



UNIVERSIDAD
DE CÓRDOBA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Implementación de una interfaz para el algoritmo
NSLVOOrd en la biblioteca ORCA

Manual Técnico

Autor

Federico García-Arévalo Calles

Directores

Pedro Antonio Gutiérrez Peña

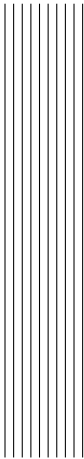
Juan Carlos Gámez Granados

ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba



ESCUELA POLITÉCNICA SUPERIOR

—
Córdoba, Junio de 2020



ÍNDICE GENERAL

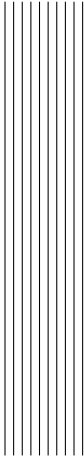
Índice General	III
Índice de Figuras	VII
Índice de Tablas	IX
I Introducción	1
1. Introducción	3
2. Definición del Problema	5
2.1. Identificación del Problema Real	5
2.2. Identificación del Problema Técnico	6
2.2.1. Funcionamiento	6
2.2.2. Entorno	7
2.2.3. Vida Esperada	8
2.2.4. Ciclo de Mantenimiento	8
2.2.5. Fiabilidad	8

3. Objetivos	9
3.1. Incluir NSLVOrd en ORCA	9
3.2. Incluir Ficheros de Formato Weka en ORCA	10
3.3. Visualización de las Reglas Difusas de NSLVOrd	10
4. Antecedentes	11
4.1. Clasificación Ordinal	11
4.2. ORCA	11
4.3. NSLVOrd	12
4.4. JFML	12
4.5. WEKA	12
4.6. Java	13
5. Restricciones	15
5.1. Factores Dato	15
5.2. Factores Estratégicos	16
6. Recursos	19
6.1. Recursos Humanos	19
6.2. Recursos Hardware	20
6.3. Recursos Software	20
II Análisis del Sistema	21
7. Casos de Uso	23
7.1. Ejecutar un Experimento	24
7.2. Exportar las Reglas	25
7.3. Visualizar las Reglas	25
7.4. Entrenamiento y Test	26

ÍNDICE GENERAL

7.5. Leer Datos	27
8. Especificación de Requisitos	29
8.1. Requisitos de Usuario	29
8.2. Requisitos del Sistema	30
8.2.1. Requisitos no Funcionales	30
8.2.2. Requisitos Funcionales	30
 III Modelado del Sistema	 33
 9. Arquitectura del Sistema	 35
9.1. Módulos	36
9.2. Interacciones del Sistema	37
 10. Diagramas de Secuencia	 39
 IV Diseño del Sistema	 43
 11. Diagramas de Clases	 45
11.1. ORCA	46
11.2. Read File	48
11.3. Algorithms	50
11.4. NSLVOrd	51
11.5. Rule View	52
11.5.1. ORCA	52
11.5.2. Java	53
11.6. JFML	55
11.6.1. ORCA	55
11.6.2. Java	56

12.Diseño de la Interfaz Gráfica	61
V Cierre	63
13.Pruebas	65
13.1. Datos para las Pruebas	66
13.2. ORCA	68
13.3. Read File	69
13.3.1. Tipo de Fichero Valido	69
13.3.2. Leer datos	71
13.4. Algorithms	76
13.5. NSLVOrd	77
13.6. Rule View	78
13.7. JFML	80
13.8. Sistema	85
13.9. Errores Durante el Desarrollo	88
14.Conclusiones y Futuras Mejoras	91
14.1. Objetivos del Problema	91
14.2. Nivel Personal	92
14.3. Futuras Mejoras	92
Bibliografía	93



ÍNDICE DE FIGURAS

9.1. Estructura modular del sistema	37
9.2. Estructura modular del sistema	38
10.1. Diagrama de secuencia para la ejecución de un experimento.	40
10.2. Diagrama de secuencia de la exportación de las reglas.	41
10.3. Diagrama de secuencia de la visualización de las reglas.	42
12.1. Esquema de la interfaz gráfica.	62
12.2. Ejemplo de la interfaz gráfica.	62
13.1. Prueba para comprobar si el tipo de ficheros es valido	70
13.2. Prueba del módulo <i>Rule View</i>	79



ÍNDICE DE TABLAS

7.1. CU-01: Ejecutar un experimento	24
7.2. CU-02: Exportar las reglas	25
7.3. CU-03: Visualizar las reglas	25
7.4. CU-04: Realizar entrenamiento	26
7.5. CU-05: Realizar test	27
7.6. CU-06: Leer datos	27
8.1. Ejemplo de convertir una variable categórica en <i>one-hot</i> . . .	31
11.1. Ejemplo general para la especificación de clases	45
11.2. Clase Utilities	46
11.3. Clase Dataset	46
11.4. Clase Experiment	47
11.5. Clase TFGFileReadClass	48
11.6. Clase ReadFileCommon	48
11.7. Clase matlab	49
11.8. Clase weka	49
11.9. Clase Algorithm	50

11.10Clase NSLVOrd	50
11.11Clase NSLVOrdJava	51
11.12Clase RuleSystem	52
11.13Clase RulesVisual	52
11.14Clase VisualRules	53
11.15Clase Rule	53
11.16Clase ConditionCategoric	54
11.17Clase ConditionFuzzyLogic	54
11.18Clase RulesExport	55
11.19Clase FuzzySystemType	56
11.20Clase FuzzyInferenceSystem	56
11.21Clase KnowledgeBaseType	57
11.22Clase RuleBaseType	58
11.23Clase MamdaniRuleBaseType	58
11.24Clase JFML	59
11.25Clase ExportPMML	59
13.1. Ejemplo general para una prueba	65
13.2. Datasets de prueba.	66
13.3. Prueba OR.01: Valor de 'archive'	68
13.4. Prueba OR.02: Acceso de Directorios	68
13.5. Prueba RF.01: Tipo de Fichero Válido	69
13.6. Prueba RF.02: Lectura de Datos	71
13.7. Prueba RF.03: Lectura de la Cabecera	71
13.8. Prueba RF.04: Conversión de los Datos Categóricos	72
13.9. Prueba RF.03.1: Cabecera correcta	72
13.10Prueba RF.03.2: Cabecera sin @data	73
13.11Prueba RF.03.3: Cabecera con atributos repetidos	74

ÍNDICE DE TABLAS

13.12Prueba RF.03.4: Cabecera con un atributo erroneo	75
13.13Prueba AL.01: Entrenamiento y Test	76
13.14Prueba AL.02: Configuración de datos	76
13.15Prueba AL.03: Procesar salidas	77
13.16Prueba NSL.01: Adaptación de las Entradas y Salidas	78
13.17Prueba RV.01: Visualizar el Sistema de Reglas	78
13.18Prueba RV.01: Visualizar el Sistema de Reglas	80
13.19Prueba SIS.01: ORCA - Read File	85
13.20Prueba SIS.02: ORCA - Algorithms	85
13.21Prueba SIS.03: Algorithms - NSLVOrd	86
13.22Prueba SIS.04: Algorithms - Rule View	86
13.23Prueba SIS.05: Algorithms - JFML	86
13.24Prueba SIS.06: Sistema	86
13.25Prueba SIS.07: Comparación del MAE	87

Parte I

Introducción



1 Introducción

Durante el desarrollo de un *software*, los desarrolladores pueden verse en la necesidad de incorporar otro *software* externo. Es decir, el *software* que se está desarrollando debe ejecutar otro en algún momento de su ejecución. La razón para hacer esto puede deberse a varios motivos, entre los que pueden encontrarse el querer aprovechar uno ya existente o tener la necesidad de usarlo.

Por un lado, los desarrolladores pueden percatarse que ya existe un *software* capaz de hacer parte del trabajo que debe realizar el suyo y por ello aprovecharlo. Una de las razones para esto puede ser ahorrar el tiempo y dinero que costaría hacer esa parte que se buscaría sustituir, o porque este ya está comprobado que funciona correctamente.

Por otro lado, en ocasiones, los desarrolladores se ven obligados a añadir dicho *software*, ya sea porque se necesita realizar una tarea que no se puede encontrar la forma de resolver, pero este ya la realiza; o porque la naturaleza misma del problema pide que se use otro.

Afortunadamente, esto se tiene en cuenta en los lenguajes de programación de tal forma que facilitan la ejecución de *software* externo.

El objetivo principal de este proyecto está relacionado con lo anteriormente explicado, ya que se pretende incluir el algoritmo NSLVOOrd [5] en un *framework* de Matlab [3] llamado ORCA [11, 7], aunque como se verá en el capítulo 3 se añadirán más objetivos. ORCA trabaja con diferentes métodos de clasificación ordinal y NSLVOOrd es un algoritmo de clasificación ordinal. Es por esto y porque NSLVOOrd ya se puede encontrar desarrollado en un paquete Java [4] por lo que es interesante incorporarlo en el *framework*.



2 Definición del Problema

Este capítulo del documento se centra en identificar el objetivo a alcanzar con este Trabajo de Fin de Grado. Para ello, el capítulo se dividirá en dos apartados en el que se diferenciarán entre el problema real, el propuesto por el cliente; y el problema técnico, aquel que afecta al desarrollo del producto.

2.1. Identificación del Problema Real

El problema real que se plantea en este Trabajo Fin de Grado es la inclusión del algoritmo NSLVOrd [5] en el *framework* ORCA [11, 7], a demás, se ha añadido como complemento a éste, la capacidad de usar datos que estén en ficheros de formato WEKA [8] y la capacidad para representar las reglas de NSLVOrd. Es decir, este trabajo se puede dividir en la resolución de los siguientes problemas:

1. **Inclusión de NSLVOrd en ORCA**

El grupo de investigación AYRNA [2] del Departamento de Informática y Análisis Numérico de la universidad de Córdoba tiene un *framework* propio de Matlab/Octave [3, 9] llamado ORCA que, entre otras cosas, hace uso de múltiples métodos de regresión ordinal. Por esta razón, se ha planteado añadir el algoritmo NSLVOrd, desarrollado por el director de este proyecto Juan Carlos Gámez Granados, ya que éste

es un algoritmo de clasificación ordinal.

2. Leer ficheros WEKA en ORCA

Actualmente, cualquier conjunto de datos de entrenamiento y test que usa ORCA deben estar almacenados en ficheros simples de texto plano que solo contengan datos numéricos, por lo tanto, ORCA no puede trabajar con valores categóricos. Para dar esta nueva funcionalidad a ORCA, se plantea la utilización de ficheros en formato WEKA [8], permitiendo así incorporar datos categóricos.

3. Representar las reglas NSLVOrd

Cuando se hace el aprendizaje en el programa que usa el algoritmo NSLVOrd, todo el proceso se guarda en un fichero de texto plano, que incluye las reglas aprendidas. Es por esto, que en este punto del problema, se plantea el poder visualizar las reglas de una forma que se facilite su entendimiento y, a demás, dar la posibilidad de poder exportar las reglas a formato JFML [12] y PMML [6].

2.2. Identificación del Problema Técnico

Una vez establecido el problema real en el punto anterior, en éste se expondrán el problema técnico del proyecto para establecer los condicionantes del mismo. Para ello, se expondrá una serie de puntos que pretende responder a las preguntas más significativas para la elaboración de este trabajo.

2.2.1. Funcionamiento

Gran parte de este trabajo se va a basar en modificar parte del código de ORCA [11, 7] y NSLVOrd [5] para añadir este segundo al primero, por lo tanto, gran parte del funcionamiento seguirá siendo el mismo y no cambiará con respecto a como es actualmente. A continuación, se mostrará los cambios de funcionamiento más significativos que se han añadido con este trabajo, dividiendo el funcionamiento a nivel de usuario e interno.

2.2. Identificación del Problema Técnico

- **Nivel usuario:** para el usuario que solo interactúa con ORCA, el funcionamiento seguirá siendo el mismo. Sin embargo, tendrá nuevas funcionalidades como son:

1. Obtener los datos de entrenamiento y test de ficheros de formato WEKA [8].
2. Guardar las reglas en ficheros con formato JFML [12] y PMML [6].
3. Representar las reglas visualmente.

- **Nivel interno:** para el funcionamiento de los programas ORCA y NSLVOrd, se tendrían que adaptar para poder incluir los objetivos del trabajo.

Por un lado, en ORCA se cambiará la forma de leer los datos, ya que habrá que incluir una forma de leer los datos como se ha estado haciendo y, además, los datos de ficheros con formato WEKA, permitiendo el uso de valores categóricos.

Por otro lado, para NSLVOrd, ORCA da facilidades para poder incluir nuevos algoritmos en él, pero se tendrá que tener en cuenta la forma en la que se intercambiarán los datos ya que el primero trabaja en Java [4] y el segundo en Matlab/Octave [3, 9]. Además, se añadirá una forma de visualizar y exportar las reglas resultantes del entrenamiento.

2.2.2. Entorno

Debido a que el trabajo que se va a realizar se basa en la incorporación de nuevas funcionalidades en ORCA [11, 7], los usuarios objetivo son aquellos que trabajen con este *framework*. Es decir, los usuarios objetivo son aquellos a los que le interesa trabajar con los diferentes algoritmos de regresión ordinal que ofrece ORCA.

2. Definición del Problema

En este trabajo, se usarán dos entornos de programación impuestos por los programas con los que se van a trabajar. Por un lado, se usará Matlab/Octave [3, 9], ya que es el lenguaje con el que trabaja ORCA. Y por otro lado, se usará Java [4] debido a que NSLVOrd [5] está programado en este lenguaje.

2.2.3. Vida Esperada

En un principio, la vida esperada de esta aplicación puede ser bastante longeva. Esto se debe, sobre todo, a que ORCA [11, 7] es usado principalmente por sus desarrolladores, el equipo de investigación AYRNA [2], aunque también lo tienen a disposición de quien quiera trabajar con él.

2.2.4. Ciclo de Mantenimiento

A priori, no sería necesario un mantenimiento para el resultante de este trabajo. Sin embargo, cabe la posibilidad de hacer futuras mejoras al código realizado en este trabajo, además de actualizar el algoritmo NSLVOrd [5] o la biblioteca JFML [12] si en un futuro reciben nuevas actualizaciones.

2.2.5. Fiabilidad

Para garantizar que el producto final funcione correctamente y sin fallos, se controlará que los datos sean correctos y tengan valores dentro de los límites. En el caso de que un valor no cumpliera con los requisitos, se pondrán un valor por defecto y se avisará al usuario de dicha acción.



3 Objetivos

El objetivo principal de este Trabajo de Fin de Grado consiste en incluir el algoritmo NSLVOrd [5] en ORCA [11, 7]. Además de este objetivo principal, se añadirá otros dos objetivos con el fin de añadir nuevas funcionalidades a ORCA como son el aceptar datos categóricos leídos de ficheros con el formato de WEKA [8] y la capacidad de guardar y representar las reglas aprendidas.

3.1. Incluir NSLVOrd en ORCA

El grupo de investigación AYRNA [2], del Departamento de Informática y Análisis Numérico de la universidad de Córdoba, ha desarrollado un *framework* en Matlab/Octave [3, 9] llamado ORCA [11, 7]. Este *framework* recopila diferentes métodos de regresión ordinal con el objetivo de permitir que los usuarios puedan ejecutar experimentos individuales o pudiendo comparar algoritmos y conjunto de datos, además de que facilita la inclusión de nuevos algoritmos.

Por otro lado, Juan Carlos Gámez Granados (uno de los directores de este trabajo), desarrolló como parte de su Tesis Doctoral el algoritmo NSLVOrd [5]. Este algoritmo esta basado en otro llamado NSLV para permitir la resolución de problemas de clasificación ordinal, ya que NSLV solo resuelve problemas de clasificación nominal.

Debido a lo dicho en los párrafos anteriores, para este objetivo se ha decidido incluir NSLVOrd en ORCA. ORCA facilita la inclusión de nuevos algoritmos haciendo que en una clase se deba incluir las funciones que corresponden a la fase de entrenamiento y a la de test; y Matlab permite incluir código en Java [4], necesario, ya que NSLVOrd está escrito en este lenguaje.

3.2. Incluir Ficheros de Formato Weka en ORCA

Actualmente, ORCA [11, 7] solo puede leer ficheros de datos que contenga únicamente datos numéricos. Debido a que NSLVOrd [5] puede trabajar con datos categóricos, se ha planteado el objetivo de incluir la capacidad de leer este tipo de datos en ORCA [11, 7]. Y para ello, se ha decidido hacer que ORCA pueda leer ficheros tipo WEKA [8]. Debido a que ORCA trabaja únicamente con variables numéricas, este objetivo implica que se deberá realizar un preprocesamiento para convertir las variables nominales en variables numéricas en función del algoritmo utilizado.

3.3. Visualización de las Reglas Difusas de NSLVOrd

Como NSLVOrd [5] es un algoritmo que sirve para el aprendizaje de reglas difusas, se ha establecido como uno de los objetivos permitir la visualización de éstas de tal forma que se buscará que dichas reglas se representen de una forma clara y fácil de entender. Para ello, habrá que buscar una representación visual adecuada, por lo que puede usarse, por ejemplo, Java [4], ya que se tiene pensado usar en el proyecto y permitiría desarrollar una ventana que mostrase las reglas de forma organizada.

Por otro lado, se plantea la idea de poder exportar las reglas. Para ello, se hará uso de una librería Java llamada JFML [12] para exportar las reglas.



4 Antecedentes

4.1. Clasificación Ordinal

El termino ordinal es usado para implicar un cierto orden en un conjunto de elementos y la clasificación es organizar una serie de elementos según un criterio. Cuando se habla de clasificación ordinal, se habla sobre un problema de clasificación supervisada en la que se tiene en cuenta que existe una relación de orden entre las categorías a la hora de predecir a qué categoría pertenece un patrón.

4.2. ORCA

ORCA (Ordinal Regression and Classification Algorithms [11, 7]) es un framework de Matlab/Octave [3, 9] que implementa e integra una amplia variedad de métodos de clasificación ordinal. Fue desarrollado por el grupo de investigación AYRNA (Aprendizaje y Redes Neuronales Artificiales [2]), del Departamento de Informática y Análisis Numérico de la Universidad de Córdoba y participó en su elaboración uno de los codirectores de este trabajo fin de grado Pedro Antonio Gutiérrez Peña.

4.3. NSLVOrd

NSLVOrd es un algoritmo desarrollado por uno de los codirectores de este trabajo fin de grado, Juan Carlos Gámez Granados, en su Tesis Doctoral definida en septiembre de 2017 en la Universidad de Granada [5].

Este algoritmo es una extensión de NSLV que permite abordar problemas de clasificación ordinal. Para lo que respecta al problema de este proyecto, cabe destacar de NSLVOrd que admite ficheros WEKA [8] y se encuentra como un paquete Java [4].

4.4. JFML

JFML [12] es una biblioteca que permite la exportabilidad de sistemas de lógica difusa en formato XML siguiendo el estándar *IEEE Std 1855TM-2016*. Además, permite la exportación de estos sistemas en otros formatos como es el de PMML [6].

4.5. WEKA

WEKA (Waikato Environment for Knowledge Analysis [8]) es una librería Java [4] desarrollada en la Universidad de Waikato. Es software de código abierto emitido bajo la GNU General Public License.

WEKA es una colección de algoritmos de aprendizaje automático para tareas de minería de datos y contiene herramientas para la preparación de datos, clasificación, regresión, agrupación, extracción de reglas de asociación y visualización. El formato de los ficheros de entrenamiento y test que utiliza WEKA se utiliza en otros *suites* por lo que se puede tomar como un estándar y es por lo que se utilizará también en este trabajo.

4.6. Java

4.6. Java

Java [4] es un lenguaje de programación orientado y semicompilado comercializada por primera vez en 1995 por Sun Microsystems. Funciona con una máquina virtual llamada Java Virtual Machine (JVM) que permite la ejecución en diferentes sistemas operativos.



5 Restricciones

En este capítulo se expondrán aquellos factores que pueden ser limitativos y restrictivos a la hora realizar este proyecto. Para ello, se dividirán en los factores datos, aquellos que son provocados por la naturaleza del problema, y los factores estratégicos, aquellos que son elegidos para ayudar a la realización del producto.

5.1. Factores Dato

En esta sección se expondrán los factores dato, es decir, aquellas restricciones que vienen impuestas por la naturaleza del problema y no se pueden cambiar. Debido a que es un Trabajo de Fin de Grado, también se añadirán las restricciones impuestas por los directores del proyecto.

- Se usará el lenguaje de programación de Matlab/Octave [3, 9], ya que ORCA [11, 7] está escrito en este lenguaje.
- No se podrán crear ficheros, ni aunque sean temporales, durante la ejecución de un experimento.
- Se deberá cumplir con todos los objetivos que se han establecido en el capítulo 3.
- Se usará Java 8 [4] ya que tiene librerías que usa JFML [12].

5.2. Factores Estratégicos

En esta sección se expondrán los factores estratégicos, es decir, aquellas opciones elegidas de forma personal por el proyectista.

- **Exportar las reglas:** exportando las reglas se pretende que estas puedan ser usadas por el usuario en otras aplicaciones. Por lo tanto, se deben exportar en un formato que sea bastante usado.

Para esto, se ha decidido usar la librería JFML [12], ya que permite exportar reglas difusas y también exportarlo en formato PMML [6].

- **Representación de reglas:** para representar las reglas se va a crear una interfaz de usuario sencilla que consistirá en una ventana donde se muestren las reglas. Para crearla se puede hacer uso de varios lenguajes de programación como Matlab [3], C [10] o Java [4]. Para este proyecto se ha decidido usar Java ya que es un lenguaje en el que ya se ha enseñado en la carrera a como hacer una interfaz gráfica con este.

- **Formato de los ficheros de datos que permiten categóricos:** ORCA [11, 7] usa ficheros de texto plano que solo permiten datos numéricos, siendo una solución intentar adaptar la carga de datos para que permita categóricos, sin embargo, el hecho de añadirlos implica cambiar la forma en la que se leen estos ficheros en ORCA y especificar los valores que puede tomar un categórico en el fichero. Esto obliga a tener que crear un nuevo tipo de fichero datos, por lo que, para no tener que cambiar por completo el formato de los ficheros que ya se usan, se ha decidido que se puedan añadir más formatos con sus correspondientes formas de leerlos.

Como se van a poder añadir más formatos de texto, en este proyecto se ha decidido que el formato que se va a incluir que permita categóricos serán ficheros de formato WEKA [8], ya que es uno bastante conocido y usado, a demás que se usa este tipo de fichero en NSLVOrd [5].

5.2. Factores Estratégicos

- **Versiones:** actualmente, la versión más reciente de Matlab es la de 2020 y la de Octave [9] es la 5.2.0, sin embargo, para este proyecto se usarán la versión de 2014 de Matlab y la 4.2.2 de Octave para evitar errores por incompatibilidad de funciones con versiones posteriores a las más recientes.

En el caso de Matlab, se ha cogido dicha versión ya que al ser un *software* de pago, no se puede imponer que tengan versiones más actualizadas para el uso de ORCA; y en el caso de Octave, porque es la versión con la que trabaja Linux.



6 Recursos

En este capítulo se expondrán los distintos recursos humanos, de hardware y de software necesarios para realizar este proyecto.

6.1. Recursos Humanos

- **Autor:** Federico León García-Arévalo Calles.
Alumno de 4º del Grado de Ingeniería Informática.
Especialidad Computación y Computadores.
- **Director:** Pedro Antonio Gutiérrez Peña.
Profesor Titular de la Universidad de Córdoba del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo AYRNA.
- **Director:** Juan Carlos Gámez Granados
P.E.S. Comisión de servicios de la Universidad de Córdoba del Dpto. de Ingeniería Electrónica y de Computadores

6.2. Recursos Hardware

- **Procesador:** Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz.
- **RAM:** 16GB.
- **Tarjeta Gráfica:** Nvidia GeForce GTX 1060 con 6GB de VRAM.
- **Disco duro:** 1TB de HDD y 500GB de SSD

6.3. Recursos Software

- **Sistema operativo:** Windows 10 (64bits).
- **Documentación:** TeXstudio.
- **Entorno de desarrollo:** Matlab 2014a, Octave 4.2.2 y Java 8.

Parte II

Análisis del Sistema



7 Casos de Uso

En este capítulo se expondrán los casos de uso que existirán para este sistema, de tal forma que nos permita obtener los posibles requisitos del sistema. Estos casos de uso permitirán entender de forma simplificada el funcionamiento del sistema y las posibles respuestas frente a diferentes casos.

Debido a que se trabaja sobre ORCA, en estos casos de uso se contemplará solo las partes en las que afecta los problemas planteados en este trabajo.

Para poder hacer los casos de uso, primero hay que indicar cuáles son los actores que interactuarán con el sistema. En este caso, solo existe un actor que es el usuario que usará ORCA, al cual llamaremos *Usuario* en los casos de uso.

7.1. Ejecutar un Experimento

Aunque se haya dicho que en estos casos de uso solo se contemplará lo añadido en este trabajo, se hará una excepción en este, ya que es necesario para saber como se ejecuta, de forma superficial, un experimento en ORCA.

CU-01	Ejecutar un experimento.
Descripción	Ejecución de un experimento en ORCA.
Actores	Usuario.
Precondiciones	<ol style="list-style-type: none"> 1. El usuario debe estar en la ruta donde se encuentra el fichero de la clase <i>Utilities</i>. 2. El usuario debe tener el fichero que contiene los parámetros del experimento.
Flujo principal	<ol style="list-style-type: none"> 1. Configura el experimento: <ol style="list-style-type: none"> 1.1. Comprueba el fichero con los datos del experimento. 1.2. Crea la carpeta donde se guarda los resultados del experimento. 2. Ejecuta el experimento: <ol style="list-style-type: none"> 2.1. Lee los parámetros de entrenamiento. 2.2. Lee los datos de entrenamiento y test (ver CU-06 en la tabla 7.6). 2.3. SI existe combinaciones de parámetros, se hace <i>cross-validate</i>. 2.4. Se hace la fase de entrenamiento (ver CU-04 en la tabla 7.4). 2.5. Se hace la fase de test (ver CU-05 en la tabla 7.5). 3. Guarda los resultados del experimento.
Postcondiciones	<ol style="list-style-type: none"> 1. Se ha creado una carpeta con los datos del experimento en 'orca/src/Experiments'.

Tabla 7.1: CU-01: Ejecutar un experimento

7.2. Exportar las Reglas

CU-02	Exportar las reglas.
Descripción	El usuario puede exportar las reglas a ficheros JFML y PMML.
Autores	Usuario
Precondiciones	1. El modelo entrenado de NSLVOrd debe estar cargado. 2. Pasar como argumento un directorio donde guardar los ficheros con las reglas.
Flujo	1. Obtener la base de conocimiento y reglas del modelo. 2. Añadir la base de conocimiento a la clase <i>ExportRules</i> . 3. Añadir la base de reglas a la clase <i>ExportRules</i> . 4. Exportar las reglas usando el algoritmo JFML.
Postcondiciones	1. Se crearán dos <i>xml</i> en el directorio pasado como argumento.

Tabla 7.2: CU-02: Exportar las reglas

7.3. Visualizar las Reglas

CU-03	Visualizar las reglas.
Descripción	El usuario puede visualizar las reglas de un modelo entrenado.
Autores	Usuario
Precondiciones	1. El modelo entrenado de NSLVOrd debe estar cargado.
Flujo	1. Obtener la base de conocimiento y las reglas del modelo. 2. Añadir la base de conocimiento a la clase <i>VisualRules</i> . 3. Añadir las reglas a la clase <i>VisualRules</i> . 4. Visualizar las reglas en una ventana creada en Java. 5. SI el usuario quiere cambiar el tamaño de las reglas: 5.1. Escribir el número de antecedentes máximo por línea en la caja de texto. 5.2. Pulsar el botón OK. 6. Cerrar la ventana.
Postcondiciones	-

Tabla 7.3: CU-03: Visualizar las reglas

7.4. Entrenamiento y Test

Como se ha dicho anteriormente, estos casos de uso irán dirigidos a los problemas planteados en este trabajo. Por ello, los siguientes casos de uso de entrenamiento y test (tablas 7.4 y 7.5 respectivamente) solo contemplarán el algoritmo NSLVOrd.

CU-04	Realizar entrenamiento.
Descripción	Proceso que sigue la función de entrenamiento <i>privfit</i> de la clase NSLVOrd.
Autores	Usuario
Precondiciones	<ol style="list-style-type: none"> 1. Pasar como argumento los datos de entrenamiento. 2. Pasar como argumento los parámetros de entrenamiento.
Flujo	<ol style="list-style-type: none"> 1. Conversión de los parámetros de entrada a objeto Java. 2. Llamar a la función en Java del entrenamiento de NSLVOrd. 3. Recoger las salidas que produce el entrenamiento. 4. Obtener la base de conocimiento del entrenamiento. 5. Obtener la base de reglas del entrenamiento. 6. Guardar en el modelo: <ol style="list-style-type: none"> 6.1. La base de conocimiento. 6.2. La base de reglas. 6.3. Los parámetros de entrenamiento. 7. SI el parámetro para visualizar las reglas está activo: <ol style="list-style-type: none"> 7.1 Ver el flujo del CU-3 en la Tabla 7.3. 8. FINSI
Postcondiciones	<ol style="list-style-type: none"> 1. Se guarda la información del modelo en <i>obj.modelo</i>. 2. Se devuelve las salidas que produce el modelo ya entrenado para los datos de entrada.

Tabla 7.4: CU-04: Realizar entrenamiento

7.5. Leer Datos

CU-05	Realizar test.
Descripción	Proceso que sigue la función de test <i>privpredict</i> de la clase NSLVOrd
Autores	Usuario
Precondiciones	1. Tener cargado el modelo en la clase NSLVOrd. 2. Pasar como argumento las entradas de los datos de prueba.
Flujo	1. Conversión de los parámetros de entrada a objeto Java. 2. Obtener del modelo: 2.1. La base de conocimiento. 2.2. La base de reglas. 3. Cargar la base de conocimiento y reglas en el algoritmo NSLVOrd. 4. Llamar a la función en Java del test de NSLVOrd. 5. Recoger las salidas que produce el test.
Postcondiciones	1. Se devuelve las salidas que produce el test para los datos de entrada.

Tabla 7.5: CU-05: Realizar test

7.5. Leer Datos

CU-06	Leer datos.
Descripción	Lectura de los datos de los fichero con formato Weka
Autores	-
Precondiciones	1. Tener el fichero a leer en una carpeta llamada weka.
Flujo	1. Leer la cabecera: 1.1. Guardar la información de los atributos. 2. Leer los datos: 2.1. SI el algoritmo permite categóricos, guardar los datos como char. 2.1. SI NO, convertir los categóricos a <i>TO-ONE-HOT</i> y guardar los datos como numérico.
Postcondiciones	1. Se devuelve una estructura con los datos de entrada, datos de salida e información sobre los atributos.

Tabla 7.6: CU-06: Leer datos



8 Especificación de Requisitos

En este capítulo se identificarán y clasificarán los requisitos que debe seguir el sistema. Los requisitos se expondrán sin entrar en los detalles de su implementación y se etiquetarán según la clasificación.

8.1. Requisitos de Usuario

A continuación se mostrarán los requisitos relacionados con la interacción del usuario y el sistema:

- RU-01: el usuario solo deberá interactuar directamente con ORCA.
- RU-02: el usuario deberá tener la opción de visualizar las reglas generadas.
- RU-03: en los ficheros que proporcionan la información para un experimento, el usuario tendrá la opción de indicar si los ficheros de datos son de tipo *matlab* o *weka*. Los de tipo *matlab* son ficheros de texto plano que no permiten datos categóricos mientras que los de tipo *weka* son ficheros en formato WEKA que sí tienen datos categóricos.

8.2. Requisitos del Sistema

En esta sección, se expondrán los requisitos que debe cumplir el sistema, diferenciándolos entre requisitos funcionales y no funcionales.

8.2.1. Requisitos no Funcionales

- RNF-01: los cambios e inclusiones realizados en ORCA deben ser compatibles con Matlab 2014a y versiones posteriores.
- RNF-02: los cambios e inclusiones realizados en ORCA deben ser compatibles con Octave 4.2.2.
- RNF-03: se reutilizará el código Java del algoritmo NSLVOrd.
- RNF-04: se usará las funciones de JFML para la exportación de las reglas.
- RNF-05: se usará Java para codificar la visualización de las reglas.
- RNF-06: las reglas deben visualizarse de forma que puedan ser fáciles de entender por cualquier tipo de usuario.

8.2.2. Requisitos Funcionales

- RF-01: ORCA leerá el tipo de fichero del que leerá los datos del fichero de configuración.
- RF-02: de ficheros tipo Weka se podrán leer datos de tipo numérico y categóricos.
- RF-03: los algoritmos indicarán si son compatibles con datos categóricos, por defecto no lo serán.
- RF-04: si un algoritmo no es compatible con datos categóricos, se crearán variables *dummy* al estilo *one-hot*. Una variable *dummy* es una variable numérica que puede tomar el valor 0 o 1 y que representa a un valor categórico; y el estilo *one-hot* es que cuando una variable

8.2. Requisitos del Sistema

categórica tiene varios valores, se crea una variable *dummy* para cada valor y se pone a 1 la variable que representa el categórico correspondiente mientras los demás se ponen a 0. En la Tabla 8.1 se pondrá un ejemplo donde la primera columna representa los valores que puede tomar una variable categórica y el resto a las variables *dummy* creadas a partir de la categórica.

Color	dummy1	dummy2	dummy3
rojo	1	0	0
verde	0	1	0
azul	0	0	1

Tabla 8.1: Ejemplo de convertir una variable categórica en *one-hot*

- RF-05: se añadirá la posibilidad de que se pueda configurar la exportación de reglas en cualquier algoritmo.
- RF-06: se añadirá la posibilidad de que se pueda configurar la visualización de las reglas en cualquier algoritmo.
- RF-07: el visualizador de reglas permitirá cambiar el número de antecedentes de una regla que puede haber por fila.

Parte III

Modelado del Sistema



9 Arquitectura del Sistema

En este capítulo se expondrá la estructura que tendrá el sistema definiendo los módulos que la componen y cómo interactúan entre ellos.

Primero, se aclarará una distinción entre estos según su origen:

- **Originales:** son aquellos que no han sido modificadas durante el desarrollo de este trabajo.
- **Modificados:** en mayor o menor medida, se ha editado el código para adaptarlo a las necesidades del problema.
- **Nuevos:** aquellos que han sido creados desde cero para este trabajo.

9.1. Módulos

- **ORCA:** modificado para adaptarlo a la nueva forma de lectura de datos. Este es el módulo principal del sistema que se encarga configurar el experimento, calcular las métricas de los resultados y exportar toda la información. Está codificado en lenguaje Matlab.
- **Read File:** módulo nuevo creado de tal forma que, al igual que los algoritmos en ORCA, se puedan añadir más métodos de lectura. Este módulo se encarga de leer los ficheros de entrenamiento y test; devolviendo los datos separando las entradas y salidas. Está codificado en lenguaje Matlab.
- **Algorithms:** módulo original, aunque se ha añadido una variable para indicar si permite datos categóricos y se le ha añadido la subclase NSLVOrd. Este módulo se encarga de contener diferentes algoritmos y de la ejecución de estos. Está codificado en lenguaje Matlab.
- **NSLVOrd:** modificado para adaptar la forma en la que recibe y devuelve los datos. Este módulo es el que se encarga de ejecutar el algoritmo NSLVOrd. Está codificado en Java.
- **Rule View:** módulo nuevo que se encarga de mostrar las reglas generadas por el algoritmo en una ventana de tal forma que cualquier tipo de usuario pueda entenderlas. Está codificado en Java.
- **JFML:** módulo original. Este módulo es el encargado de exportar las reglas generadas por el algoritmo a ficheros XML en formato JFML y PMML en un directorio especificado por el usuario. Está codificado en Java.

9.2. Interacciones del Sistema

En la Figura 9.1 se muestra gráficamente y de forma sencilla los módulos mencionados en la sección anterior y como están conectados entre ellos.

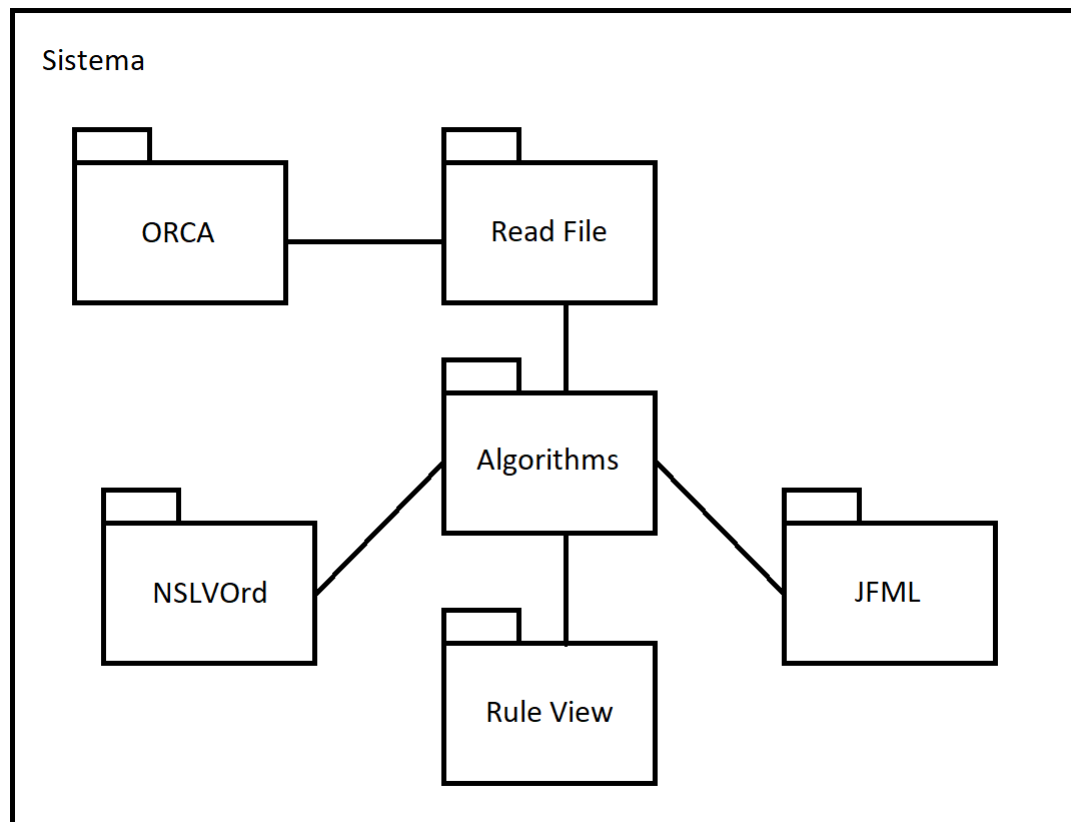


Figura 9.1: Estructura modular del sistema

9. Arquitectura del Sistema

Por último, se va a representar las relaciones de control entre los distintos módulos. El diagrama de la Figura 9.2 muestra como son las interacciones entre las relaciones de los módulos.

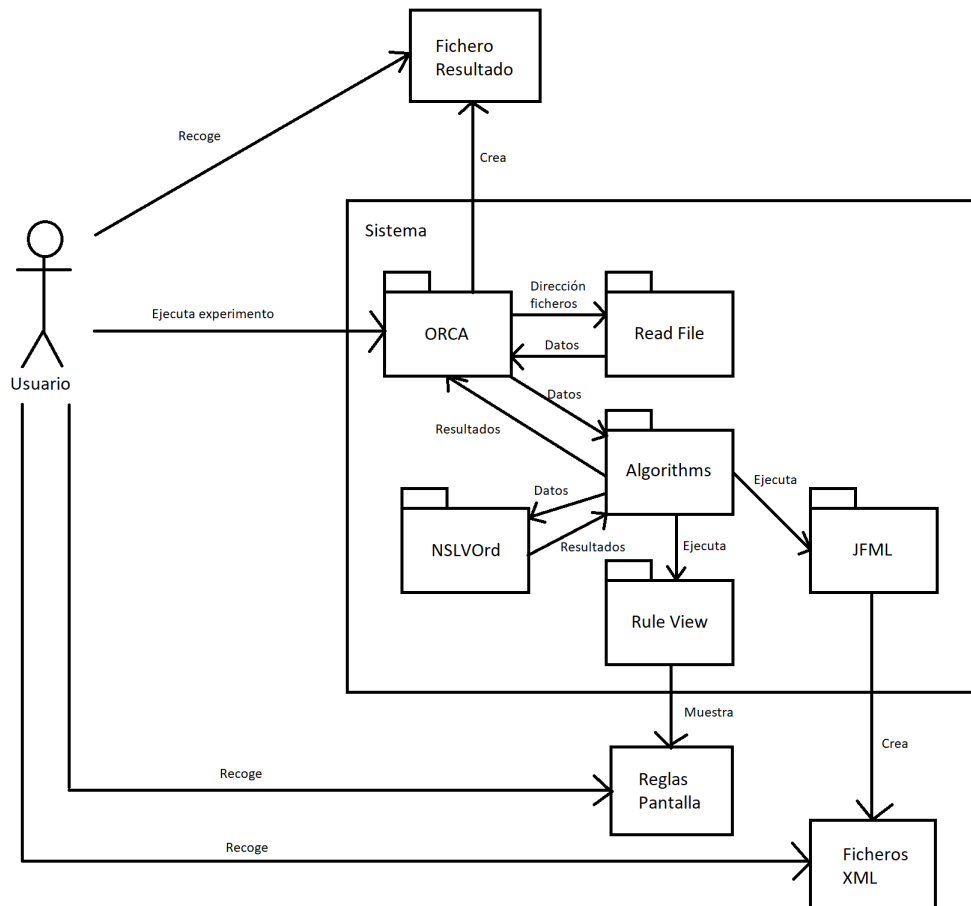


Figura 9.2: Estructura modular del sistema



10 Diagramas de Secuencia

En este capítulo, se pretende mostrar, de forma más específica que en el apartado anterior, las interacciones que se producen entre el usuario y el sistema, además de las interacciones entre los módulos de este.

En la Figura 10.1 se muestra las interacciones del sistema cuando se ejecuta un experimento. Y por otro lado, en las Figuras 10.2 y 10.3 se muestran las interacciones cuando se exportan y se visualizan las reglas respectivamente, pudiendo ver que son muy similares.

10. Diagramas de Secuencia

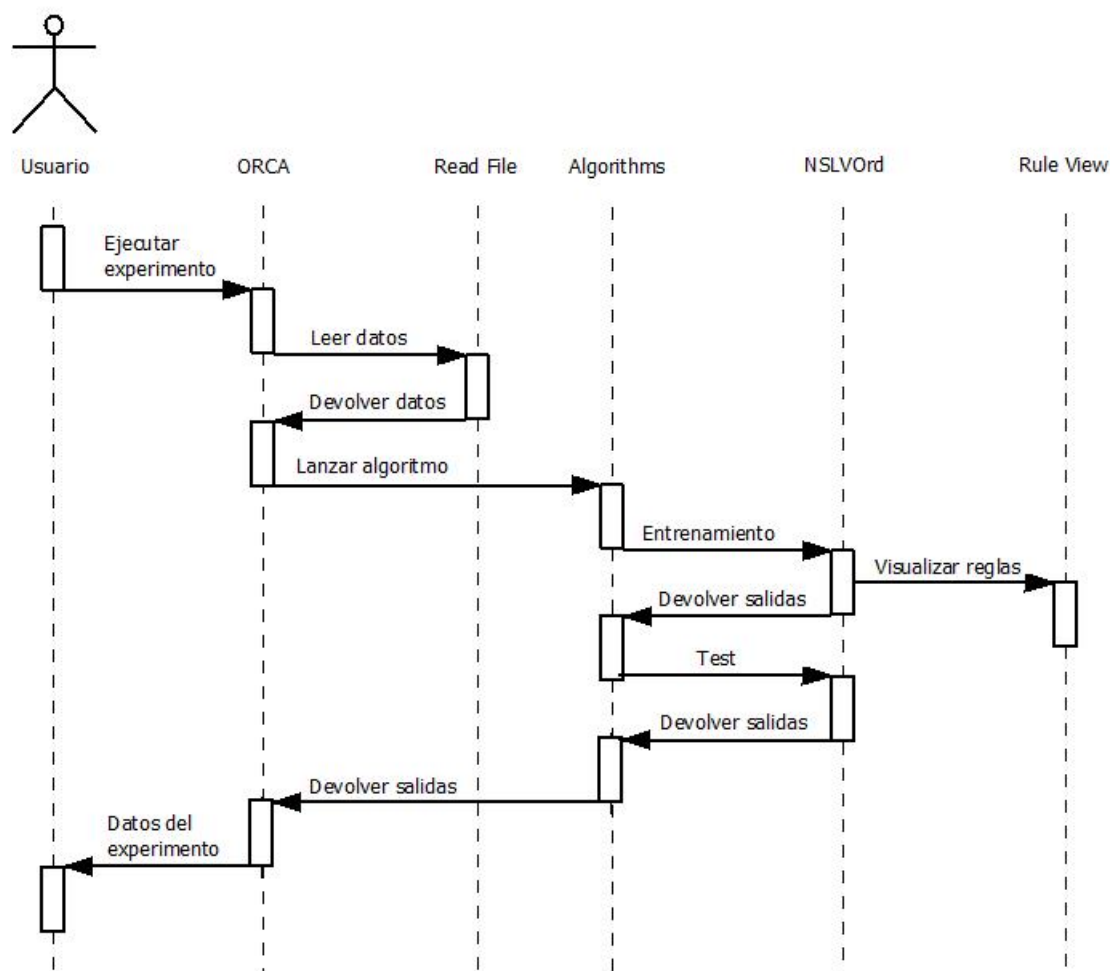


Figura 10.1: Diagrama de secuencia para la ejecución de un experimento.

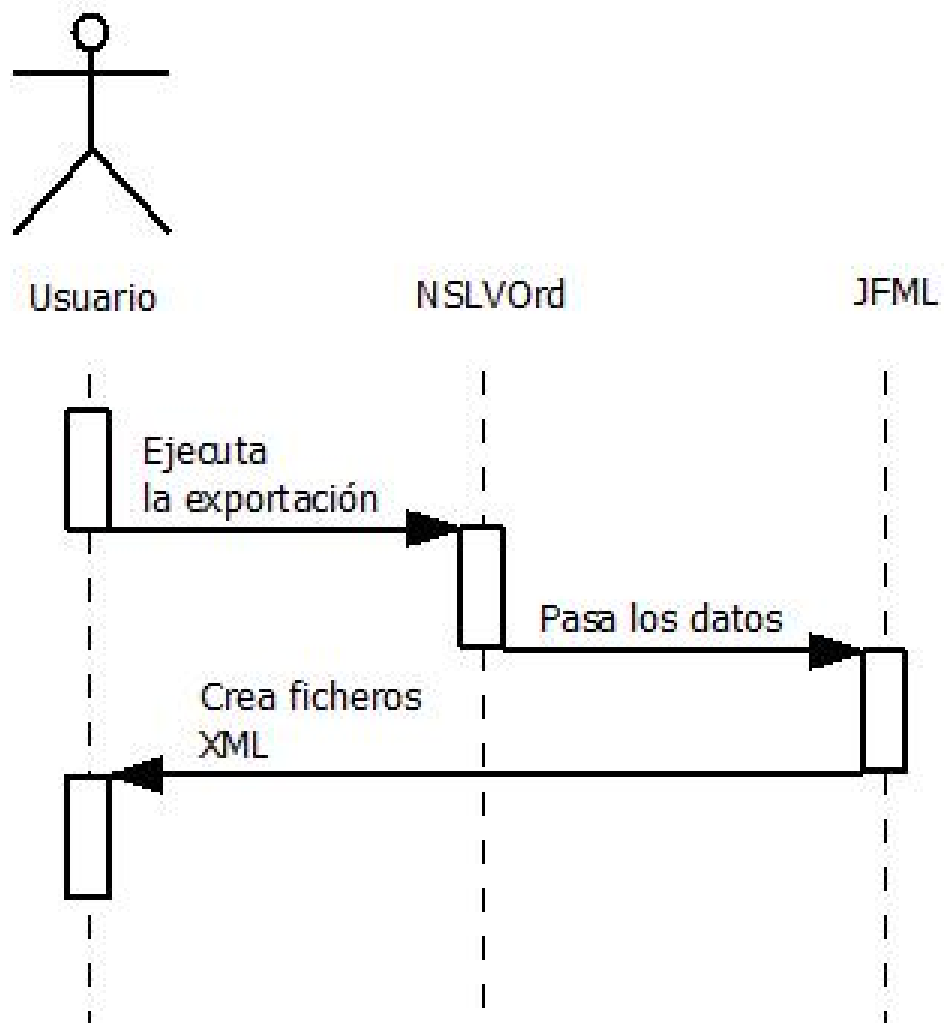


Figura 10.2: Diagrama de secuencia de la exportación de las reglas.

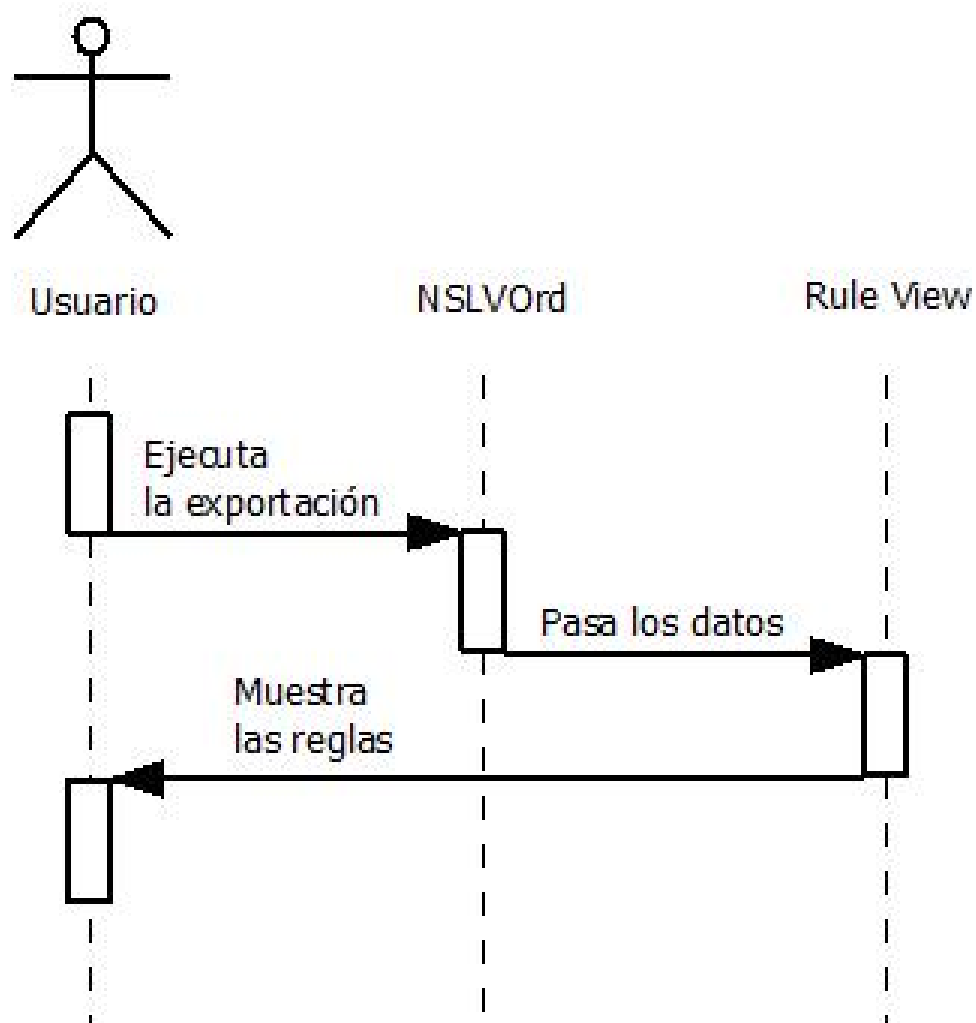


Figura 10.3: Diagrama de secuencia de la visualización de las reglas.

Parte IV

Diseño del Sistema



11 Diagramas de Clases

En este capítulo se expondrán las clases más relevantes que intervienen en este trabajo. Es decir, las nuevas clases que se han añadido y las modificaciones de las ya existentes. Para una mejor organización de las clases, estas se van a dividir según al módulo que pertenezca de los mostrados en el capítulo 9.1.

Para describir los métodos y datos de cada clase seguiremos una notación similar a la que se muestra en la Tabla 11.1. Tanto para las clases, los datos y los métodos, se acompañarán con un ‘*’ si solo han sido modificadas y ‘**’ si han sido creadas (en el caso de una clase creada, no se añadirá la anotación a los datos o métodos).

Clase: nombre
Descripción de la clase
Hereda: clase de la que hereda
Datos
variable1, variable2, ...
Métodos
método1, método2, ...

Tabla 11.1: Ejemplo general para la especificación de clases

11.1. ORCA

Este módulo al no ser originario de este trabajo, solo se presentarán sus clases más relevantes. Es decir, la clase principal y aquellas que se relacionan directamente con otros módulos.

Clase Utilities

Esta clase es la principal del módulo y del sistema en general. Por eso, esta clase será descrita como se puede ver en la Tabla 11.2.

Clase: Utilities*
Clase que contiene métodos para configurar y ejecutar los experimentos.
Hereda: handle
Datos
Métodos
runExperiments, octaveParallelAuxFunction, results*, configureExperiment*, runExperimentFold, processDirectory*, checkDatasets*, preparePool, closePool, parseParArgs

Tabla 11.2: Clase Utilities

Clase Dataset

Esta clase, representada en la Tabla 11.3, se conecta directamente con el módulo *Read File*.

Clase: Dataset*
Clase para especificar el nombre de los conjuntos de datos y realizar el preprocesamiento de datos.
Hereda: handle
Datos
directory, train, test, standarize, dataname, nOfFolds
Métodos
dataSet, set.directory, preProcessData*, standarizeData, standarizeFunction, scaleData, deleteNonNumericValues, deleteConstantAtributes,

Tabla 11.3: Clase Dataset

11.1. ORCA

Clase Experiment

En la Tabla 11.4 se describe a la clase *Experiment*, que se conecta con el módulo *Algorithms*.

Clase: Experiment*
Clase que crea y ejecuta el experimento.
Hereda: handle
Datos
data, method, cvCriteria, croosvalide, resultsDir, report_sum, seed, parameters, logsDir
Métodos
launch, run, process, saveResults*, crossValideParams, mapsToCell

Tabla 11.4: Clase Experiment

11.2. Read File

Clase TFGFileReadClass

Esta clase, que se representa en la Tabla 11.5, es la que sirve de intermediario entre el módulo ORCA (sección 11.1) y las clases de este módulo que se encargan de leer los datos.

Clase: TFGFileReadClass**
Clase que contiene métodos para realizar la lectura de los datos.
Hereda: -
Datos
path
Métodos
Valid_archive, ReadFile, TFGFileName, SearchInvalidValue

Tabla 11.5: Clase TFGFileReadClass

Clase ReadFileCommon

Al igual que la clase *Algorithm* (Tabla 11.9), la clase *ReadFileCommon* (Tabla 11.6) se ha creado como una clase abstracta para que se puedan ir añadiendo más subclases y poder aumentar el número de métodos con los que poder leer los datos.

Clase: ReadFileCommon**
Clase abstracta que define los ajustes para los métodos de lectura de datos.
Hereda: handle
Datos
categ, info, categ_att
Métodos
FormatFile, Format, ReadFileFunction, ReadFile, deleteNonNumericValues

Tabla 11.6: Clase ReadFileCommon

11.2. Read File

Clase matlab

La clase *matlab* (Tabla 11.7) es la que se encarga de leer ficheros de datos soportados por Matlab y Octave. Debido a como está montado ORCA, solo permite que estos ficheros contengan datos numéricos.

Clase: matlab**
Clase que contiene los métodos para leer archivos de tipo matlab.
Hereda: ReadFileCommon
Datos
Métodos
Format, ReadFile

Tabla 11.7: Clase matlab

Clase weka

La clase *weka* (Tabla 11.8) es la encargada de leer los ficheros de formato WEKA. Con esta clase, se añade la posibilidad de añadir datos categóricos a ORCA.

Clase: weka**
Clase que contiene los métodos para leer archivos de tipo weka.
Hereda: ReadFileCommon
Datos
attrs
Métodos
Format, ReadFile, ReadWekaFile, ReadHeader, NewAttribute, ReadDatan, ToOneHot, ToNumeric, Categorical_to_Numeric

Tabla 11.8: Clase weka

11.3. Algorithms

Este módulo no es originario de este trabajo, por lo que solo se mostrará la clase más relevante. Sin embargo, a este módulo se le ha añadido una clase nueva para poder integrar el algoritmo NSLVOrd.

Clase Algorithm

La clase *Algorithm* (Tabla 11.9) es la principal del módulo. Es una clase abstracta que se usa como base para crear las clases de los algoritmos que posee ORCA.

Clase: Algorithm*
Clase abstracta que define los ajustes para los algoritmos.
Hereda: handle
Datos
model, categ**, export**, visual**
Métodos
runAlgorithm, fit, predict*, privfit, privpredict, parseArgs, setParam, getModel, setModel, getParameterNames, export_rules**, visual_rules**

Tabla 11.9: Clase Algorithm

Clase NSLVOrd

En la Tabla 11.10 se representa la clase *NSLVOrd* que se ha añadido para cumplir el objetivo de añadir NSLVOrd a ORCA (sección 3.1).

Clase: NSLVOrd**
Clase que contiene los métodos que ejecutan el algoritmo NSLVOrd.
Hereda: Algorithms
Datos
description, parameters
Métodos
NSLVOrd, initParameters, getHeader, getDatas, toJavaString, ConvertTargetsToCategoric, ConvertCategoricToTarget, toChar, toCell, privfit, privpredict, visual_rules, export_rules, subrules, getant

Tabla 11.10: Clase NSLVOrd

11.4. NSLVOOrd

Este módulo ha sido incorporado al sistema en este trabajo, sin embargo, se basa en un programa ya creado que ha sido modificado para adaptar la forma de recibir y devolver las entradas. Por eso, solo se expone la clase más relevante.

Clase NSLVOOrdJava

La clase NSLVOOrd, reflejada en la Tabla 11.11, es la clase principal de este módulo y la que ha sido adaptada, tanto modificando métodos como añadiendo otros nuevos.

Clase: NSLVOOrdJava*
Clase principal de la ejecución del algoritmo NSLVOOrd.
Hereda: -
Datos
E_par, E_par_test, R, alpha, costMatrix, tSet, fileResultDebug, iSet fuzzyProblem, homogeneousLabel, poblacionParam, seed, shift, time iter, numDesplazamientos, numLabelsInputs, randomNum
Métodos
Train**, GetFuzzyProblem**, get_knowledge_base**, LoadModel**, ReadSet**, Load_KnowledgeBase**, Load_RuleBase**, Test**, initParameters*, executePredict*, get_rule_base**, Targets**, executeNSLVOOrd*, writeSyntax, executeNSLVOOrdPredict*, get_rules**, checkSyntax, getParametersKeel, getCostMatrix, executeLearning*,

Tabla 11.11: Clase NSLVOOrdJava

Clase RuleSystem

En la Tabla 11.12 se refleja la clase RuleSystem que se encarga de configurar la base de conocimiento y reglas para ser exportadas a ORCA.

Clase: RuleSystem**
Clase que configura la base de conocimiento y reglas para exportarlas a ORCA.
Hereda: -
Datos
_f, _fuzzyProblem, _R
Métodos
Export_KnowledgeBase, Export_RuleBase, Export_Antecedents, Export_Rules

Tabla 11.12: Clase RuleSystem

11.5. Rule View

Este módulo está dividido en una parte codificada dentro de ORCA (sección 11.5.1) y otra codificada en Java (sección 11.5.2).

11.5.1. ORCA

Clase RulesVisual

Esta clase (Tabla 11.13) sirve como intermediaria para que se pueda configurar las reglas para ser mostradas en los algoritmos.

Clase: RulesVisual**
Clase que configura la visualización de las reglas.
Hereda: handle
Datos
rules
Métodos
visual_rules, detect_number, new_rule, add_antecedent, new_consequent

Tabla 11.13: Clase RulesVisual

11.5. Rule View

11.5.2. Java

Clase VisualRules

La clase *VisualRules* (Tabla 11.14) es la encargada de crear la ventana en la que se representa las reglas y permite cambiar el número de antecedentes por línea de una regla.

Clase: VisualRules**
Clase que crea una ventana donde se representan las reglas.
Hereda: javax.swing.JFrame
Datos
_OK, _actual_num_row, _cont, _info, _lista, jLabel1, jPanel1, rules
Métodos
CreateRules, SeeRules, _OKActionPerformed, _actual_num_rowKeyTyped, change_num_rules_in_row, getAntecedent, initComponents, new_antecedent, new_consequent, new_rule

Tabla 11.14: Clase VisualRules

Clase Rule

La clase *Rule* (Tabla 11.15) es usada por la clase *VisualRules* para mostrar una regla.

Clase: Rule**
Clase que representa una regla individual.
Hereda: javax.swing.JPanel
Datos
_IF, _THEN, _consequent, _ifpanel, _num, _num_row, _num_rules, _weight, _thenpanel
Métodos
add_antecedent, add_categoric_antecedent, add_fuzzy_antecedent, weight, initComponents, number, regroup_components, update_panel, consequent

Tabla 11.15: Clase Rule

Clase ConditionCategoric

Esta clase (Tabla 11.16) es usada por la clase *Rule* cuando uno de los antecedentes es de tipo categórico.

Clase: ConditionCategoric**
Clase que representa un antecedente individual de un dato categórico.
Hereda: javax.swing.JPanel
Datos
_is, _label, _num_labels, _parenthesis, _variable
Métodos
addLabel, initComponents, setVariable

Tabla 11.16: Clase ConditionCategoric

Clase ConditionFuzzyLogic

Esta clase (Tabla 11.17) es usada por la clase *Rule* cuando uno de los antecedentes se le ha aplicado lógica difusa.

Clase: ConditionFuzzyLogic**
Clase que representa un antecedente individual de un dato al que se le ha aplicado lógica difusa.
Hereda: javax.swing.JPanel
Datos
_is, _label, _num_series, _series, _variable
Métodos
add, createGraph, initComponents, new_series, setVariable

Tabla 11.17: Clase ConditionFuzzyLogic

11.6. JFML

11.6. JFML

Este módulo está dividido en una parte codificada dentro de ORCA (sección 11.6.1) y otra codificada en Java (sección 11.6.2).

11.6.1. ORCA

Clase RulesExport

Esta clase (Tabla 11.18) sirve como intermediaria para que se pueda configurar las reglas para ser exportadas en los algoritmos.

Clase: RulesExport**
Clase que configura la exportacion de las reglas.
Hereda: handle
Datos
rules, knowledge_base
Métodos
export_rules, detect_number, new_variables, add_terms, new_rule, add_antecedent, new_consequent

Tabla 11.18: Clase RulesExport

11.6.2. Java

Esta sección del módulo consiste en el programa JFML (sección 4.4) sin ningún tipo de modificación, pero aún así, se destacarán las clases de este que han sido más relevantes para este trabajo.

Clase *FuzzyInferenceSystem*

La clase *FuzzyInferenceSystem*, representada en la Tabla 11.20, es la que representa el sistema de reglas en su totalidad, incluyendo la base de conocimiento y de reglas. También se ha representado en la Tabla 11.19 la clase *FuzzySystemType* ya que es de la que hereda la clase *FuzzyInferenceSystem* y donde se encuentran los métodos que utiliza esta.

Clase: <i>FuzzySystemType</i>
Clase abstracta que representa al sistema que contiene las reglas.
Hereda: -
Datos
knowledgeBase, name, networkAddress, ruleBase
Métodos
addRuleBase, evaluate, evaluateAny, evaluateMamdani, evaluateRules, evaluateTsk, evaluateTsukamoto, getAllRuleBase, reset, getInferenceResults, getJAXBElement, getKnowledgeBase, getName, getNetworkAddress, getRuleBase, getVariable, getVariables, toString, setKnowledgeBase, setName, setNetworkAddress, setVariableValue

Tabla 11.19: Clase *FuzzySystemType*

Clase: <i>FuzzyInferenceSystem</i>
Clase que representa al sistema que contiene las reglas.
Hereda: <i>FuzzySystemType</i>
Datos
Métodos

Tabla 11.20: Clase *FuzzyInferenceSystem*

11.6. JFML

Clase KnowledgeBaseType

La clase *KnowledgeBaseType* (Tabla 11.21) representa la base de conocimiento del sistema, es decir, almacena la información de las variables que intervienen en el sistema. Para crear la base de conocimiento, intervienen otras clases que son *FuzzyVariableType* y *FuzzyTermType*, pero su intervención es escasa y no se ha visto necesaria su representación.

Clase: KnowledgeBaseType
Clase que representa la base de conocimiento del sistema.
Hereda: -
Datos
networkAddress, variable
Métodos
addVariable, getKnowledgeBaseVariables, getNetworkAddress, getVariable, getVariables, setNetworkAddress, toString

Tabla 11.21: Clase KnowledgeBaseType

Clase MamdaniRuleBaseType

La clase *MamdaniRuleBaseType* (Tabla 11.23) representa la base de reglas del sistema, es decir, almacena la información de las reglas que intervienen en el sistema. También se ha representado en la Tabla 11.22 la clase *RuleBaseType* ya que es de la que hereda la clase *MamdaniRuleBaseType* y donde se encuentran los métodos que utiliza esta. Para crear la base de reglas, intervienen otras clases que son *AntecedentType*, *ClauseType* y *ConsequentType*, pero su intervención es escasa y no se ha visto necesaria su representación.

Clase: RuleBaseType
Clase que representa la base de reglas del sistema.
Hereda: FuzzySystemRuleBase
Datos
activationMethod, andMethod, name, networkAddress, orMethod, rules
Métodos
activationMamdani, activationTsukamoto, addRule, evaluate, evaluateAntecedents, getActivatedRules, getActivationMethod, getAndMethod, getName, getNetworkAddress, getOrMethod, getRules, implyConsequent, reset, setActivationMethod, setAndMethod, setName, setNetworkAddress, setOrMethod, toString

Tabla 11.22: Clase RuleBaseType

Clase: MamdaniRuleBaseType
Clase que representa la base de reglas del sistema.
Hereda: RuleBaseType
Datos
Métodos

Tabla 11.23: Clase MamdaniRuleBaseType

11.6. JFML

Export

En la Tabla 11.24 y Tabla 11.25 se representan las clases JFML y ExportPMML respectivamente. Estas clases son las encargadas de exportar las reglas en ficheros xml.

Clase: JFML
Clase que contiene los métodos para exportar el sistema de reglas a un xml.
Hereda: -
Datos
Métodos
cheKingTerms, initializeMembershipFunctions, load, readFSTfromXML, removePrefixNS, writeFSTtoXML

Tabla 11.24: Clase JFML

Clase: ExportPMML
Clase que contiene los métodos para exportar el sistema de reglas a un xml con formato PMML.
Hereda: Export
Datos
Métodos
exportFuzzySystem, rankClauses

Tabla 11.25: Clase ExportPMML



12 Diseño de la Interfaz Gráfica

Para este trabajo se ha desarrollado una interfaz gráfica con la que poder visualizar las reglas generadas con alguno de los algoritmos de ORCA y cumpliendo el objetivo del trabajo correspondiente (sección 3.3). Y en este capítulo se describirá el diseño de esta interfaz.

En la Figura 12.1 se muestra un esquema de la interfaz que servirá para explicar las diferentes secciones de esta, además, en la Figura 12.2 se muestra un ejemplo con la interfaz en funcionamiento. A continuación se explican las diferentes secciones:

1. Barra de la ventana donde sale el nombre y los botones de minimizar, extender y cerrar la ventana.
2. Sección en la que puedes indicar el número de antecedentes de una regla por línea.
3. Botón para aplicar el cambio de la sección 2.
4. Información extra para aumentar o dejar por defecto las gráficas de los antecedentes en los que se haya aplicado lógica difusa.
5. Sección en la que se muestran todas las reglas.

12. Diseño de la Interfaz Gráfica

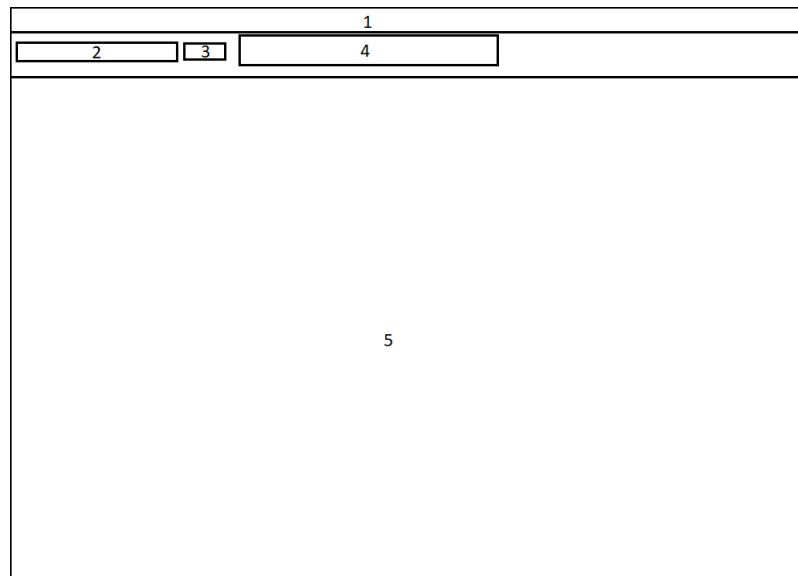


Figura 12.1: Esquema de la interfaz gráfica.

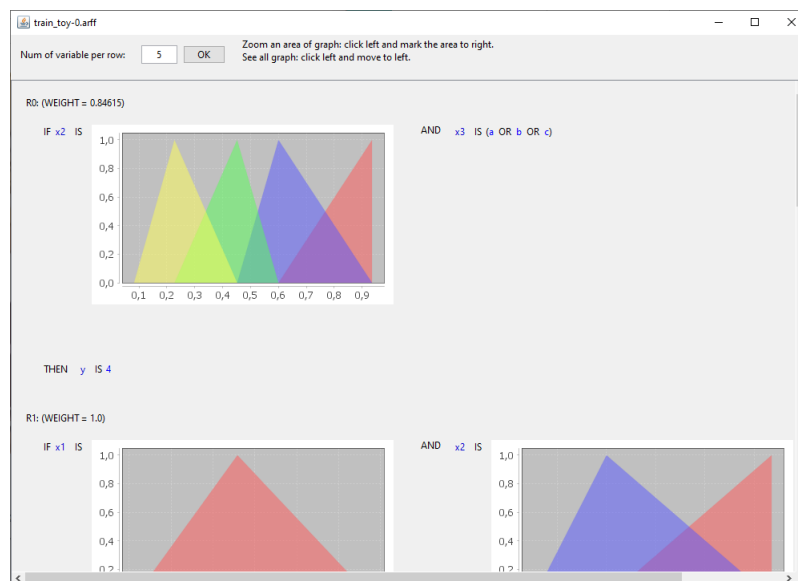


Figura 12.2: Ejemplo de la interfaz gráfica.

Parte V

Cierre

13 Pruebas

Con el fin de garantizar el correcto funcionamiento del sistema, se han ido haciendo una serie de pruebas durante el desarrollo de éste. Por ello, en este capítulo se pretende mostrar qué pruebas se han realizado para el sistema.

Como se ha descrito en el capítulo 9, el sistema se divide en varios módulos que interactúan entre sí, por lo que se harán pruebas a estos por separado para comprobar que funcionan correctamente de forma individual antes de probarlo como un sistema completo.

Para aclarar, entre las pruebas se encontrarán, explicadas de forma simple: pruebas de caja blanca o pruebas estructurales que se encarga de testear los posibles flujos de ejecución del programa; pruebas de caja negra que comprueba que a partir de unas entradas dé las salidas esperadas.

Las pruebas se especificarán como se puede ver en la Tabla 13.1.

Prueba Cod.XX	
Descripción	-
Exito	1. - ...
Fallo	1. - ...

Tabla 13.1: Ejemplo general para una prueba

13.1. Datos para las Pruebas

Datasets

En la Tabla 13.2 se encuentran los *datasets* que se usarán para las pruebas. Los *datasets* [1] y la tabla están sacados de la tesis doctoral [5] de Juan Carlos Gámez Granados, ya que se pretende hacer la comparación con los resultados de éste. Citando la descripción de la tabla: “[...] *Atrib (Num/Nom)* es el número de atributos totales, numéricos y nominales, *Ejemplos (part, entren, test)* es el número total de ejemplos, número de ejemplos en cada partición, número de ejemplos de entrenamiento de cada particion y número de ejemplos de test de cada particion, y *C* es el número de clases”.

Nombre	Atrib (Num/Nom)	Ejemplos (part/entren/test)	C
automobile	25 (15/1)	4590 (205/153/52)	6
balance-scale	4 (4/0)	14040 (625/468/157)	3
bondrate	10 (4/6)	1260 (57/42/15)	5
contact-lenses	5 (0/4)	540 (24/18/6)	3
ERA	4 (4/0)	22500 (1000/750/250)	9
ESL	4 (4/0)	10980 (488/366/122)	9
eucalyptus	19 (14/5)	16560 (736/552/184)	5
LEV	4 (4/0)	22500 (1000/750/250)	5
newthyroid	5 (5/0)	4830 (215/161/54)	3
pasture	22 (21/1)	810 (36/27/9)	3
squash-stored	24 (21/3)	1170 (52/39/13)	3
squash-unstored	24 (21/3)	1170 (52/39/13)	3
SWD	10 (10/0)	22500 (1000/750/250)	4
tae	5 (1/4)	3390 (151/113/38)	3
toy	2 (2/0)	6750 (300/225/75)	5
winequality-red	11 (11/0)	35970 (1599/1199/400)	6

Tabla 13.2: Datasets de prueba.

13.1. Datos para las Pruebas

Sistema de Reglas

A continuación, se mostrará un sistema de reglas sencillas con las que se harán las pruebas en las que se necesite. Se intentará que incluya todas las posibles formas de representar los datos:

1. Base de conocimiento:

- Variable A: entrada categórica
 - Valores: uno, dos, tres, cuatro
- Variable B: entrada lógica difusa
 - Valores: S2, S1, CE, B1, B2
 - S2: 0, 0.5, 1
 - S1: 0.5, 1, 1.5
 - CE: 1, 1.5, 2
 - B1: 1.5, 2, 2.5
 - B2: 2, 2.5, 3
- Variable C: salida categórica
 - Valores: a, b, c

2. Base de reglas:

- R1: 0.5 weight
 - Antecedentes: A = uno
 - Consecuente: C = c
- R2: 0.837 weight
 - Antecedentes: A = dos, cuatro; B = CE, B1
 - Consecuente: C = a
- R3: 0.234 weight
 - Antecedentes: B = S2, B2
 - Consecuente: C = c
- R4: 0.9459 weight
 - Antecedentes: B = S2; A = cuatro
 - Consecuente: C = b

13.2. ORCA

Los cambios de este módulo han tenido como objetivo el adaptarlo a la nueva forma de leer los datos, por lo que el número de pruebas no es muy alto. Sin embargo, se han hecho pequeñas pruebas de caja blanca con las que se ha podido comprobar que estos cambios actúan de manera correcta.

En las Tablas 13.3 y 13.4 se explican las pruebas realizadas en este módulo (sin tener en cuenta las pruebas entre módulos).

Prueba OR.01: Valor de 'archive'	
Descripción	Comprueba que se detecta correctamente la nueva variable 'archive' del fichero de configuración.
Exito	1. Lee correctamente el valor de la variable. 2. Si en el fichero de configuración no está la variable, por defecto su valor será 'matlab'
Fallo	1. No lee el valor de la variable. 2. No le asigna el valor por defecto en el caso de que no esté la variable en el fichero.

Tabla 13.3: Prueba OR.01: Valor de 'archive'

Prueba OR.02: Acceso de Directorios	
Descripción	Según el valor de la variable 'archive' se accederá al directorio donde se encuentran los <i>datasets</i> .
Exito	1. Accede a los directorios correctamente.
Fallo	1. Falla en acceder a los directorios aunque existan.

Tabla 13.4: Prueba OR.02: Acceso de Directorios

13.3. Read File

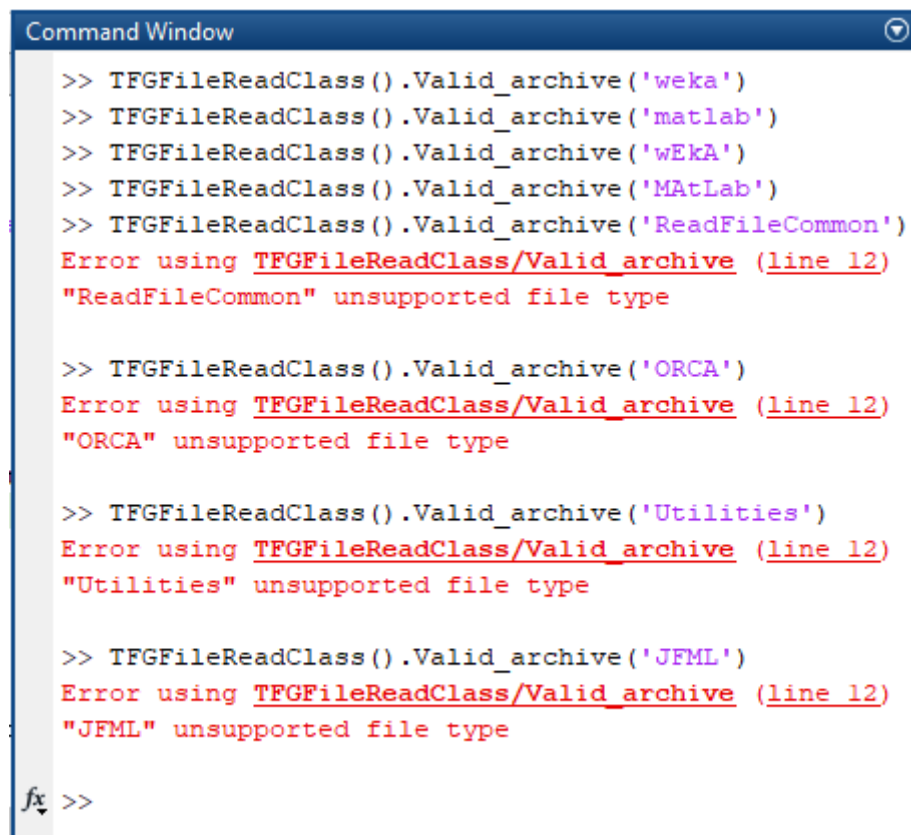
Como en este módulo ya se sabe como deben ser las salidas a partir de una entrada (los datos de las salidas deben concordar con los datos de los ficheros de entrada) se han realizado pruebas de caja negra con diferentes ficheros de los *datasets* establecidos anteriormente.

13.3.1. Tipo de Fichero Valido

En la Tabla 13.5 se muestra la especificación de la prueba que consiste en comprobar que el tipo de fichero especificado sea valido. En el caso de este proyecto solo se permite que sea de tipo ‘matlab’ o ‘weka’ sin ser sensible a mayúsculas y minúsculas. En la Figura 13.1, se puede ver un ejemplo de dicha prueba.

Prueba RF.01: Tipo de Fichero Válido	
Descripción	Comprueba que el tipo de fichero establecido en el fichero de configuración del experimento sea válido. Para ello, ejecutamos la función de la clase <i>TFGFileReadClass</i> (Tabla 11.5) que se encarga de esto con diferentes entradas para comprobar que si el tipo no es ‘matlab’ o ‘weka’, se produzca un error.
Exito	1. Continúa con el experimento si el tipo de fichero es válido. 2. Produce un error si el tipo de fichero no es valido
Fallo	1. No detecta como válido un tipo de fichero que sí lo es. 2. Detecta como válido un tipo de fichero que no lo es.

Tabla 13.5: Prueba RF.01: Tipo de Fichero Válido

A screenshot of a MATLAB Command Window. The window has a blue title bar with the text "Command Window" and a small icon on the right. The main area is white with black text. The text shows a series of commands and error messages. The commands are: `>> TFGFileReadClass().Valid_archive('weka')`, `>> TFGFileReadClass().Valid_archive('matlab')`, `>> TFGFileReadClass().Valid_archive('wEkA')`, `>> TFGFileReadClass().Valid_archive('MATLab')`, `>> TFGFileReadClass().Valid_archive('ReadFileCommon')`, `>> TFGFileReadClass().Valid_archive('ORCA')`, `>> TFGFileReadClass().Valid_archive('Utilities')`, and `>> TFGFileReadClass().Valid_archive('JFML')`. The error messages are: `Error using TFGFileReadClass/Valid_archive (line 12)` followed by `"ReadFileCommon" unsupported file type`, `"ORCA" unsupported file type`, `"Utilities" unsupported file type`, and `"JFML" unsupported file type`. At the bottom left, there is a small icon and the text `>>`.

```
>> TFGFileReadClass().Valid_archive('weka')
>> TFGFileReadClass().Valid_archive('matlab')
>> TFGFileReadClass().Valid_archive('wEkA')
>> TFGFileReadClass().Valid_archive('MATLab')
>> TFGFileReadClass().Valid_archive('ReadFileCommon')
Error using TFGFileReadClass/Valid_archive (line 12)
"ReadFileCommon" unsupported file type

>> TFGFileReadClass().Valid_archive('ORCA')
Error using TFGFileReadClass/Valid_archive (line 12)
"ORCA" unsupported file type

>> TFGFileReadClass().Valid_archive('Utilities')
Error using TFGFileReadClass/Valid_archive (line 12)
"Utilities" unsupported file type

>> TFGFileReadClass().Valid_archive('JFML')
Error using TFGFileReadClass/Valid_archive (line 12)
"JFML" unsupported file type

fx >>
```

Figura 13.1: Prueba para comprobar si el tipo de ficheros es valido

13.3. Read File

13.3.2. Leer datos

En la Tabla 13.6 se muestra la prueba general que se ha hecho a las lecturas de datos tanto para ficheros de tipo Matlab como para los de formato WEKA. En el caso de los de tipo WEKA, también se han hecho otras pruebas que se ven reflejadas en las Tablas 13.7 y 13.8.

En las Tablas 13.9-13.12 se muestran unos ejemplos de las pruebas que se explican en la Tabla 13.7 en las que se puede ver que con una cabecera mal definida, se detectan los errores.

Prueba RF.02: Lectura de Datos	
Descripción	Con diferentes ficheros de datos, algunos con errores e incluso sin ser del tipo correcto, se comprueba que en la lectura de ficheros se devuelven los datos correctos y sin ningún valor no valido.
Exito	1. Devuelve los datos del fichero con solo valores permitidos. 2. Produce un error si hay un carácter entre los datos (en el caso de ‘matlab’).
Fallo	1. Permite leer un fichero que no tiene el formato que le corresponde. 2. Entre los datos devueltos hay un valor no válido.

Tabla 13.6: Prueba RF.02: Lectura de Datos

Prueba RF.03: Lectura de la Cabecera	
Descripción	Se comprueba que la cabecera de los ficheros de tipo WEKA se lean correctamente y en el caso de que haya algo que no debe, se produzca un error para que se corrija en el fichero.
Exito	1. La cabecera se lee correctamente. 2. Produce error si hay algún atributo que no sea numérico o categórico.
Fallo	1. Lee la cabecera con un atributo que no es numérico ni categórico. 2. Continúa la ejecución aunque no se encuentre ‘@data’. 3. Acepta dos atributos o más con el mismo nombre.

Tabla 13.7: Prueba RF.03: Lectura de la Cabecera

Prueba RF.04: Conversión de los Datos Categóricos	
Descripción	En 'weka' cuando un algoritmo no permite datos categóricos, hay que convertirlos a numérico. Esta prueba sirve para garantizar que un atributo categórico de entrada se convierta en <i>One-Hot</i> (explicado en el requisito funcional 04 en la subsección 8.2.2) y las salidas en un valor numérico simple.
Exito	<ol style="list-style-type: none"> 1. Los categóricos se han convertido en numéricos correctamente. 2. En el caso que haya un dato que no corresponde a uno de los valores del categórico, éste se convierta en 'NaN'.
Fallo	<ol style="list-style-type: none"> 1. Convierte datos que no existe como valor en el categórico sin convertirlo en 'NaN'. 2. En el caso de 'One-Hot', los atributos no se actualicen añadiendo nuevos por cada variable que se haya creado.

Tabla 13.8: Prueba RF.04: Conversión de los Datos Categóricos

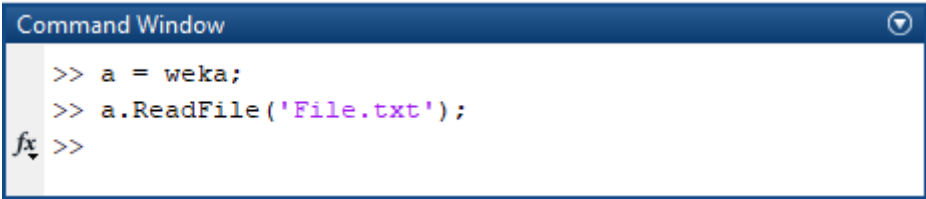
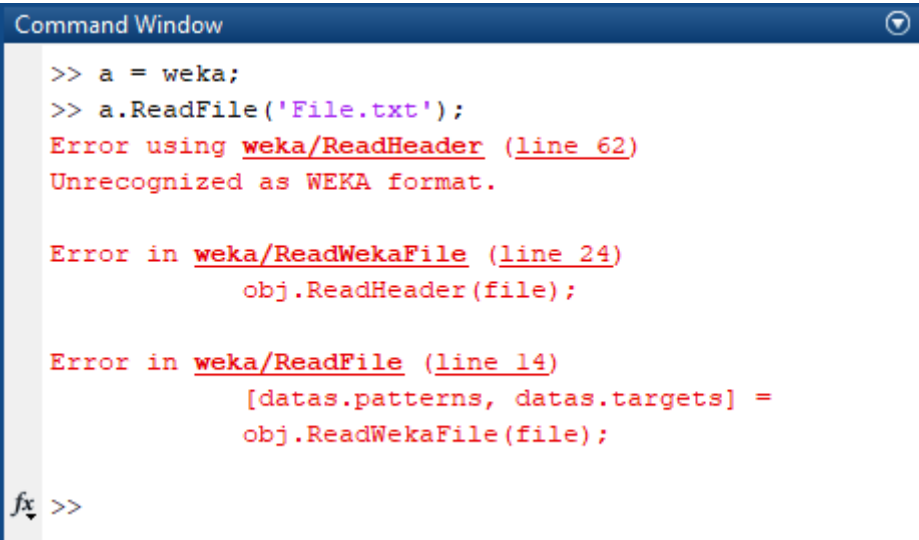
<pre> @relation NSLVOrd @attribute att1 numeric @attribute att2 {a1,a2,a3,a4} @attribute att3 numeric @attribute att4 {b1,b2,b3} @attribute y {a,b,c,d} @data </pre>
 <pre> Command Window >> a = weka; >> a.ReadFile('File.txt'); fx >> </pre>

Tabla 13.9: Prueba RF.03.1: Cabecera correcta

13.3. Read File

```
@relation NSLVOld

@attribute att1 numeric
@attribute att2 {a1,a2,a3,a4}
@attribute att3 numeric
@attribute att4 {b1,b2,b3}
@attribute y {a,b,c,d}
```



The image shows a MATLAB Command Window with a blue title bar and a dropdown arrow. The window contains the following text:

```
>> a = weka;
>> a.ReadFile('File.txt');
Error using weka/ReadHeader (line 62)
Unrecognized as WEKA format.

Error in weka/ReadWekaFile (line 24)
    obj.ReadHeader(file);

Error in weka/ReadFile (line 14)
    [datas.patterns, datas.targets] =
        obj.ReadWekaFile(file);

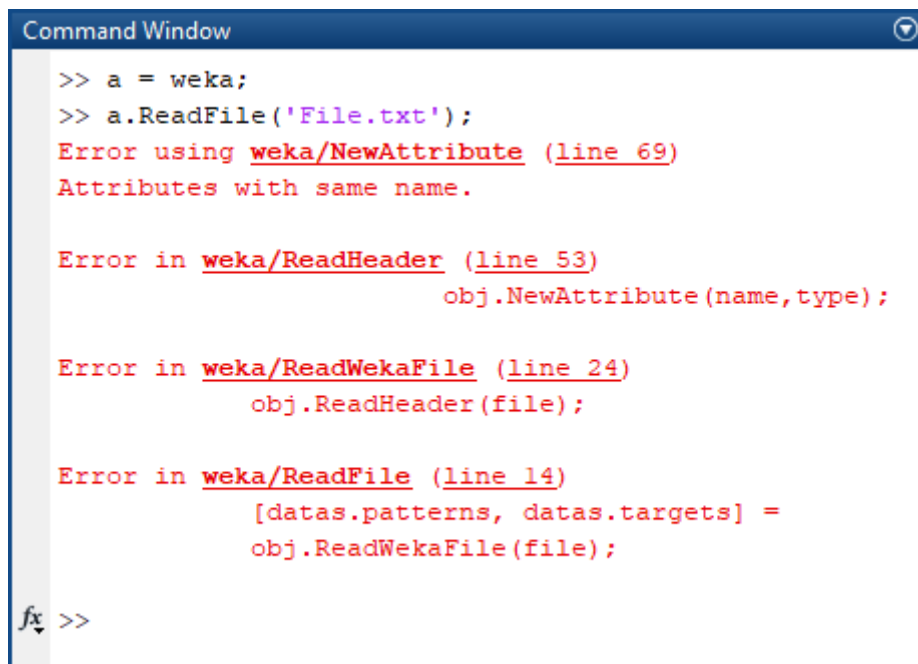
fx >>
```

Tabla 13.10: Prueba RF.03.2: Cabecera sin @data

```
@relation NSLVOrd

@attribute att1 numeric
@attribute att2 {a1,a2,a3,a4}
@attribute att2 numeric
@attribute att4 {b1,b2,b3}
@attribute y {a,b,c,d}

@data
```



Command Window

```
>> a = weka;
>> a.ReadFile('File.txt');
Error using weka/NewAttribute (line 69)
Attributes with same name.

Error in weka/ReadHeader (line 53)
        obj.NewAttribute(name,type);

Error in weka/ReadWekaFile (line 24)
        obj.ReadHeader(file);

Error in weka/ReadFile (line 14)
        [datas.patterns, datas.targets] =
        obj.ReadWekaFile(file);

fx >>
```

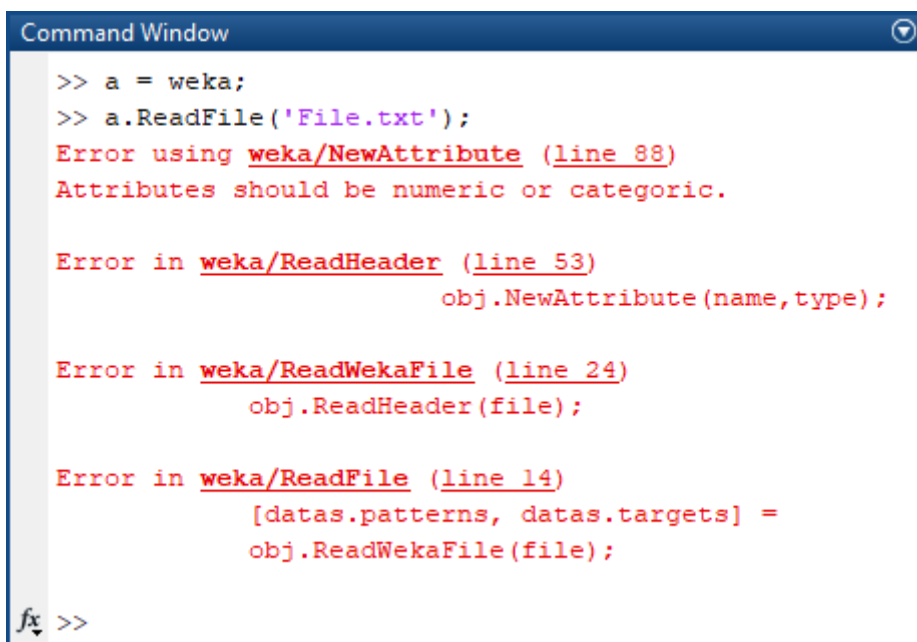
Tabla 13.11: Prueba RF.03.3: Cabecera con atributos repetidos

13.3. Read File

```
@relation NSLVOrd

@attribute att1 numeric
@attribute att2 {a1,a2,a3,a4}
@attribute att3
@attribute att4 {b1,b2,b3}
@attribute y {a,b,c,d}

@data
```



The screenshot shows a MATLAB Command Window with a blue title bar. The window contains the following text:

```
>> a = weka;
>> a.ReadFile('File.txt');
Error using weka/NewAttribute (line 88)
Attributes should be numeric or categoric.

Error in weka/ReadHeader (line 53)
    obj.NewAttribute(name,type);

Error in weka/ReadWekaFile (line 24)
    obj.ReadHeader(file);

Error in weka/ReadFile (line 14)
    [datas.patterns, datas.targets] =
    obj.ReadWekaFile(file);

fx >>
```

Tabla 13.12: Prueba RF.03.4: Cabecera con un atributo erroneo

13.4. Algorithms

A este módulo se le han hecho pruebas de caja blanca con las que se ha comprobado que las funciones de la clase *NSLVOrd* realicen su trabajo de manera correcta con los diferentes parámetros que afectan a su ejecución.

Prueba AL.01: Entrenamiento y Test	
Descripción	Pruebas del entrenamiento y test en las que se han ido comprobando el correcto funcionamiento de la configuración de los datos que se pasarán al algoritmo (cabecera, datos y parámetros) según el tipo de <i>datasets</i> del experimento. Además del procesamiento de los datos devueltos por el algoritmo
Exito	<ol style="list-style-type: none"> 1. La cabecera, datos y parámetros son aceptados por la función del algoritmo. 2. En el caso del entrenamiento, los datos guardados en el modelo son usados por el test tal y como se pretendía. 3. Las funciones devuelven los resultados con el formato y dimensiones adecuadas.
Fallo	<ol style="list-style-type: none"> 1. El formato de los resultados no era validos. 2. Las dimensiones de los resultados concordaban con los resultados reales. 3. La cabecera no tiene los atributos adecuados. 4. Al configurar los datos, estos no concuerdan con los datos esperados.

Tabla 13.13: Prueba AL.01: Entrenamiento y Test

Prueba AL.02: Configuración de datos	
Descripción	Los datos que se van a pasar al algoritmo deben ser primero procesados convirtiéndolos en vectores de tipo 'char'. En esta prueba se le han pasado datos específicos a las funciones de la clase <i>NSLVOrd</i> 11.10 y se ha comprobado si los resultados eran los esperados.
Exito	<ol style="list-style-type: none"> 1. Los resultados concuerdan con lo establecido.
Fallo	<ol style="list-style-type: none"> 1. La cabecera no concuerda con la esperada. 2. Durante la conversión de los <i>datasets</i> se han perdido datos o no los ha convertido bien.

Tabla 13.14: Prueba AL.02: Configuración de datos

13.5. NSLVOrd

Prueba AL.03: Procesar salidas	
Descripción	Pruebas realizadas a la conversión de los resultados de tipo 'char' a su respectivo valor numérico y al guardado de los datos necesarios en el modelo.
Exito	<ol style="list-style-type: none">1. Los resultados se convierten en numéricos correspondiendo a como se representan.2. Los diferentes datos son guardados y usados en las demás funciones correctamente.
Fallo	<ol style="list-style-type: none">1. En los resultados, para dos datos con el mismo valor, se convierten en un numérico diferente el uno del otro.2. Al guardar los datos en el modelo, no son los mismos cuando se usan en otra función.

Tabla 13.15: Prueba AL.03: Procesar salidas

13.5. NSLVOrd

Como se dijo en la sección 9.1, este módulo solo ha sido modificado para adaptarlo a las necesidades de este trabajo. Aún así, se han hecho pruebas de caja blanca y caja negra con ayuda de los *datasets* de la sección 13.1.

Con las pruebas de caja blanca se ha comprobado que los cambios que se han hecho no han afectado al funcionamiento normal del algoritmo y que las funcionalidades añadidas actúan como estaba predicho.

Por otro lado, con las pruebas de caja negra se han realizado experimentos tantos con la versión original como con la modificada para poder comparar los resultados y comprobar que los cambios no afecta al algoritmo y no altera los resultados. Estas pruebas están expuestas en la Tablas 13.16.

Prueba NSL.01: Adaptación de las Entradas y Salidas	
Descripción	Se han hecho pruebas introduciendo parámetros específicos y se ha comprobado con ayuda de la versión original que los datos devueltos son correctos, tanto los resultados del entrenamiento y test como las bases de conocimiento y reglas.
Exito	<ol style="list-style-type: none"> 1. Los resultados del entrenamiento y test concuerdan con la ejecución de la versión original del algoritmo. 2. El sistema de reglas concuerda con el de la versión original del algoritmo.
Fallo	<ol style="list-style-type: none"> 1. Los resultados varían con los de la versión original. 2. La base de reglas tiene reglas diferentes.

Tabla 13.16: Prueba NSL.01: Adaptación de las Entradas y Salidas

13.6. Rule View

Para este módulo, se han usado pruebas de caja blanca para la parte codificada en Java para garantizar que las reglas se creen de forma adecuada según su tipo y la cantidad de valores que muestra para una variable. Además, se han hecho pruebas de caja negra para el módulo completo como se especifica en la Tabla 13.17 y se puede ver un ejemplo en la Figura 13.2 con respecto al sistema de reglas de la sección 13.1.

Prueba RV.01: Visualizar el Sistema de Reglas	
Descripción	Usando la clase especificada en la Tabla 11.13 se ha introducido una base de reglas específica y se ha observado si la representaba correctamente.
Exito	1. Están todas las reglas con los antecedentes y consecuentes correctos.
Fallo	<ol style="list-style-type: none"> 1. Faltan antecedentes en algunas reglas o están en otra. 2. No están todas las reglas representadas.

Tabla 13.17: Prueba RV.01: Visualizar el Sistema de Reglas

13.6. Rule View



Figura 13.2: Prueba del módulo *Rule View*

13.7. JFML

Como la parte codificada en Java no hay necesidad de probarla, solo se han hecho pruebas de caja negra como se especifica en la Tabla 13.18.

Prueba RE.01: Exportar el Sistema de Reglas	
Descripción	Usando la clase 11.18 se ha introducido un sistema de reglas específico y se ha observado si se exportaba correctamente.
Exito	<ol style="list-style-type: none"> 1. La base de conocimiento y de reglas coincide con los datos. 2. Los ficheros se encuentran en la carpeta especificada
Fallo	<ol style="list-style-type: none"> 1. Faltan antecedentes en algunas reglas o están en otra. 2. No están todas las reglas representadas. 3. La base de conocimiento no tienen los atributos bien representados. 4. En las reglas los valores de las variables no coinciden con ninguno de la base de conocimiento.

Tabla 13.18: Prueba RV.01: Visualizar el Sistema de Reglas

A continuación, se mostrará como sería el fichero en formato JFML que generaría el sistema de reglas de la sección 13.1:

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <fuzzySystem xmlns="http://www.ieee1855.org">
3   <knowledgeBase>
4     <fuzzyVariable name="A" scale="" domainleft="0.0"
5       domainright="3.0" type="input">
6       <fuzzyTerm name="uno" complement="false">
7         <trapezoidShape param1="0.0" param2="0.0" param3
8           ="0.0" param4="0.0"/>
9       </fuzzyTerm>
10      <fuzzyTerm name="dos" complement="false">
11        <trapezoidShape param1="1.0" param2="1.0" param3
12          ="1.0" param4="1.0"/>
13      </fuzzyTerm>
14      <fuzzyTerm name="tres" complement="false">
15        <trapezoidShape param1="2.0" param2="2.0" param3
16          ="2.0" param4="2.0"/>
17      </fuzzyTerm>

```

```

14      <fuzzyTerm name="cuatro" complement="false">
15          <trapezoidShape param1="3.0" param2="3.0" param3
              ="3.0" param4="3.0"/>
16      </fuzzyTerm>
17  </fuzzyVariable>
18  <fuzzyVariable name="B" scale="" domainleft="0.0"
              domainright="3.0" type="input">
19      <fuzzyTerm name="S2" complement="false">
20          <trapezoidShape param1="0.0" param2="0.5" param3
              ="0.5" param4="1.0"/>
21      </fuzzyTerm>
22      <fuzzyTerm name="S1" complement="false">
23          <trapezoidShape param1="0.5" param2="1.0" param3
              ="1.0" param4="1.5"/>
24      </fuzzyTerm>
25      <fuzzyTerm name="CE" complement="false">
26          <trapezoidShape param1="1.0" param2="1.5" param3
              ="1.5" param4="2.0"/>
27      </fuzzyTerm>
28      <fuzzyTerm name="B1" complement="false">
29          <trapezoidShape param1="1.5" param2="2.0" param3
              ="2.0" param4="2.5"/>
30      </fuzzyTerm>
31      <fuzzyTerm name="B2" complement="false">
32          <trapezoidShape param1="2.0" param2="2.5" param3
              ="2.5" param4="3.0"/>
33      </fuzzyTerm>
34  </fuzzyVariable>
35  <fuzzyVariable name="C" scale="" domainleft="0.0"
              domainright="2.0" type="output">
36      <fuzzyTerm name="a" complement="false">
37          <trapezoidShape param1="0.0" param2="0.0" param3
              ="0.0" param4="0.0"/>
38      </fuzzyTerm>
39      <fuzzyTerm name="b" complement="false">
40          <trapezoidShape param1="1.0" param2="1.0" param3
              ="1.0" param4="1.0"/>
41      </fuzzyTerm>
42      <fuzzyTerm name="c" complement="false">
43          <trapezoidShape param1="2.0" param2="2.0" param3
              ="2.0" param4="2.0"/>
44      </fuzzyTerm>

```

```

45     </fuzzyVariable>
46 </knowledgeBase>
47
48 <mamdaniRuleBase name="" activationMethod="MIN" andMethod="
MIN" orMethod="MAX">
49     <rule name="R1" andMethod="MIN" connector="and" weight="
0.5">
50         <antecedent>
51             <clause>
52                 <variable>A</variable>
53                 <term>uno</term>
54             </clause>
55         </antecedent>
56         <consequent>
57             <then>
58                 <clause>
59                     <variable>C</variable>
60                     <term>c</term>
61                 </clause>
62             </then>
63         </consequent>
64     </rule>
65     <rule name="R2_1" andMethod="MIN" connector="and" weight
="0.837">
66         <antecedent>
67             <clause>
68                 <variable>A</variable>
69                 <term>dos</term>
70             </clause>
71             <clause>
72                 <variable>B</variable>
73                 <term>CE</term>
74             </clause>
75         </antecedent>
76         <consequent>
77             <then>
78                 <clause>
79                     <variable>C</variable>
80                     <term>a</term>
81                 </clause>
82             </then>
83         </consequent>

```

```
84     </rule>
85     <rule name="R2_2" andMethod="MIN" connector="and" weight
      ="0.837">
86         <antecedent>
87             <clause>
88                 <variable>A</variable>
89                 <term>dos</term>
90             </clause>
91             <clause>
92                 <variable>B</variable>
93                 <term>B1</term>
94             </clause>
95         </antecedent>
96         <consequent>
97             <then>
98                 <clause>
99                     <variable>C</variable>
100                     <term>a</term>
101                 </clause>
102             </then>
103         </consequent>
104     </rule>
105     <rule name="R3_1" andMethod="MIN" connector="and" weight
      ="0.234">
106         <antecedent>
107             <clause>
108                 <variable>B</variable>
109                 <term>S2</term>
110             </clause>
111         </antecedent>
112         <consequent>
113             <then>
114                 <clause>
115                     <variable>C</variable>
116                     <term>c</term>
117                 </clause>
118             </then>
119         </consequent>
120     </rule>
121     <rule name="R3_2" andMethod="MIN" connector="and" weight
      ="0.234">
122         <antecedent>
```

```

123         <clause>
124             <variable>B</variable>
125             <term>B2</term>
126         </clause>
127     </antecedent>
128     <consequent>
129         <then>
130             <clause>
131                 <variable>C</variable>
132                 <term>c</term>
133             </clause>
134         </then>
135     </consequent>
136 </rule>
137 <rule name="R4" andMethod="MIN" connector="and" weight="
138     0.9459">
139     <antecedent>
140         <clause>
141             <variable>B</variable>
142             <term>S2</term>
143         </clause>
144         <clause>
145             <variable>A</variable>
146             <term>cuatro</term>
147         </clause>
148     </antecedent>
149     <consequent>
150         <then>
151             <clause>
152                 <variable>C</variable>
153                 <term>b</term>
154             </clause>
155         </then>
156     </consequent>
157 </rule>
158 </mamdaniRuleBase>
159 </fuzzySystem>

```


13.8. Sistema

Una vez finalizadas las pruebas individuales de los módulos, se han realizado diferentes pruebas al sistema completo y entre diferentes módulos. Se ha comprobado que los módulos intercambien información de forma correcta y mostrando las salidas, y se ha visto que salen las correctas al compararlas con las que se sacaron en la prueba de la sección 13.5.

En la Tabla 13.25 se hace una comparación del error absoluto medio (MAE) entre el algoritmo original y usando en ORCA para comprobar que la incorporación del algoritmo en ORCA ha sido un éxito.

Prueba SIS.01: ORCA - Read File	
Descripción	Prueba de interacción entre el módulo ORCA y Read File.
Exito	1. Los directorios donde se encuentran los <i>datasets</i> se han pasado correctamente. 2. Se han devuelto los datos correctos.
Fallo	1. El directorio es incorrecto. 2. Los datos devueltos tienen fallos.

Tabla 13.19: Prueba SIS.01: ORCA - Read File

Prueba SIS.02: ORCA - Algorithms	
Descripción	Prueba de interacción entre el módulo ORCA y Algorithms.
Exito	1. El intercambio de datos tienen valores correctos.
Fallo	1. Los datos de entrenamiento y test no coinciden con los originales. 2. Los resultados se pasan como un tipo no valido.

Tabla 13.20: Prueba SIS.02: ORCA - Algorithms

Prueba SIS.03: Algorithms - NSLVOrd	
Descripción	Prueba de interacción entre el módulo Algorithms y NSLVOrd.
Exito	1. Los parámetros coinciden con los creados. 2. Devuelve los resultados esperados.
Fallo	1. Los parámetros se pasan en un formato no valido. 2. Los resultados no son los esperados.

Tabla 13.21: Prueba SIS.03: Algorithms - NSLVOrd

Prueba SIS.04: Algorithms - Rule View	
Descripción	Prueba de interacción entre el módulo Algorithms y Rule View.
Exito	1. Las reglas se configuran correctamente.
Fallo	1. Faltan reglas. 2. Las reglas están mal configuradas.

Tabla 13.22: Prueba SIS.04: Algorithms - Rule View

Prueba SIS.05: Algorithms - JFML	
Descripción	Prueba de interacción entre el módulo Algorithms y JFML.
Exito	1. El sistema de reglas se configura correctamente.
Fallo	1. Los atributos y sus valores no están correctos en la base de conocimiento. 2. Faltan reglas o están mal creadas.

Tabla 13.23: Prueba SIS.05: Algorithms - JFML

Prueba SIS.06: Sistema	
Descripción	Prueba de la ejecución del sistema que comprueba la correcta incorporación de NSLVOrd y de los datos categóricos. Se realizan ejecuciones y se comparan los datos estadísticos.
Exito	1. Los resultados serán los mismos que en un experimento de la versión original del algoritmo.
Fallo	1. Dan resultados diferentes.

Tabla 13.24: Prueba SIS.06: Sistema

13.8. Sistema

MMAE	NSLVOrd	ORCA
automobile	0,0386	0,038562233
balance-scale	0,1207	0,1206553
bondrate	0,0191	0,019047733
contact-lenses	0,0167	0,0166668
ERA	1,2703	1,270222233
ESL	0,2867	0,286611967
eucalyptus	0,0305	0,030434833
LEV	0,3845	0,384400067
newthyroid	0,0729	0,072877867
pasture	0,0321	0,032098733
squash-stored	0,0197	0,0196581
squash-unstored	0,0265	0,0264957
SWD	0,3856	0,385511167
tae	0,1812	0,181120967
toy	0,1955	0,1954074
winequality-red	0,3039	0,303836467

Tabla 13.25: Prueba SIS.07: Comparación del MAE

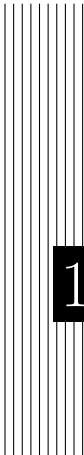
13.9. Errores Durante el Desarrollo

En esta sección se mostrarán los errores más significativos que se han producido durante el desarrollo de este proyecto.

- **Error al acceder a los datasets de tipo ‘weka’:** este error se produjo porque ORCA especificaba en la configuración del experimento que los datasets se debían encontrar en una carpeta llamada ‘matlab’. Se corrigió haciendo que ORCA especificara la carpeta según el tipo de dato que indica el usuario.
- **Error al validar un tipo de fichero:** aunque el tipo de fichero era correcto, se identificó como incorrecto por la forma de especificar la ruta en la que se encontraban las clases que leen los ficheros, que identificaban el separador ‘como un carácter especial. Esto se ha solucionando usando funciones que detectan el directorio actual y que unen directorios.
- **Al convertir los datos categóricos en numéricos, todos se identificaban como no validos:** este error se produjo por intentar tratar a datos de tipo *char* como si fuesen numéricos y usarlos en matrices, provocando que al comparar los datos, ninguno se identificase correctamente.
- **Al procesar los resultados, el número de salidas no coincidía con las que se comparaba:** este error se debe a que la eliminación de valores no validos solo se hacía con los datos que se leían a la hora de realizar el experimento, pero no cuando se leía para obtener los resultados originales y compararlos. Esto se ha resuelto haciendo que se eliminen los datos no validos siempre que se lee un fichero.
- **Los resultados de NSLVOrd en ORCA no coincidía con los resultados de NSLVOrd original:** al adaptar el algoritmo para poder usarlo en ORCA, se debió modificar algo que producía alguna alteración en el entrenamiento y por consecuente, que salieran diferentes resultados a los esperados. No se sabe cual fue el error, ya que se rehizo desde cero todas las modificaciones y se arregló el problema.

13.9. Errores Durante el Desarrollo

- **No permitía guardar el modelo entrenado:** al principio se intentaba guardar en el modelo objetos Java, pero eso es algo que no se puede hacer y provocaba un error a la hora de cargar el modelo para realizar el test. Esto se solucionó convirtiendo el modelo creado en datos de tipo *char*.



14 Conclusiones y Futuras Mejoras

Para acabar este proyecto, en este capítulo se hablará sobre las conclusiones que se han obtenido de este trabajo y que mejoras se podrían realizar a futuro.

14.1. Objetivos del Problema

Para empezar, se dará un pequeño repaso a los objetivos que están más detallados en el capítulo 3:

1. **Incluir NSLVOrd en ORCA:** NSLVOrd es un algoritmo de clasificación ordinal y se pretende incluir entre los métodos de ORCA.
2. **Incluir ficheros de formato WEKA en ORCA:** se pretende que ORCA pueda leer datos categóricos y para ello los leerá de ficheros con formato WEKA.
3. **Visualización de las reglas difusas de NSLVOrd:** se pretende que ORCA pueda mostrar de forma visual las reglas generadas en el aprendizaje.

En este trabajo se ha estado abordando la elaboración de estos objetivos y se han podido realizar de forma que todos estos han sido completados y cumpliendo todos los requisitos establecidos en el capítulo 8.

14.2. Nivel Personal

Durante el desarrollo de este trabajo se han obtenido diferentes conceptos, métodos y habilidades que se podrán aplicar en futuros trabajos:

- Aunque en el grado se haya trabajado con Matlab y Java, solo se dio de manera superficial y para dar a conocer diferentes conceptos. Sin embargo, en este trabajo se ha podido ver un poco más de estos lenguajes y aprenderlos de forma más profunda.
- Se ha obtenido la experiencia de poder mostrar en un trabajo más complejo las capacidades que se han obtenido a lo largo del grado.
- Se ha experimentado el hecho de tener que trabajar con sistemas ya elaborados teniendo que analizarlos y comprender su funcionamiento.

14.3. Futuras Mejoras

En un principio, los cambios realizados en este trabajo se podrían dar por terminado y completo, sin embargo, siempre se pueden hacer mejoras. A continuación, se mostrará las posibles mejoras:

- Actualizar el módulo NSLVOrd en el caso de que salgan nuevas versiones del algoritmo homónimo.
- Actualizar el módulo JFML en el caso de que salgan nuevas versiones del algoritmo homónimo.
- Incluir nuevos métodos para leer más tipos de ficheros.



BIBLIOGRAFÍA

- [1] <http://www.uco.es/grupos/ayrna/ucobigfiles/datasets-orreview.zip>, accedido el 06-06-2020
- [2] Grupo de Investigación en Aprendizaje y Redes Neuronales Artificiales. Universidad de Córdoba. <http://www.uco.es/grupos/ayrna>, accedido el 06-06-2020
- [3] Matlab optimization toolbox (2014)
- [4] Eckel, B.: Thinking in Java. Prentice Hall, 4 edn. (2006)
- [5] Granados, J.G.: Uso de técnicas de aprendizaje para clasificación ordinal y regresión. Universidad de Granada (2017)
- [6] Guazzelli, A., Zeller, M., Lin, W.C., Williams, G.: Pmml: An open standard for sharing models. The R Journal 1, 60–65 (2009)
- [7] Gutiérrez, P., Pérez-Ortiz, M., Sánchez-Monedero, J., Fernandez-Navarro, F., Hervás-Martínez, C.: Ordinal regression methods: survey and experimental study. IEEE Transactions on Knowledge and Data Engineering 28(1), 127–146 (2016), <http://dx.doi.org/10.1109/TKDE.2015.2457911>
- [8] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations 11, 10–19 (2009)

- [9] John W. Eaton, David Bateman, S.H., Wehbring, R.: GNU Octave version 3.8.1 manual: a high-level interactive language for numerical computations. CreateSpace Independent Publishing Platform (2014), <http://www.gnu.org/software/octave/doc/interpreter>, ISBN 1441413006
- [10] Kernighan, B.W., Ritchie, D.M.: The C Programming Language. Prentice Hall Professional Technical Reference, 2nd edn. (1988)
- [11] Sánchez-Monedero, J., Gutiérrez, P.A., Pérez-Ortiz, M.: Orca: A matlab/octave toolbox for ordinal regression. *Journal of Machine Learning Research* 20(125), 1–5 (2019), <http://jmlr.org/papers/v20/18-349.html>
- [12] Soto-Hidalgo, J.M., Alonso, J.M., Acampora, G., Alcalá-Fdez, J.: Jfml: A java library to design fuzzy logic systems according to the iee Std 1855-2016. *IEEE Access* 6(1), 54952–54964 (December 2018)