es menor, es decir, el modelo clasifica peor la clase EVO que la VO.

#### 5.2.5. SMO

Éste algoritmo cuenta con los siguientes parámetros a configurar:

- Parámetro complejidad: como se ha explicado en el punto en el que se detallaba el funcionamiento de éste algoritmo, el parámetro de complejidad es el encargado de permitir mayores o menores errores de clasificación en la fase de entrenamiento del modelo. Así, éste parámetro tiene mucho que ver con el sobreajuste del mismo.
- Calibrador: Se usa para obtener estimaciones probabilísticas más ajustadas a la realidad. El que suele ser más usual como calibrador es el algoritmo logístico.

- Tipo filtro: se puede elegir entre normalizar los datos del set de entrenamiento, estandarizarlos (reescalarlos de forma que tengan de media 0 y de desviación típica 1), o no realizar ninguna operación.
- Kernel: es el núcleo que se va a usar para la transformación del espacio.
- Número pliegues: Para los modelos de calibración, se generan sets de entrenamiento mediante validación cruzada. Este parámetro define el número de subdivisiones.

El kernel que mejores resultados ha dado, con mucha diferencia respecto a otros, es el polyKernel, que se puede configurar especificando el grado, o habilitando y deshabilitando una opción para que el modelo sólo coja los términos de menor orden.

Mediante las pruebas realizadas, se ha comprobado que la variación del parámetro de complejidad no altera la evaluación del modelo sobre un set de pruebas. Por el contrario, la selección de un filtro de estandarización, o la no aplicación de filtros supone un empeoramiento de dicha evaluación, por lo que se ha aplicado un filtro de

```
Number of support vectors: 24

Number of kernel evaluations: 1833 (97.71% cached)



Time taken to build model: 0.01 seconds

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

Correctly Classified Instances          54               85.7143 %
Incorrectly Classified Instances         9               14.2857 %
Kappa statistic                          0.7036
Mean absolute error                      0.1429
Root mean squared error                  0.378
Relative absolute error                 29.3818 %
Root relative squared error             76.6131 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,892    0,192    0,868      0,892    0,880      0,704    0,850     0,838     EVO
               0,808    0,108    0,840      0,808    0,824      0,704    0,850     0,758     VO
Weighted Avg.  0,857    0,158    0,857      0,857    0,857      0,704    0,850     0,805

=== Confusion Matrix ===

  a  b   <-- classified as
 33  4 |  a = EVO
  5 21 |  b = VO
```

**Ilustración 49: Resultados sobre set de entrenamiento con validación cruzada SMO**

normalización. Éstos resultados se han obtenido para un polykernel de exponente 4. Los resultados sobre el set de pruebas se pueden ver en la ilustración 49.

Éstos resultados son ligeramente mejores que los obtenidos mediante el algoritmo SimpleLogistic, ya que el método clasifica mejor el set de datos.

El índice Kappa de Cohen, muestra un nivel de concordancia sustancial, y la precisión para la clase EVO es ligeramente mayor que para la clase VO. Esto a su vez significa que la sensibilidad para la clase VO es ligeramente menor.

Todo ello revierte en un área PRC mayor en la clase EVO, al ser éste área un balance entre precisión y sensibilidad.

### 5.2.6. Perceptrón Multicapa

Para la construcción de un modelo basado en el algoritmo del perceptrón multicapa, se pueden configurar los siguientes parámetros:

- Razón de aprendizaje: como se explicó en el punto referente a éste algoritmo, es la penalización que se aplica a la actualización de los pesos de la red. Éste parámetro influye en la velocidad de convergencia de la red. Así, para valores altos de éste parámetro, la convergencia será más rápida, pero hay mucho riesgo de que el algoritmo oscile alrededor de un mínimo. Mientras que, si el valor es muy bajo, la convergencia será más lenta.
- Capas ocultas: se puede configurar tanto el número de capas ocultas como el número de neuronas por capa. Esto influye, como se ha visto con anterioridad, en la forma de la región de clasificación creada para cada clase.
- Momento: es una corrección que se realiza sobre la razón de aprendizaje, almacenando la última actualización de pesos y ponderar ésta con la actual.
- Épocas: es el número de iteraciones del algoritmo de backpropagation.
- Decaimiento: la razón de aprendizaje disminuye.
- Tamaño de set de validación: es el tanto por ciento de instancias que se emplean como set de validación.
- Umbral Validación: Si los resultados del error sobre el set de validación empeoran durante el número de veces consecutivas que se establezca, el aprendizaje se detiene.

- Normalizar atributos: como en el SMO, se trata de aplicar un filtro a los atributos que hace que la media de cada uno de éstos sea 0 y la desviación típica, 1. Esto también se puede realizar sobre la clase, si ésta es numérica.
- Filtro Nominal a Binario: aplica un filtro a las instancias que convierte los atributos nominales a binarios.

Las pruebas realizadas han mostrado que la no aplicación de un filtro de normalización, ésta vez da unos resultados mucho peores que si está aplicado. Algo similar ha pasado al activar el decaimiento.

Asimismo, las configuraciones de capas ocultas con más de dos capas, empeoran los resultados, siendo la configuración que mejor resultados ha obtenido, aquella en la que hay dos capas, la primera con un número de neuronas que sea igual al número de atributos más la clase, y la segunda capa con un número de neuronas igual a la mitad del número de atributos más el número de clases (t, a). La razón de aprendizaje se ha ajustado a 0.2, y el momento a 0.8. Como la tasa de aprendizaje está ajustada a un número muy pequeño, el número de épocas se ha ajustado a 2000.

```
Time taken to build model: 20.58 seconds

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

Correctly Classified Instances          56               88.8889 %
Incorrectly Classified Instances         7               11.1111 %
Kappa statistic                          0.7695
Mean absolute error                      0.1245
Root mean squared error                  0.3193
Relative absolute error                 25.6161 %
Root relative squared error             64.7309 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,919    0,154    0,895      0,919    0,907      0,770    0,912     0,889     EVO
               0,846    0,081    0,880      0,846    0,863      0,770    0,912     0,925     VO
Weighted Avg.  0,889    0,124    0,889      0,889    0,889      0,770    0,912     0,903

=== Confusion Matrix ===

  a  b   <-- classified as
 34  3 |  a = EVO
  4 22 |  b = VO
```

**Ilustración 50: Resultados sobre el set de entrenamiento con validación cruzada, Perceptrón Multicapa**

Como el set de datos es pequeño, los resultados obtenidos usando un set de validación para la convergencia del error, han sido sensiblemente peores.

Más irrelevantes han sido, como cabía esperar por otro lado, ajustes como el filtro nominal a binario (no hay ningún atributo nominal), o el filtro que normaliza una clase numérica (al ser la clase binaria).

A pesar de los buenos resultados que se han obtenido, éste método tiene una gran desventaja con respecto a otros: el alto coste computacional, que hace que el tiempo de construcción del modelo se haya demorado hasta los 20 segundos.

A pesar del tiempo empleado, los resultados sobre el set de validación son buenos, basándose en los indicadores estadísticos que proporciona Weka:

El análisis de la Kappa de Cohen obtenida, dice que hay una concordancia sustancial y, al igual que SMO, el modelo creado por éste clasificador es algo menos preciso clasificando instancias de la clase EVO que de la clase VO. Pese a esta ligera inexactitud (dada por un falso positivo), los resultados del área PRC muestran que el modelo creado es un buen clasificador.

### 5.2.7. OneR

Éste es el clasificador más sencillo escogido, con sólo un parámetro a configurar. Éste es mínimo tamaño de cubo, que es el número mínimo de instancias de la clase mayoritaria que tiene que tener un subgrupo para que se cree ese subgrupo. Éste parámetro evita el sobreajuste.

El modelo que da los resultados óptimos es aquel cuyo parámetro de tamaño mínimo de cubo es igual a 6.

```
MaxiS6:
        < 1.006144407735        -> EVO
        < 1.00760261194 -> VO
        < 1.0092166116049999    -> EVO
        < 1.01201244532 -> VO
        < 1.01826488249 -> EVO
        >= 1.01826488249        -> VO
(55/63 instances correct)
```

**Ilustración 51:reglas de clasificación creadas**

```
Time taken to build model: 0 seconds

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

Correctly Classified Instances          51               80.9524 %
Incorrectly Classified Instances        12               19.0476 %
Kappa statistic                          0.5931
Mean absolute error                      0.1905
Root mean squared error                  0.4364
Relative absolute error                 39.1757 %
Root relative squared error             88.4652 %
Total Number of Instances               63

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,919    0,346    0,791      0,919    0,850      0,606    0,786     0,774     EVO
                 0,654    0,081    0,850      0,654    0,739      0,606    0,786     0,699     VO
Weighted Avg.    0,810    0,237    0,815      0,810    0,804      0,606    0,786     0,743

=== Confusion Matrix ===

  a  b   <-- classified as
 34  3 |  a = EVO
  9 17 |  b = VO
```

**Ilustración 52: Resultados sobre el set de entrenamiento con validación cruzada, OneR**

Las reglas de clasificación creadas con éste modelo, que se pueden ver en la ilustración 51, dan unos resultados que, como se puede ver (ver ilustración 52), son sensiblemente peores que los algoritmos que utilizan funciones matemáticas. Esto se hace evidente en el análisis de la Kappa de Cohen, que da un nivel de acuerdo moderado. Así también, se puede ver en el análisis del área PRC, que en el caso de la clase VO, tiene un valor más bajo que en las anteriores, consecuencia del bajo valor de sensibilidad que tiene ésta clase, cosa que se puede observar en el hecho de que hay 9 falsos positivos para la clase EVO.

Se puede concluir, por tanto, que éste clasificador da peores resultados que SMO y el perceptrón multicapa.

### 5.3. Conclusiones de los Resultados

Como se ha podido observar a lo largo de la exposición de los resultados dados por los clasificadores, los mejores resultados son los obtenidos por aquellos algoritmos que utilizan funciones matemáticas para la creación de regiones que cataloguen las instancias por clases, separando éstas mediante hiperplanos, como

son SMO y el perceptrón multicapa. Los resultados de éstos dos algoritmos han resultado ser los que mayor concordancia tenían entre realidad y predicción, y los que mayor número de instancias han logrado clasificar de forma exitosa.

El algoritmo SimpleLogistic también ha demostrado ser un clasificador aceptable para la problemática planteada en el presente trabajo ya que, como se ha visto, la función discriminante usada es ideal para clasificaciones binarias.

Algo peor ha sido el desempeño del algoritmo PART, que ha demostrado ajustarse mejor a los datos que el J48, a pesar de ser una construcción menos compleja de éste último.

El algoritmo OneR ha tenido un rendimiento peor que los anteriores, aunque los resultados no han sido malos pese a la sencillez con la que éste algoritmo crea las reglas de decisión.

Pero el algoritmo que ha obtenido los peores resultados es el K*, que ha demostrado ser menos capaz que los anteriores de realizar de forma correcta la clasificación, pudiendo tomar como conclusión que los algoritmos “lazy”, basados en las k instancias vecinas, son los menos recomendados para la clasificación de aceites según su calidad.

# 6. HERRAMIENTA IMPLEMENTADA

En éste apartado se presentará la forma final de la herramienta realizada en Java, que incorpora los clasificadores anteriormente expuestos.

Ésta consta de dos partes: la parte en la cual se crean los archivos que servirán como entrada del clasificador, y aquella en la que se realiza la clasificación, con los diferentes algoritmos implementados para ello. Todo ello está contenido en la pantalla de inicio que se puede ver en la ilustración 53.


**Ilustración 53: Pantalla principal**

En ella se puede acceder a éstas dos partes, la de lectura y creación de archivos, y la de clasificación (ver ilustración 54), mediante la correspondiente pestaña de la barra

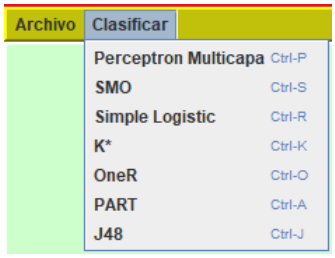



**Ilustración 54: pestaña Clasificar**

de herramientas.

Como se puede observar, también se puede acceder a cada una de estas partes mediante el atajo de teclado habilitado para ello.

## 6.1. Preparación y carga de archivos

Para preparar los archivos de entrada al clasificador, que se obtendrán a partir de archivos CSV que contienen diferentes muestras, se tiene la siguiente interfaz.

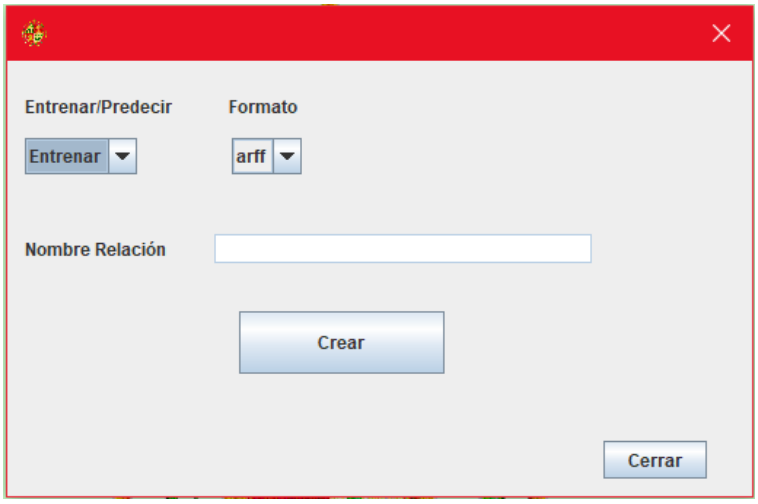



**Ilustración 55: interfaz de creación de archivos**

En ella (ver ilustración 5), se puede seleccionar, a través de las correspondientes cajas de opciones, tanto el formato del archivo de salida, arff o csv, como si éste archivo contiene o no la clase de cada instancia (entrenar o predecir). Asimismo, para los archivos de formato arff, se dispone de un campo de texto para el establecimiento del nombre de la relación, que sólo se habilita si se ha seleccionado como formato del archivo de salida, el arff.

Ésta ventana contempla la posibilidad de que el usuario cancele la operación de creación en cualquier momento, lanzando el correspondiente mensaje de cancelación

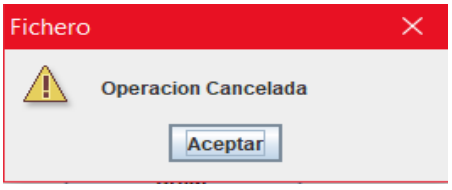



**Ilustración 56: Mensaje cancelación**

de la operación (ver ilustración 56). Si no se cancela antes, la operación termina con la creación del archivo deseado, guardado en la carpeta que el usuario escoja.

La carga de los archivos que servirán como entrada para los clasificadores, se realiza mediante otra interfaz gráfica (ver ilustración 57), a la que se accede por el correspondiente botón de cargar archivo.

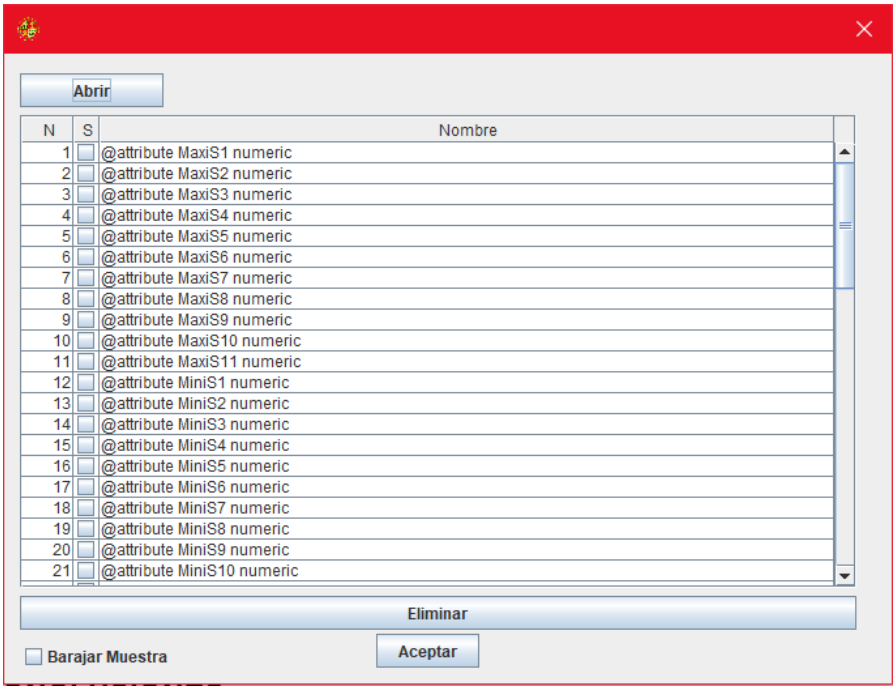



**Ilustración 57: Ventana de carga de archivo**

En ella, se puede cargar un archivo mediante el botón abrir. Una vez cargado, los atributos se muestran en el área central de una tabla. Ésta está formada por los atributos, de los que se especifica su nombre y tipo, el número que le corresponde a cada uno, y un checkbox donde se puede seleccionar si el atributo se va a eliminar o no.

Ésta pantalla también implementa, además del filtro de eliminación, un filtro que baraja las instancias que componen la muestra, para evitar que haya un conjunto de instancias ordenado por clase, que se puede habilitar o deshabilitar mediante el correspondiente checkbox.

## 6.2. Clasificación de las instancias

La interfaz gráfica de los diferentes clasificadores tiene el aspecto de la ilustración 58. Los resultados de clasificación se muestran en la región central de la ventana, mientras que los parámetros de ajuste se pueden cambiar presionando el correspondiente botón de Settings, de la misma.

Cada clasificador puede ser evaluado sobre el set de datos, con la opción entrenar, pero también podrá ser evaluado sobre un set de test que se cargue después

de generar el modelo, mediante la selección de la opción evaluar. Asimismo, la opción

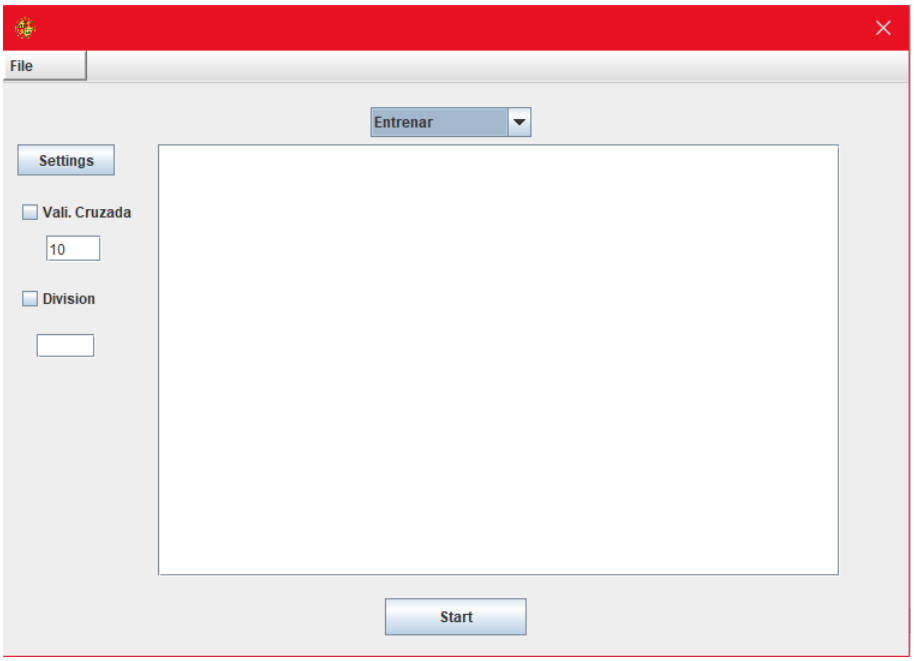



**Ilustración 58:Interfaz clasificadores**

de la evaluación sobre el set de datos sobre el que se construye el modelo, se puede realizar de varias maneras: bien con validación cruzada, con posibilidad de elegir las subdivisiones con las cuales se va a implementar, si se selecciona el checkbox correspondiente, o bien mediante la división de éste set de datos en set de entrenamiento y set de validación, eligiéndose el checkbox correspondiente y especificando qué tanto por ciento se usará para el entrenamiento,

Si no hay ningún set de datos cargado sobre el que construir el modelo, el programa lanzará el mensaje de error correspondiente (ver ilustración 59).

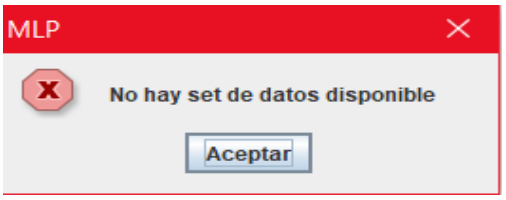



**Ilustración 59: error por falta de set de datos**

Por el contrario, si la herramienta dispone de un set de datos sobre el cual construir y evaluar el modelo, se presentan los resultados como se puede ver en la ilustración 59.

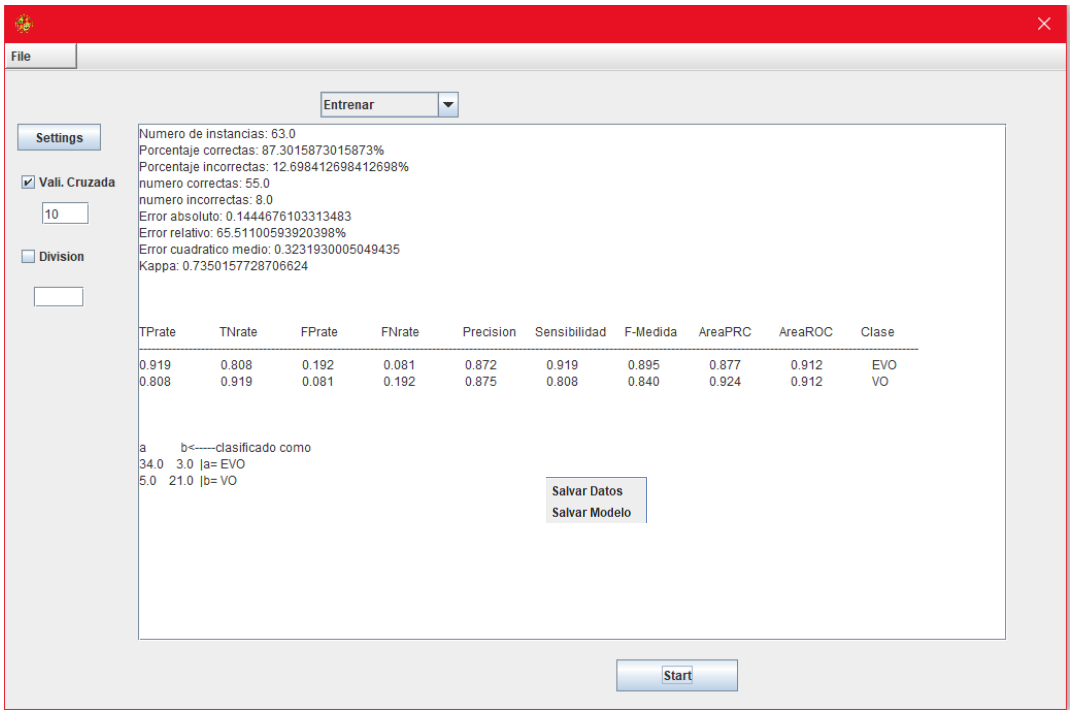



**Ilustración 60: Resultados evaluación**

Los resultados mostrados de la evaluación (ver ilustración 60), a su vez, se pueden guardar en un archivo de texto mediante el uso del correspondiente botón del menú emergente que se despliega al pulsar con el botón derecho sobre el área de texto, o bien haciendo uso del botón salvar datos que se encuentra al desplegar las opciones de File.

No solo se pueden guardar los datos de la evaluación, sino que también se puede guardar el modelo, haciendo uso de los mismos menús, en los que se encuentra el correspondiente botón de salvar modelo.

Ambos se podrán salvar en la carpeta que el usuario desee.

Para configurar los parámetros de cada clasificador, se tienen las siguientes ventanas:

Para la configuración del algoritmo j48, se tiene la ventana de diálogo que se puede ver en la ilustración 62, que, por similitud con el algoritmo PART, será muy parecida a la de éste último, con excepción de la ausencia del filtro de Laplace.

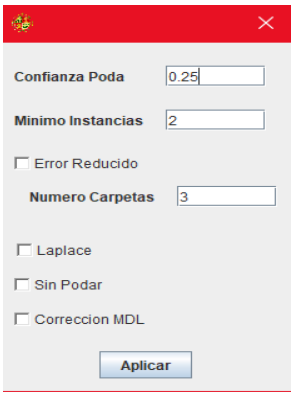



**Ilustración 62: opciones J48**

Del mismo modo, la ventana para ajustar los parámetros del algoritmo KStar (ver ilustración 61), permite el ajuste del parámetro de mezclado, el establecimiento de los valores perdidos (a para establecer el filtro de la media, d para borrar las instancias con valores perdidos, m para el uso del filtro de máxima diferencia, y n para usar la normalización) o habilitar y deshabilitar la mezcla entrópica.

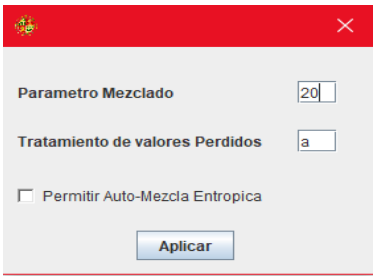



**Ilustración 61: Opciones K***

El algoritmo SimpleLogistic se configura mediante la ventana de diálogo de la ilustración 63, teniendo la posibilidad de configuración de las diversas paradas del algoritmo LogitBoost, así como el establecimiento del parámetro β.

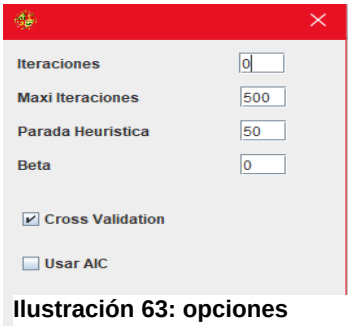



**Ilustración 63: opciones SimpleLogistic**

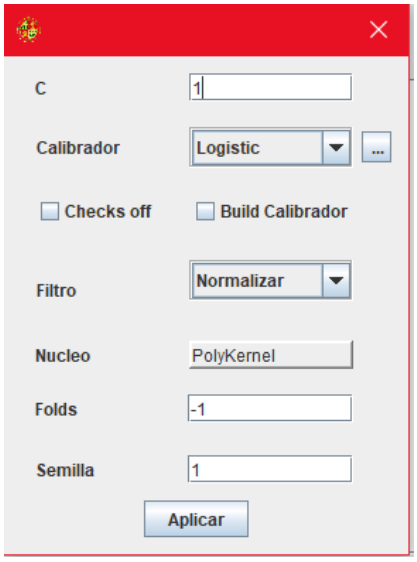



**Ilustración 64: opciones SMO**

El algoritmo SMO posee varias ventanas de configuración, una para las configuraciones generales (ver ilustración 64), en la que se puede establecer el núcleo (ver ilustración 65), el parámetro de complejidad, el calibrador, la correspondiente habilitación o deshabilitación de su construcción, las subdivisiones para la validación cruzada, el filtro a aplicar a las instancias y la semilla.

Pulsando sobre el botón representado por tres puntos, se abre una ventana de configuración, que será la ventana de configuración asociada a cada clasificador. Y, del mismo modo, si se pulsa sobre el campo de texto del PolyKernel, se abre la ventana de configuración del mismo donde se puede establecer el exponente y donde también se puede habilitar la opción que permite escoger los valores de orden menor.


**Ilustración 65: opciones kernel**

El establecimiento de los parámetros que configuran la creación de un perceptrón multicapa, se realiza mediante la ventana de diálogo de la ilustración 66,

en la que la configuración de las capas ocultas, se puede realizar con los valores preestablecidos explicados con anterioridad en el punto referente a la implementación

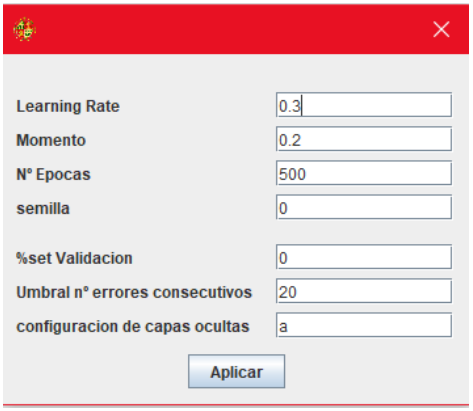



**Ilustración 66: opciones MLP**

del algoritmo (t,a,i,o), pero también se puede realizar de forma libre, añadiendo el número de neuronas y capas ocultas que se desee.

Por su parte, el algoritmo oneR es el único que no dispone de una ventana de configuración, dado que sólo se dispone de un parámetro de modificación.

En lugar de poseer su propia ventana de configuración, la propia ventana de clasificación servirá para el establecimiento de dicho parámetro a ajustar (ver

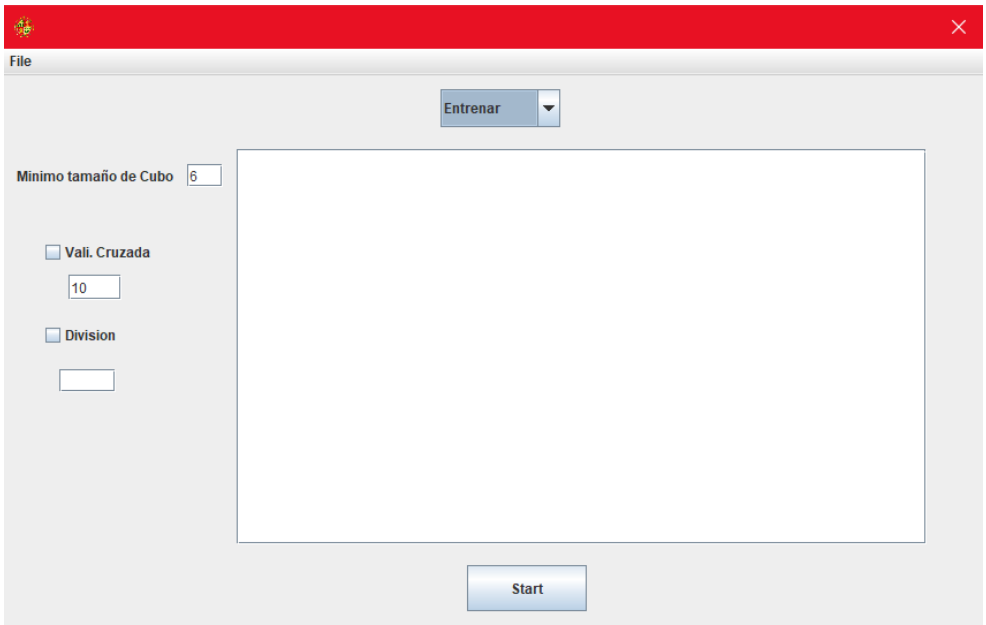



**Ilustración 67: ventana oneR**

ilustración 67).

# 7. CONCLUSIONES

Como se ha podido ver a lo largo de éste documento, se han podido completar los puntos que se plantearon como objetivos del trabajo.

Se ha completado la realización de una aplicación que extrae, de forma satisfactoria y con formato adecuado, los datos procedentes de un conjunto de sensores que miden las características de unas determinadas muestras de aceite.

También se han escogido, de entre todos los clasificadores disponibles en el entorno de análisis de conocimiento de WEKA, aquellos que resolvían de un modo más efectivo la tarea de clasificación que se aborda en el presente trabajo, es decir, distinguir entre muestras de aceite cuáles cumplen los estándares para ser de oliva virgen extra y cuáles no (EVO o VO).

Como se ha visto en el punto en el que se ofrecían los resultados de los clasificadores, en el cual se han ido ajustando los parámetros de los mismos con objetivo de encontrar los de resultados óptimos, los algoritmos que mejor desempeño ofrecían eran el perceptrón multicapa y el SMO, ambos algoritmos utilizan funciones matemáticas para crear regiones de clasificación de las diferentes clases mediante hiperplanos.

Para cada uno de estos clasificadores escogidos, se han estudiado sus fundamentos, derivando esto en una mejor comprensión acerca del rendimiento ofrecido por los mismos, y de las consecuencias que tienen sobre éste rendimiento la modificación de los parámetros ajustables.

Una vez realizado éste estudio, se han integrado los clasificadores seleccionado en una herramienta, desarrollada en Java, que tiene la capacidad, tanto de crear los archivos de entrada a los clasificadores, como de construir los modelos a partir de los mismos y evaluar su rendimiento sobre distintos sets de prueba.

Cabe mencionar, por otra parte, que un estudio más efectivo acerca de las capacidades clasificadoras de los algoritmos hubiera derivado de la disposición de un mayor número de muestras, con objeto de tener las suficientes instancias como para crear un set de entrenamiento lo suficientemente grande como para permitir al algoritmo aprender, y un set de prueba sobre el que probar éste, en vez de utilizar la técnica de la validación cruzada, aunque ésta sea bastante fiable.

# ANEXO I: MANUAL DE USUARIO DE LA HERRAMIENTA CLASIFICADORA

## 1. Introducción

En la actualidad, la tecnología está estrechamente relacionada con todos los ámbitos de la industria. Así pasa también, por tanto, con la elaiotecnia, es decir, la industria que se encarga de la elaboración del aceite.

Con el fin de mecanizar el proceso de control de calidad del aceite, el grupo de investigación en Robótica, Automática y Visión por Computador de la Universidad de Jaén ha creado una nariz electrónica capaz de llevar a cabo éste proceso, mediante la medición de las componentes volátiles del mismo.

Para que éste proceso derive en la correcta clasificación del aceite de oliva, distinguiendo el de calidad virgen extra, se requiere de un procesamiento de los datos obtenidos por los sensores que componen la nariz, así como de unos algoritmos que sean capaces de clasificar correctamente las muestras a partir de los datos procesados.

La herramienta aquí presentada se encarga de éste procesamiento y clasificación de las muestras de aceite de oliva sensadas por el dispositivo E-NOSE.

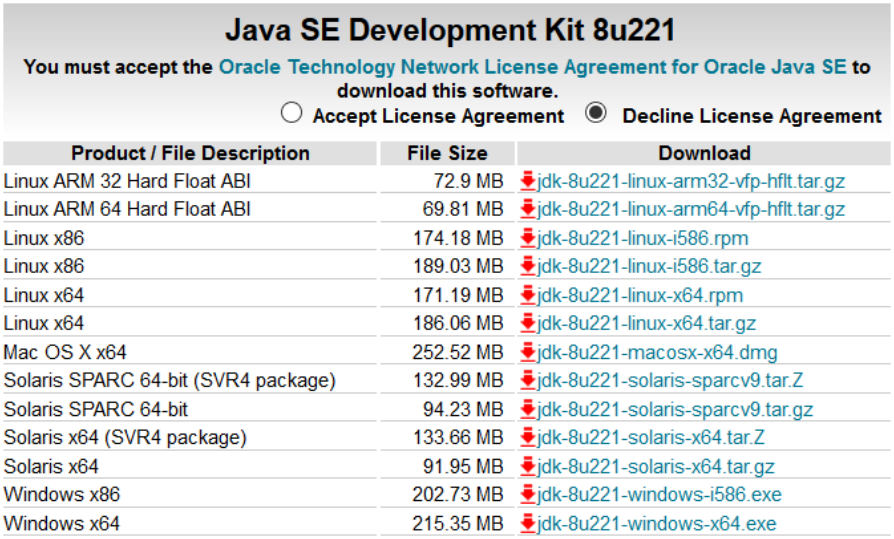
## 2. Sistema operativo compatible e instalación

Para la ejecución de la aplicación, se dispone de un archivo .jar que se ejecutará directamente, sin necesidad de instalación previa.

Para que el sistema sea capaz de ejecutar éste archivo, el PC tiene que tener preinstalada la plataforma JAVA Standard Edition. Ésta se puede descargar de la


**Ilustración 68: Descarga JAVA SE I**

página web oficial [29], y es compatible tanto para Windows, como para Mac Os X o Linux. Para ello, se hace click sobre el botón download rodeado de rojo de la



**Java SE Development Kit 8u221**

You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 32 Hard Float ABI | 72.9 MB | jdk-8u221-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 69.81 MB | jdk-8u221-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 174.18 MB | jdk-8u221-linux-i586.rpm |
| Linux x86 | 189.03 MB | jdk-8u221-linux-i586.tar.gz |
| Linux x64 | 171.19 MB | jdk-8u221-linux-x64.rpm |
| Linux x64 | 186.06 MB | jdk-8u221-linux-x64.tar.gz |
| Mac OS X x64 | 252.52 MB | jdk-8u221-macosx-x64.dmg |
| Solaris SPARC 64-bit (SVR4 package) | 132.99 MB | jdk-8u221-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 94.23 MB | jdk-8u221-solaris-sparcv9.tar.gz |
| Solaris x64 (SVR4 package) | 133.66 MB | jdk-8u221-solaris-x64.tar.Z |
| Solaris x64 | 91.95 MB | jdk-8u221-solaris-x64.tar.gz |
| Windows x86 | 202.73 MB | jdk-8u221-windows-i586.exe |
| Windows x64 | 215.35 MB | jdk-8u221-windows-x64.exe |

**Ilustración 69: Descarga JAVA SE II**

ilustración 69.

Una vez hecho eso, aparecerá la ventana de la ilustración 69. En ella, se marcará la opción Accept License Agreement y se seleccionará el sistema operativo que posee el equipo en el que se va a instalar la máquina virtual Java.

Una vez descargado, se abre el archivo ejecutable y se siguen los siguientes pasos:

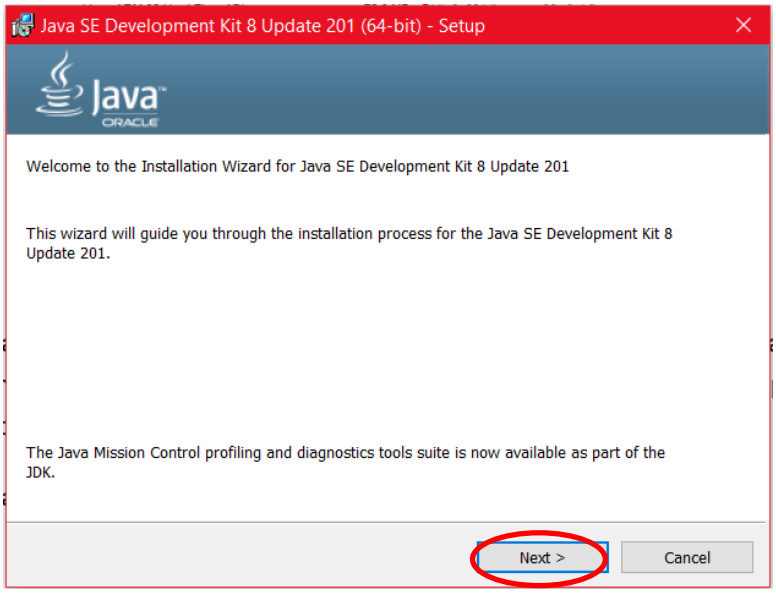



**Ilustración 70: Instalación JAVA SE I**

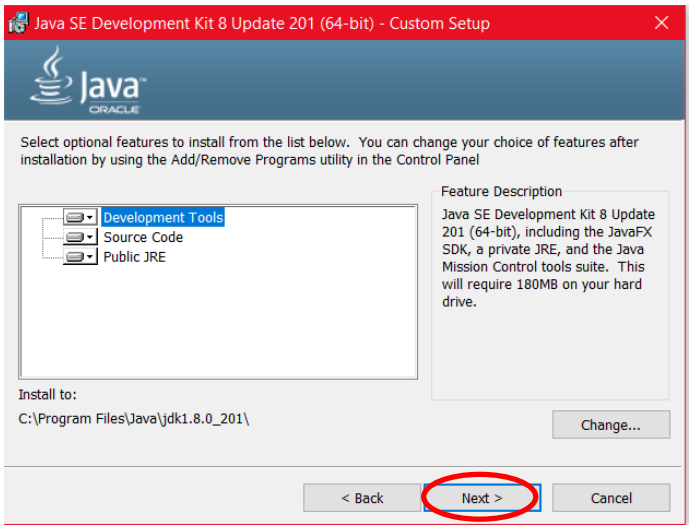



**Ilustración 71: Instalación JAVA SE II**

En la ventana de la ilustración 71, se puede cambiar la carpeta de instalación mediante el botón change, y una vez conforme con el directorio seleccionado, se da a next.

Con éstos pasos, se completará la instalación de la máquina virtual de Java, que permitirá la ejecución del programa.

## 3. Descripción y uso de la aplicación

El programa consta de varias partes, cada una de las cuales están dedicadas a los diferentes procesos que cubre ésta, pudiendo dividirse éstas en dos grandes grupos atendiendo a su funcionalidad: por un lado, las interfaces dedicadas a la creación y el manejo de archivos, y por otro, las inherentes a la tarea de clasificación


**Ilustración 72: pantalla principal**

abordada.

### 3.1. Creación de los Archivos.

Para la creación de los archivos, se tiene la ventana de la ilustración 73, accesible haciendo click sobre la pestaña archivo-crear archivo, o mediante el atajo de teclado Ctrl+C, de la ilustración 72.

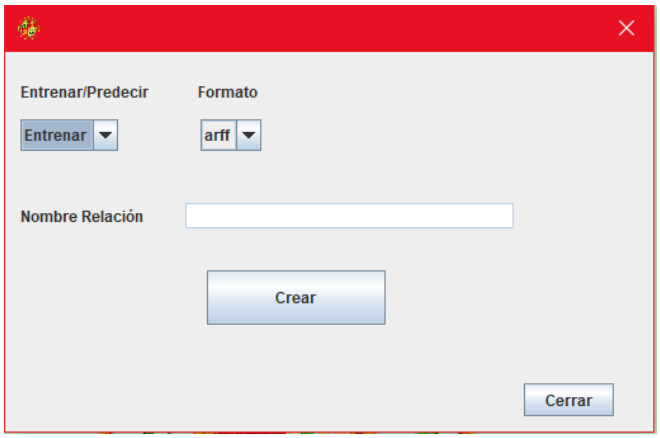



**Ilustración 73: Creación de Archivos**

Para la creación, primero se tendrá que seleccionar si el archivo va a ser de entrenamiento o de test, y el formato del mismo. En caso de seleccionar arff, se tendrá que rellenar el campo Nombre Relación.

Una vez hecho esto, y presionado el botón crear, se abre el cuadro de diálogo que permite seleccionar las muestras de que va a componerse el archivo. El sistema guiará al usuario a través de mensajes informativos como el de la Ilustración 75.


**Ilustración 74: Mensaje informativo**

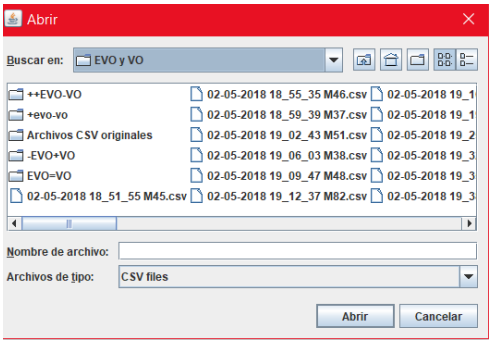



**Ilustración 75: Seleccionador Archivos**

Seleccionados ya los archivos que formarán las instancias del archivo de salida, se darán dos situaciones diferentes, en función de si el archivo va a ser de entrenamiento o de test.

Si es de entrenamiento, se indicará que se deben escoger los archivos que contienen la información de la catalogación de las muestras. Para ello, se tendrán dos archivos, uno con las muestras catalogadas como EVO (el nombre del archivo deberá contener dicha palabra), y otro con las VO. Esto se realizará mediante otro selector de archivo (ver ilustración 76).

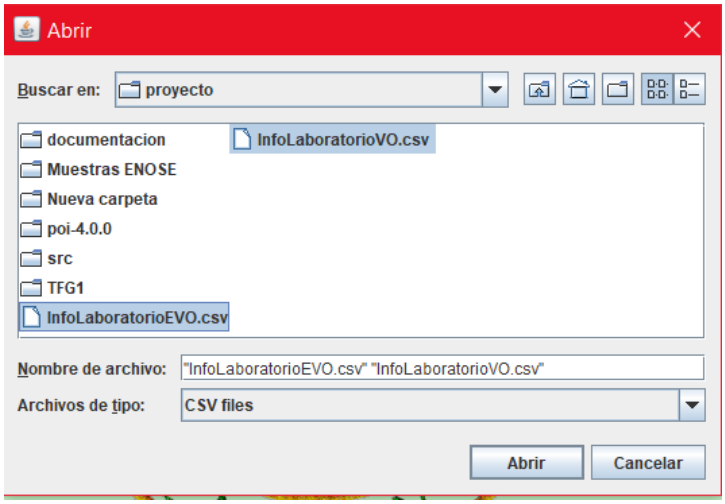



**Ilustración 76: Creación archivo de entrenamiento**

Si el archivo es de test, o si ya se han seleccionado los archivos donde están seleccionados en caso de que el archivo sea de entrenamiento, el programa requerirá al usuario que especifique la carpeta donde desea que se guarde éste archivo mediante la correspondiente ventana de guardado (ver ilustración 77).

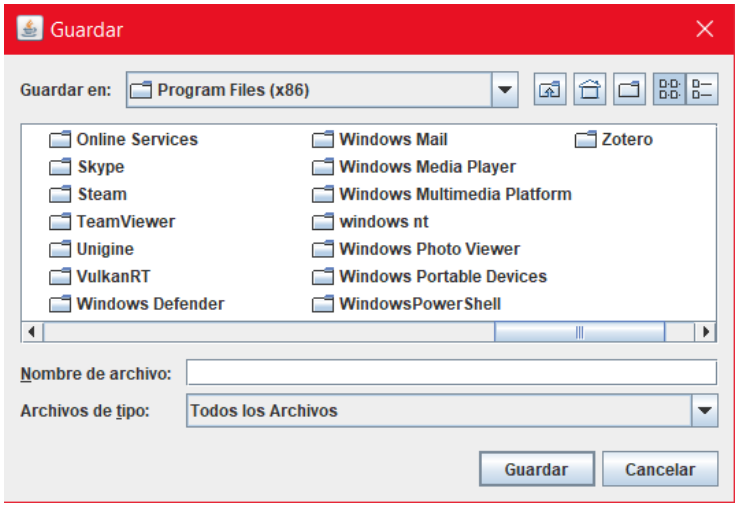



**Ilustración 77: Creación de Archivos II**

Una vez finalizado el proceso, el archivo estará ya creado en la carpeta especificada.

### 3.2. Carga de los Archivos

Para la carga de los archivos de entrenamiento, se dispone de la interfaz de la ilustración 78, a la que se accede mediante la pantalla principal, haciendo click sobre Cargar archivo, dentro de la pestaña Archivo.

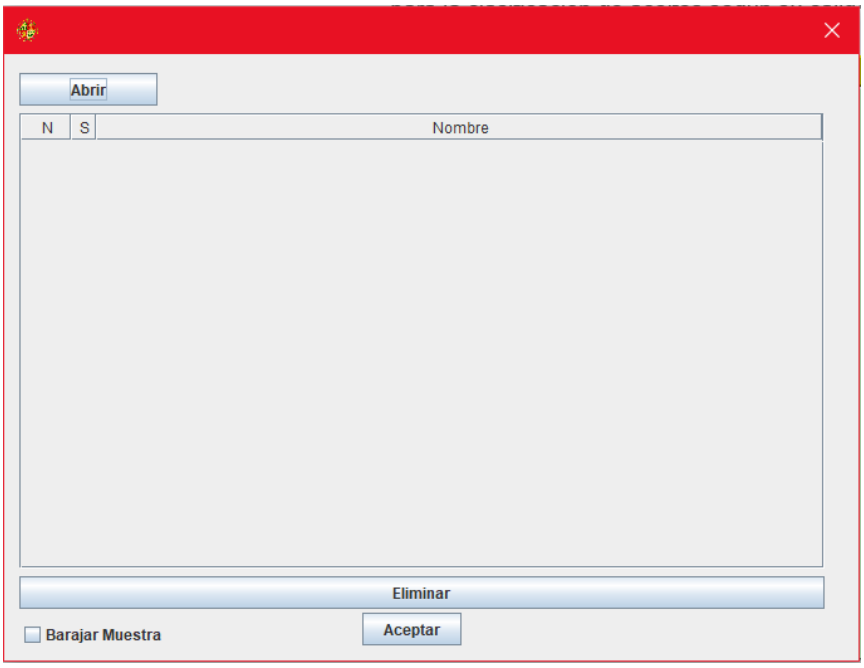



**Ilustración 78: Cargador de los Archivos**

Una vez dentro, se hace click sobre Abrir, abriéndose el cuadro de diálogo que permite la selección del archivo deseado (ver ilustración 79).

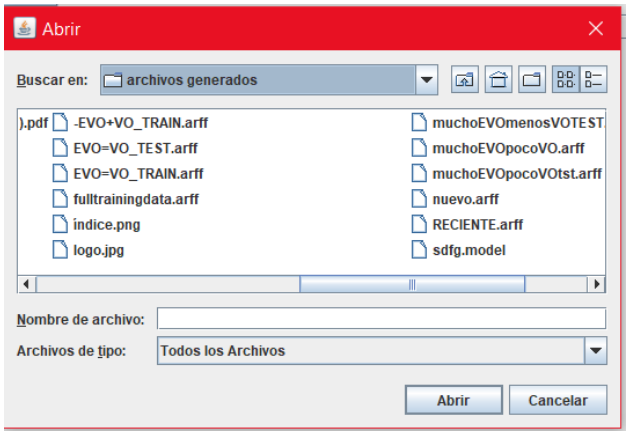



**Ilustración 79: Seleccionando archivo**

Una vez seleccionado, se presiona abrir y los datos se cargarán sobre la tabla de la interfaz de usuario (ver ilustración 80).

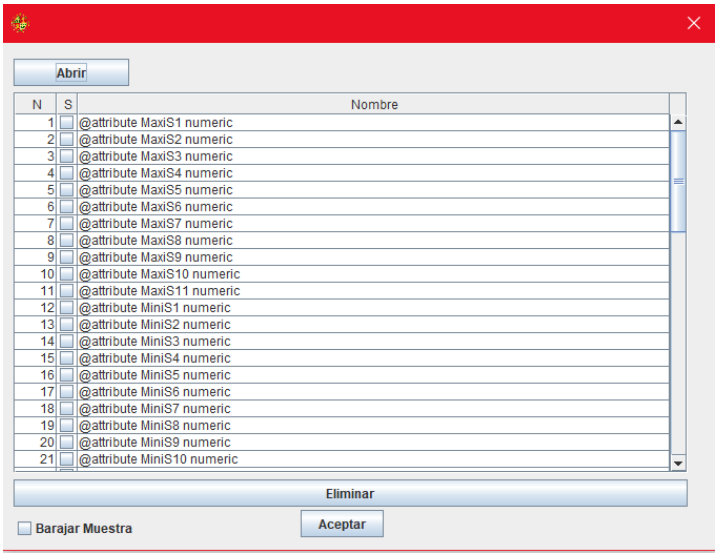



**Ilustración 80: Archivo Cargado**

Si se quiere eliminar un atributo, se marca el checkbox correspondiente al atributo a eliminar, de la columna etiquetada como “S”, y se presiona sobre el botón eliminar. Asimismo, si se desea barajar la muestra, se marcará el correspondiente checkbox sito en la esquina inferior izquierda.

Una vez terminada de tratar la muestra, se hace click en aceptar para guardar cambios y salir de la ventana.

### 3.3. Clasificación

Para acceder a los algoritmos clasificadores, se presiona sobre la pestaña clasificar de la pantalla principal del programa. (ver ilustración 81). Una vez hecho esto, se desplegará una lista con los clasificadores disponibles.

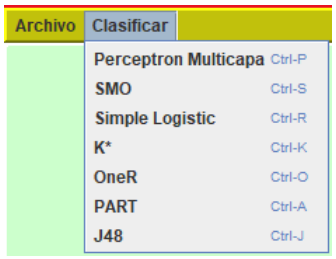



**Ilustración 81:**
**Clasificadores disponibles**

Éstos, a su vez, están disponibles también mediante el correspondiente atajo de teclado.

El proceso a seguir para clasificar las muestras, es similar para cualquier clasificador escogido: haciendo click sobre el clasificador seleccionado, se abre la interfaz que permite configurar éste y visualizar los resultados (ver ilustración 82).

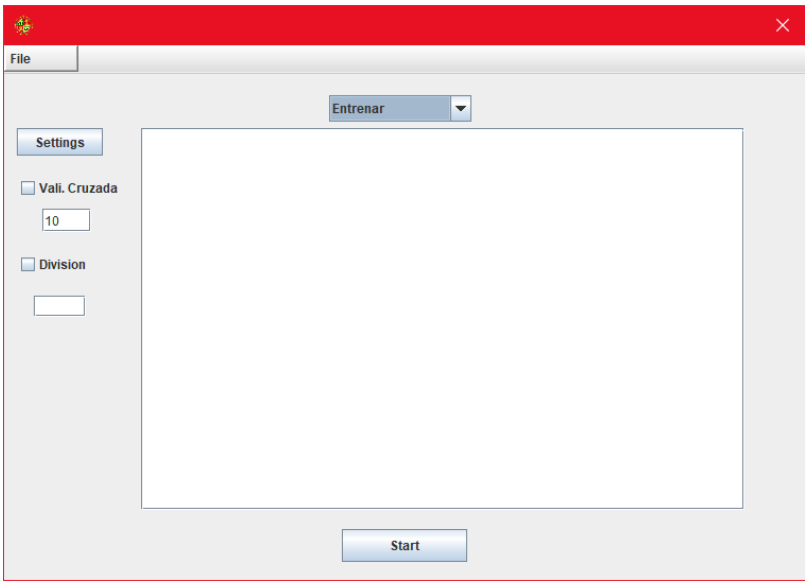



**Ilustración 82: interfaz clasificador**

Para la construcción del modelo clasificatorio, se seleccionará la opción “Entrenar” del combobox. Acto seguido, se configurarán los parámetros del clasificador (ver ilustración 83), mediante el botón Settings.

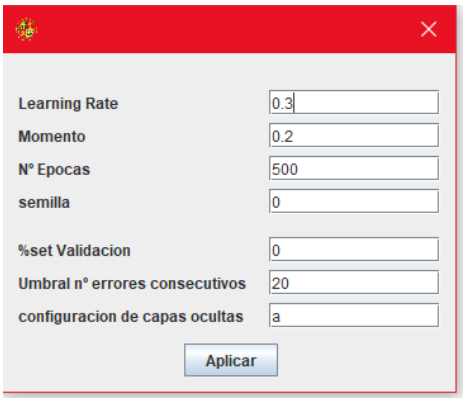



**Ilustración 83: Opciones Perceptrón Multicapa**

Una vez ajustados los parámetros, se presiona en el botón Aplicar para guardar los cambios realizados sobre la construcción del modelo.

Ajustados ya los parámetros que van a regir la construcción del modelo con el algoritmo seleccionado, se pueden elegir diferentes modalidades para la evaluación sobre el set de entrenamiento, del modelo creado.

Para elegir el método de validación cruzada, se selecciona el checkbox asociado y se establece un número de iteraciones del mismo. Si no se especifica ninguno, se inicializará a 10 por defecto.

Si, por el contrario, se prefiere dividir el set de datos en set de entrenamiento y set de validación, se seleccionará el checkbox correspondiente a “División” y se especificará un número entero, que será el porcentaje del set de datos que se usará para entrenamiento.

Una vez ajustado todo, se presiona el botón “Start”. Cuando el modelo haya sido creado, se notificará mediante el correspondiente mensaje de información, y se escribirán los resultados de la evaluación sobre el área habilitada a tal efecto (ver

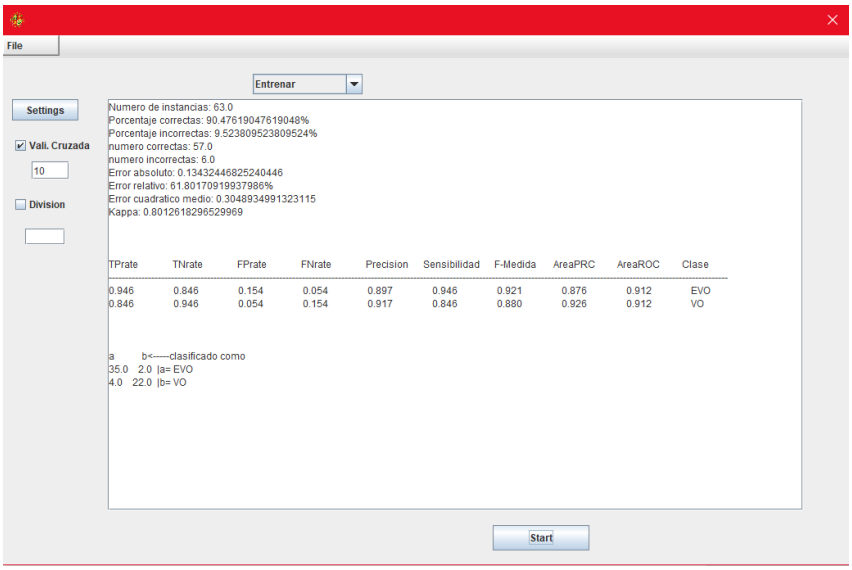



**Ilustración 84: Resultado Modelo Perceptrón Multicapa**

ilustración 84).

Una vez construido el modelo, se puede realizar una comprobación de su funcionamiento mediante un set de prueba, o simplemente la clasificación de instancias nuevas, mediante la selección de la opción “Evaluar”, del ComboBox.

Si se elige ésta opción y se presiona el botón “Start”, se abrirá un cuadro de diálogo con el cual se puede seleccionar el archivo de entrada que contiene las muestras a predecir (ver ilustración 85).

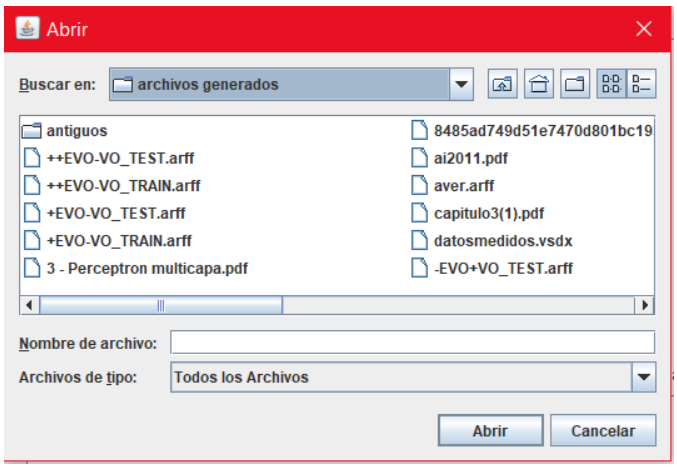



**Ilustración 85: Selector de archivo de test**

Una vez seleccionado y presionado el botón abrir, se procede a la tarea de predicción y, nuevamente, los resultados se podrán ver en el área habilitada.

Si se desea evaluar el set de prueba usando un modelo anterior, se puede cargar éste mediante la opción “Cargar Modelo”, situada en el menú “File”, o mediante el atajo de teclado Ctrl+M.

### 3.4. Guardado de los Resultados

Una vez construido el modelo, y realizada la tarea de clasificación, se pueden guardar los resultados y el modelo, de varias formas: Si se hace click derecho sobre el área de texto que contiene los resultados, se abrirá un menú emergente (ver

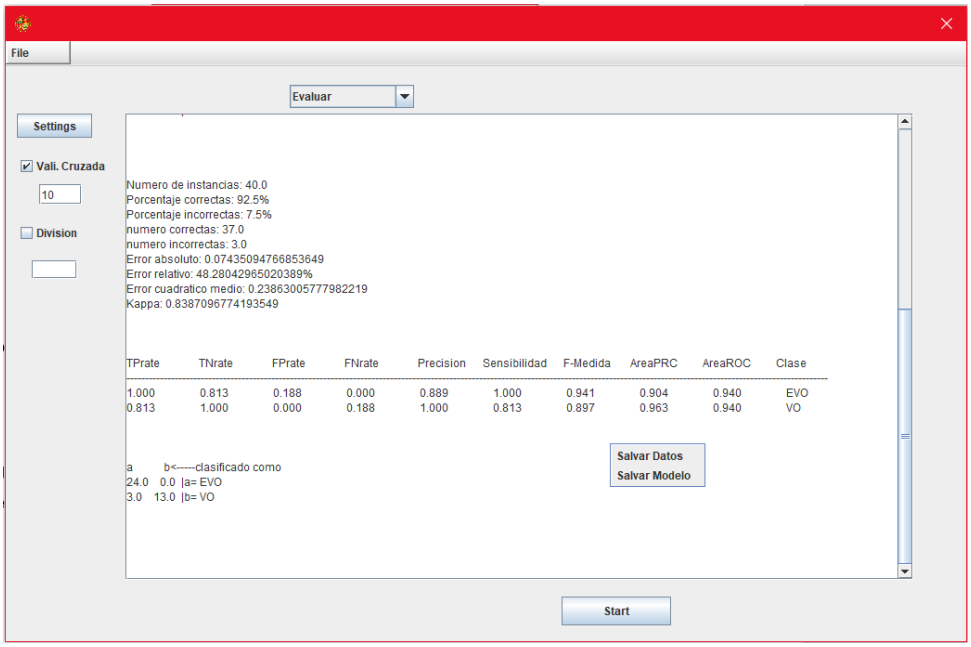



**Ilustración 86: Salvado de datos I**

ilustración 86).

Para guardar el modelo, se presiona sobre el botón “Guardar Modelo”, bien del menú emergente o bien del menú que se despliega al presionar sobre el botón “File”.

Una vez presionado, se abre un cuadro de diálogo similar a la ilustración 77, que permite seleccionar el directorio de guardado, y especificar el nombre del archivo.

De una forma análoga se puede proceder al guardado del modelo creado, seleccionando la opción Salvar Modelo sobre el menú emergente, sobre el menú que se despliega al presionar el botón “File”, o tecleando Ctrl+S.

# Índice de Figuras

## Referencias

[1] López García, C., & Fernández Veiga, M. (2002). *Teoría de la Información y Codificación*

[2] (2004). Recuperado de https://cs.uns.edu.ar/~ldm/mypage/data/ss/info/teoria_de_la_informacion1.pdf

[3] Quinlan, J. (1992). *C4.5 - programs for machine learning*. San Mateo, Calif: Kaufmann.

[4] Frank, E., & Witten, I. (1998). *Generating accurate rule sets without global optimization*. Hamilton, N.Z.: Dept. of Computer Science, University of Waikato.

[5] Cleary, J., & Trigg, L. (1998). *K*: An Instance-based Learner Using an Entropic Distance Measure*. Hamilton, N.Z.: Dept. of Computer Science, University of Waikato.

[6] Camarero Rioja, L., Almazán Llorente, A., & Mañas Ramírez, B. *Regresión Logística: Fundamentos y aplicación a la investigación sociológica* [Ebook].

[7] StatQuest with Josh Starmer. (2018). *Logistic Regression Details Pt 2: Maximum Likelihood* [Video]. Recuperado de https://www.youtube.com/watch?v=BfKanl1aSG0

[8] WekaMOOC. (2013). *Data Mining with Weka (4.4: Logistic regression)* [Video]. Recuperado de https://www.youtube.com/watch?v=ThmZU3dTIDo

[9] Friedman, J., Hastie, T., & Tibshirani, R. (1998). *Additive Logistic Reggresion: A statistical view of boosting*.

[10] Landwehr, N., Hall, M., & Frank, E. (2004). *Logistic Model Trees*.

[11] Platt, J. (2000). *Fast Training of Support Vector Machines using Sequential Minimal Optimization* [Ebook].

[12] Isasi Viñuela, P., & Galván León, I. (2008). *Redes de neuronas artificiales*. Madrid, [etc.]: Pearson-Prentice Hall.

[13] Rojas, R. *Neural Networks*

[14] García, J. (2015). *4c.- Redes Neuronales: Fácil y desde cero* [Video]. Recuperado de https://www.youtube.com/watch?v=NqwTFtOQMoM&list=PLAnA8FVrBl8AWkZmbswwWiF8a_52dQ3JQ&index=7&t=609s

[15] García, J. (2015). *4d.- Redes Neuronales: Fácil y desde cero* [Video]. Recuperado de https://www.youtube.com/watch?v=toAPgzWDZJE&list=PLAnA8FVrBl8AWkZmbswwWiF8a_52dQ3JQ&index=7

[16] Holte, R. (1993). *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets* [Ebook].

[17] Chesñevar, C. (2009). *Datamining y Aprendizaje Automatizado 05 – Aprendizaje de Conjuntos de Reglas* [Ebook]. Departamento de Cs. e Ing. de la ComputaciónUniversidad Nacional del Sur. Recuperado de http://cs.uns.edu.ar/~cic/dm2009/downloads/transparencias/05_dm%20(Learning_rules).pdf

[18] CSVLoader. (2019). Recuperado de http://weka.sourceforge.net/doc.dev/weka/core/converters/CSVLoader.html

[19] ArffLoader. (2019). Recuperado de http://weka.sourceforge.net/doc.dev/weka/core/converters/ArffLoader.html

[20] KStar. (2019). Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/lazy/KStar.html

[21] Recuperado de http://cruise.eecs.uottawa.ca/git/umple/Umplificator/UmplifiedProjects/weka-umplified-0/src/main/java/weka/classifiers/lazy/KStar.java

[22] SimpleLogistic. Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SimpleLogistic.html

[23] PolyKernel. (2019). Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/supportVector/PolyKernel.html

[24] SMO. Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html

[25] MultilayerPerceptron. Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html

[26] OneR. Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/OneR.html

[27] J48. Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

[28] PART. Recuperado de http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/PART.html

[29] Descargas de Java SE. (2019). Recuperado de https://www.oracle.com/technetwork/es/java/javase/downloads/index.html