TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

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

Manual Técnico

Autor

Federico García-Arévalo Calles

Directores

Pedro Antonio Gutiérrez Peña

Juan Carlos Gámez Granados

Escuela Politécnica Superior

—

Córdoba, mes de 2020

Índice General

Índice de Figuras

Índice de Tablas

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 NSLVOrd [5] en un *framework* de Matlab [3] llamado ORCA [8, 6], aunque como se verá en el capítulo 3 se añadirán más objetivos. ORCA trabaja con diferentes métodos de clasificación ordinal y NSLVOrd es un algoritmo de clasificación ordinal. Es por esto y porque NSLVOrd 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 principal problema real que se plantea en este Trabajo Fin de Grado es la inclusión del algoritmo NSLVOrd en el *framework* ORCA, sin embargo, se ha añadido como complemento a este, la capacidad de usar datos que estén en ficheros de formato Weka y la capacidad para representar las reglas de NSLVOrd. Es decir, este trabajo se puede dividir en los siguientes puntos:

1. ***Inclusión de NSLVOrd en ORCA***

   El grupo de investigación AYRNA, del Departamento de Informática y Análisis Numérico de la universidad de Córdoba, tiene un *framework* propio de Matlab 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 este 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. Es por esto que se plantea que ORCA tenga la capacidad de leer ficheros en formato WEKA y, de esta forma, permitir que los métodos que se quieran añadir a partir de ahora, puedan usar valores 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, esto 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 y PMML.

2.2. Identificación del Problema Técnico

Una vez establecido el problema real en el punto anterior, en este se expondrán los problemas técnicos del proyecto para establecer los condicionantes del problema. 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 y NSLVOrd 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 :

- **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.
  2. Guardar las reglas en ficheros con formato JFML y PMML.
  3. Representar las reglas visualmente.

- **Nivel interno:** para el funcionamiento de los programas, ORCA y NSLVOrd solo sería tener que adaptarlos 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 y el segundo en Matlab. 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, los usuarios objetivo son aquellos que trabajen con este *framewor*. Es decir, los usuarios objetivo son aquellos a los que le interesa trabajar con los diferentes algoritmos de regresión ordinal que ofrece ORCA.

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, ya que es el lenguaje con el que trabaja ORCA. Y por otro lado, se usará Java debido a que NSLVOrd esta programado en este lenguaje.

2.2.3. Vida Esperada

En un principio, la vida esperada de esta aplicación pude ser bastante longeva. Esto se debe, sobre todo, a que ORCA es usado principalmente por sus desarrolladores, el equipo de investigación AYRNA, 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 o JFML si en un futuro reciben nuevas actualizaciones.

2.2.5. Fiabilidad

Para garantizar que producto final funcione correctamente y sin fallos extraños, se controlará que los datos sean correctos y tengan valores dentro de los límites. En el caso de que un valor no cumpliese 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 en ORCA. 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 y la capacidad de guardar y representar las reglas aprendidas.

3.1. Incluir NSLVOrd en ORCA

El grupo de investigación AYRNA, del Departamento de Informática y Análisis Numérico de la universidad de Córdoba, ha desarrollado un *framework* en Matlab llamado ORCA. 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. 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, tarea ya simplificada por ORCA y Matlab. ORCA facilita la inclusión de nuevos algoritmos haciendo que en una clase se deba incluir las funciones que representan a la fase de entrenamiento y test; y Matlab permite de forma sencilla incluir código en Java, necesario, ya que NSLVOrd está escrito en este lenguaje.

3.2. Incluir ficheros de formato Weka en ORCA

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

3.3. Visualización de las reglas difusas de NSLVOrd

Como NSLVOrd es un algoritmo que sirve para el aprendizaje de reglas difusas, se ha establecido como uno de los objetivos permitir la visualización de estas 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, 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 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 cuál pertenece un patrón.

4.2. ORCA

ORCA (Ordinal Regression and Classi?cation Algorithms [8, 6]) es un framework de Matlab [3] 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 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 [7] y se encuentra como un paquete Java.

4.4. JFML

JFML [9] es una biblioteca que permite la exportabilidad de sistemas de lógica difusa en formato XML siguiendo el estandar *IEEE Std 1855TM-2016*. A demás, permite la exportación de estos sistemas en otros formatos como es el de PMML.

4.5. Weka

Weka (Waikato Environment for Knowledge Analysis [7]) es una librería Java 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

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 este es un Trabajo de Fin de Grado, no hay un cliente que imponga unas restricciones que haya que cumplir, sin embargo, se pueden sacar algunas restricciones:

- Se usará el lenguaje de programación de Matlab, ya que ORCA está escrito en este mismo lenguaje.
- Se deberá cumplir con todos los objetivos que se han establecido en el capítulo 3.

5.2. Factores Estratégicos

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

- Para exportar las reglas creadas por el algoritmo, se usará la librería JFML, ya que permite representar en XML los sistemas de lógica difusa. Además, también permite exportarlo a PMML.
- Se usará Java en el proyecto, debido a que el algoritmo NSLVOrd ya está implementado en este lenguaje y como está a libre disposición, se reutilizará el código. Además, como se ha dicho en el punto anterior, se usará JFML para exportar las reglas, el cual también se encuentra implementado en este lenguaje.
- El formato de los ficheros de datos que permitan los categóricos serán de tipo Weka. Esto es debido a que es el tipo de fichero del que lee los datos NSLVOrd y es un formato conocido entre los que trabajan en aprendizaje automático.
- Los cambios realizados en ORCA deben ser compatibles con la versión de 2014 de Matlab y Octave.

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 y Java.

Parte II

Análisis del Sistema

7 Casos de Uso

En este capítulo se expondrán lo casos de uso que existirán para este sistema, de tal forma que nos permita obtener los posibles requisitos del sistema. Estos casos de uso permitirá 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 cuales 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	1. El usuario debe estar en la ruta donde se encuentra el fichero de la clase *Utilities*. 2. El usuario debe tener el fichero que contiene los parámetros del experimento.
Flujo principal	1. Configura el experimento: 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: 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 *cross-validate*. 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	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 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 *ExportRules*. 3. Añadir la base de reglas a la clase *ExportRules*. 4. Exportar las reglas usando el algoritmo JFML.
Postcondiciones	1. Se crearán dos *xml* 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 debe estar cargado.
Flujo	1. Obtener la base de conocimiento y las reglas del modelo. 2. Añadir la base de conocimiento a la clase *VisualRules*. 3. Añadir las reglas a la clase *VisualRules*. 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 linea 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 *privfit* de la clase NSLVOrd.
Autores	Usuario
Precondiciones	1. Pasar como argumento los datos de entrenamiento. 2. Pasar como argumento los parámetros de entrenamiento.
Flujo	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: 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: 7.1 Ver el flujo del CU-3 en la Tabla 7.3. 8. FINSI
Postcondiciones	1. Se guarda la información del modelo en *obj.model*. 2. Se devuelve las salidas que produce el modelo ya entrenado para los datos de entrada.

Tabla 7.4: CU-04: Realizar entrenamiento

CU-05	Realizar test.
Descripción	Proceso que sigue la función de test *privpredict* 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 *TO-ONE-HOT* 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 exprondrán 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.

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.
- 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*.
- 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.

Pero primero, se aclarará un 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

En esta sección se explicarán los módulos de este sistema. Para los módulos originales y modificados, se contará como una sola cosa, sin embargo, en el caso de ORCA se separará la lectura de datos y los algoritmos. En total, quedarían los siguientes 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 Figura9.1 se muestra gráficamente y de forma sencilla los módulos mencionados en la sección anterior y como están conectados entre ellos.

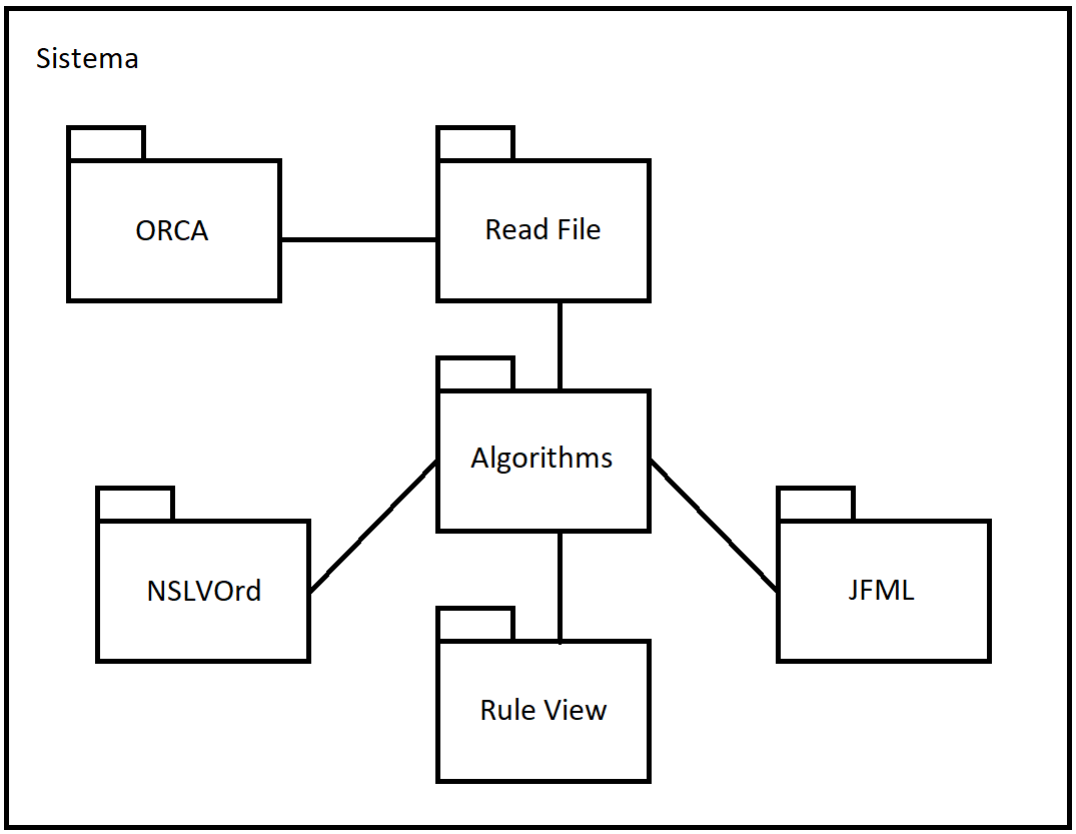


Figura 9.1: Estructura modular del sistema

Por último, se va a representar las relaciones de control entre los distintos módulos. El diagrama de la Figura9.2 muestra como son las interacciones entre las relaciones de los módulos.

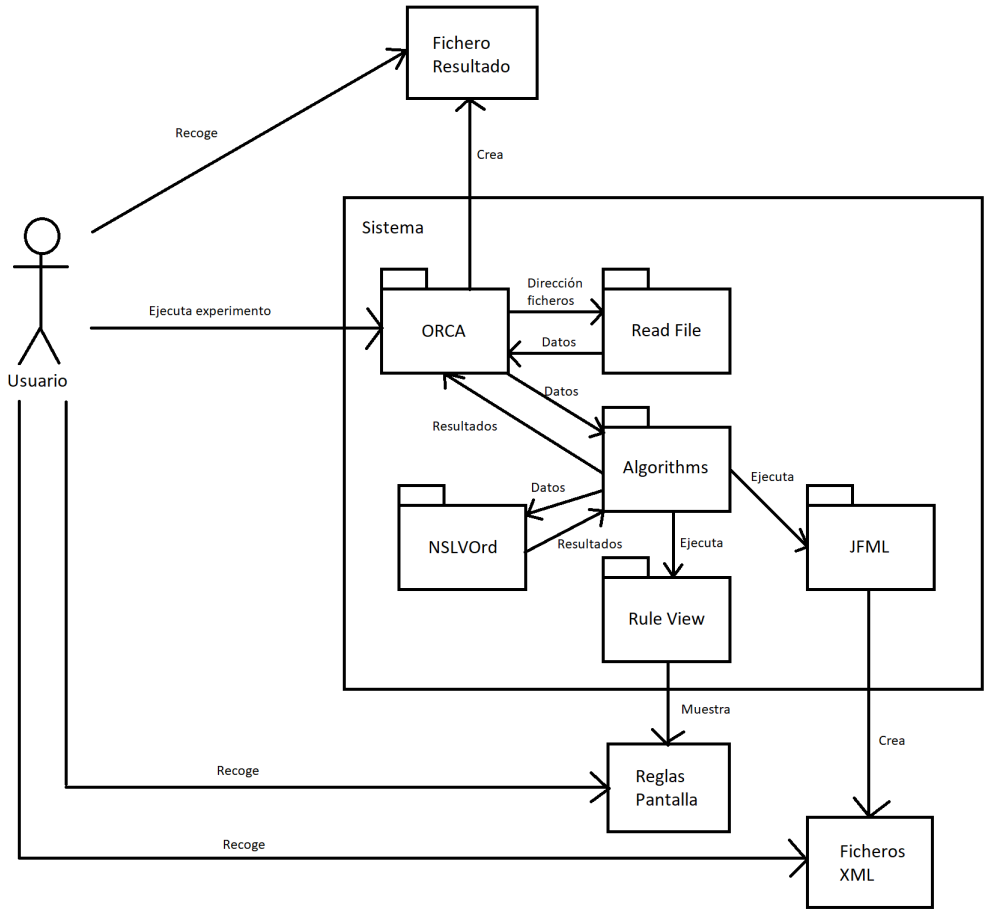


Figura 9.2: Estructura modular del sistema

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 Figura10.1 se muestra las interacciones del sistema cuando se ejecuta un experimento. Y por otro laso, en las Figura10.2 y Figura10.3 se muestran las interacciones cuando se exportan y se visualizan las reglas respectivamente, pudiendo ver que son muy similares.

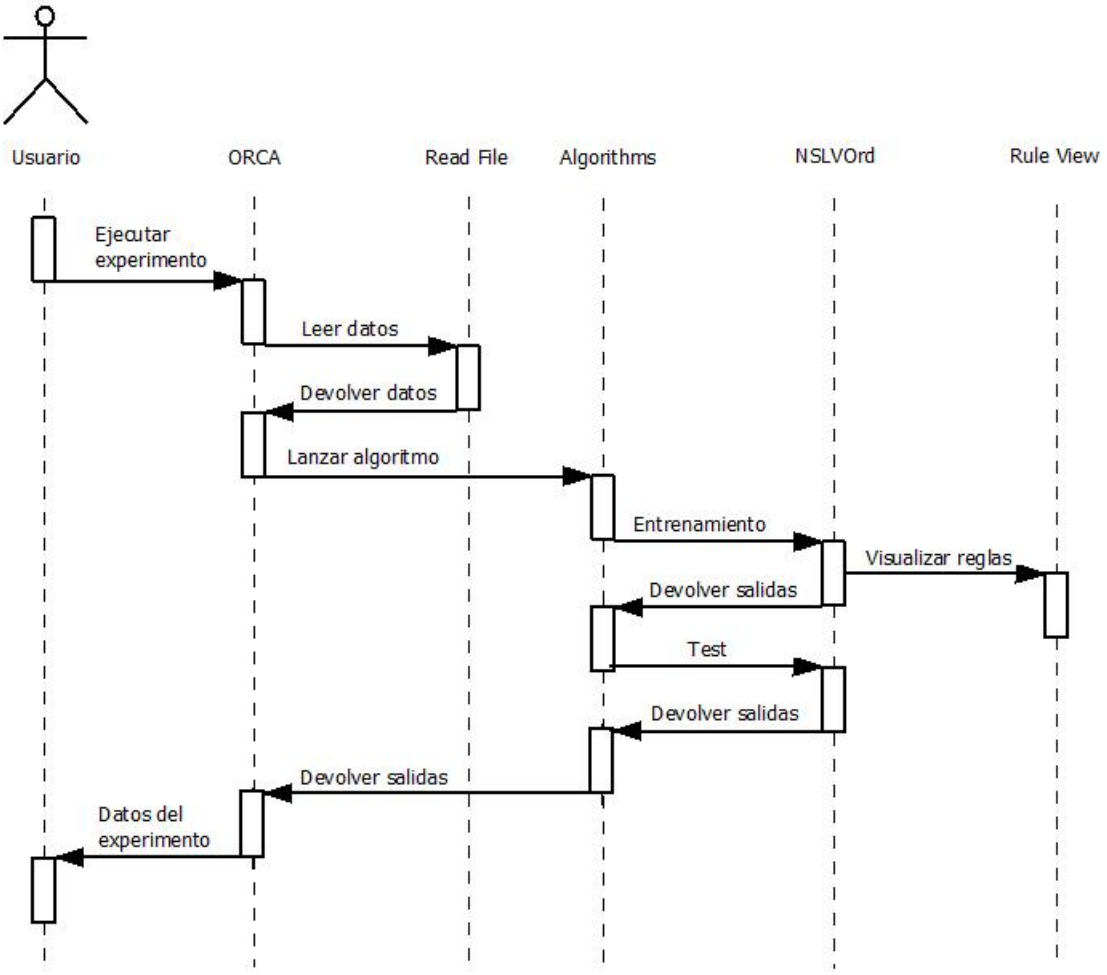


Figura 10.1: Diagrama de secuencia para la ejecución de un experimento.

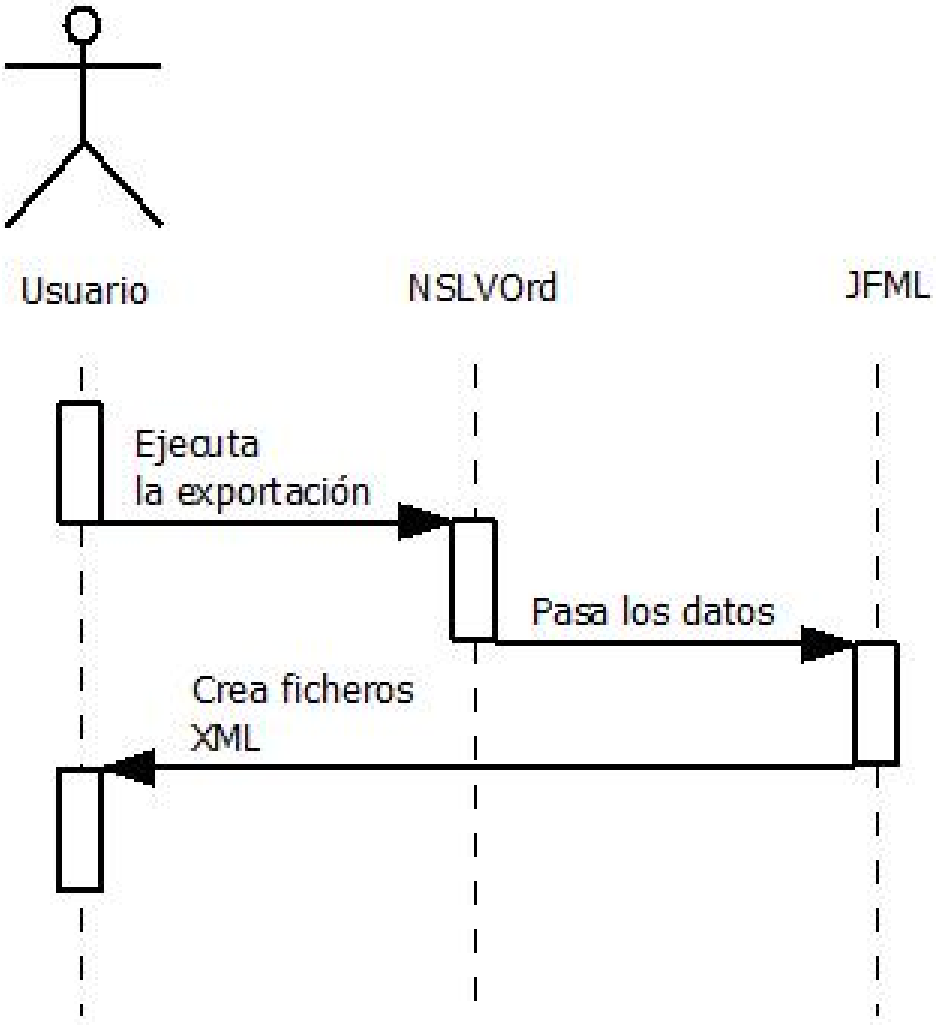


Figura 10.2: Diagrama de secuencia de la exportación de las reglas.

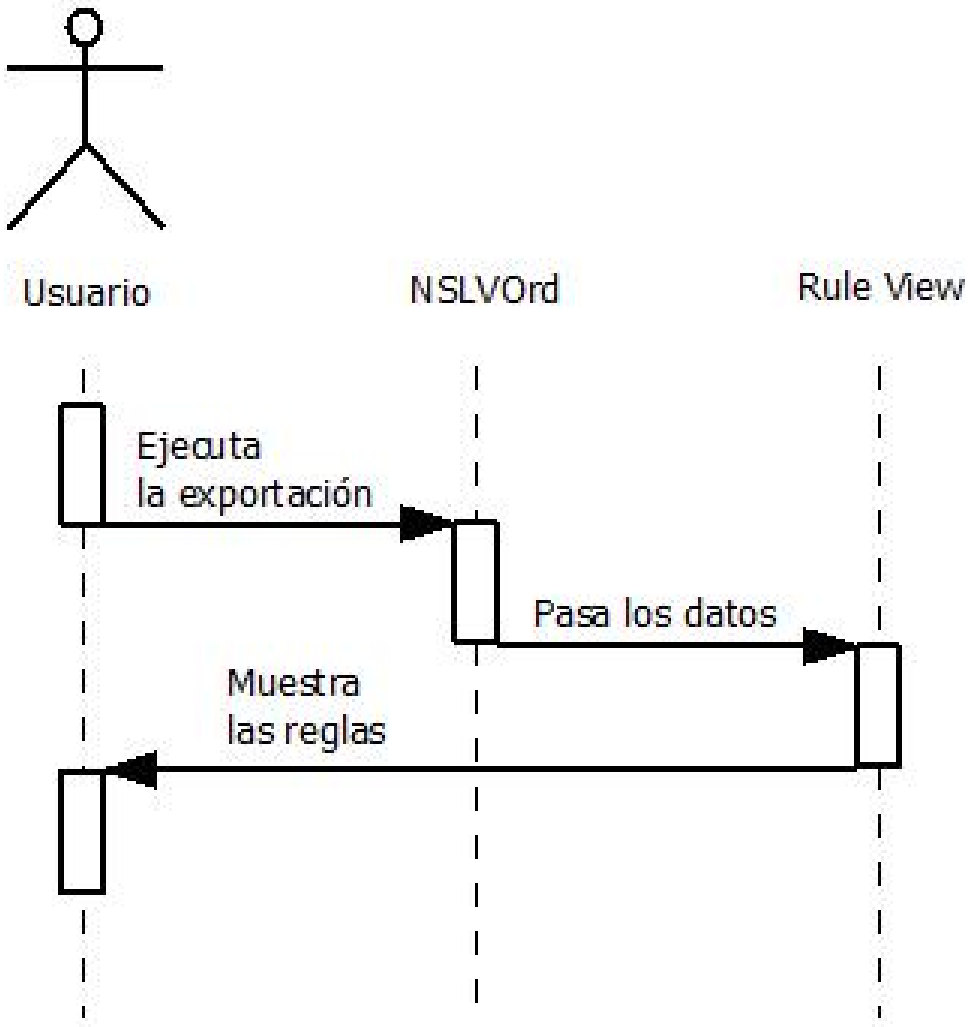


Figura 10.3: Diagrama de secuencia de la visualización de las reglas.

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 principales 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 Tabla11.1.

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 Tabla11.2.

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 Tabla11.3, 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

Clase Experiment

En la Tabla11.4 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 Tabla11.5, 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 Common

Al igual que la clase *Algorithms* (Tabla11.9), la clase *Common* (Tabla11.6) 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: Common
Clase abstracta que define los ajustes para los métodos de lectura de datos.
Hereda: -
Datos
categ, info, categ_att
Métodos
FormatFile, format, ReadFileFunction, ReadFile

Tabla 11.6: Clase Common

Clase matlab

La clase *matlab* (Tabla11.7) es la que se encarga de leer ficheros de datos soportados por Matlab. 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: Common
Datos
Métodos
format, ReadFile

Tabla 11.7: Clase matlab

Clase weka

La clase *weka* (Tabla11.8) 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: Common
Datos
attrs
Métodos
format, ReadFile, ReadWekaFile, ReadHeader, NewAttribute, ReadDatas, ToOneHot, ToNumeric, Categoric_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 Algorithms

La clase *Algorithms* (Tabla11.9) 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: Algorithms
Clase abstracta que define los ajustes para los algoritmos.
Hereda: handle
Datos
model, categ
Métodos
runAlgorithm, fit, predict, privfit, privpredict, parseArgs, setParam, getModel, setModel, getParameterNames

Tabla 11.9: Clase Algorithms

Clase NSLVOrd

En la Tabla11.10 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. NSLVOrd

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 expondrá la clase más relevante.

Clase NSLVOrdJava

La clase NSLVOrd, reflejada en la Tabla11.11, es la clase principal de este módulo y la que ha sido adaptada, tanto modificando métodos como añadiendo otros nuevos.

Clase: NSLVOrdJava
Clase principal de la ejecución del algoritmo NSLVOrd.
Hereda: -
Datos
E_par, E_par_test, R, alpha, clasificacion, configFile, costMatrix, tSet, errorAcceptable, fileCostMatrix, fileResultDebug, fileResultTest, fileTest, fileResultTrain, fileTrain, fileValid, fuzzyProblem, homogeneousLabel, iSet, iter, learning, numDesplazamientos, numLabelsInputs, numLabelsOutput, parametersKeel, percentErrorAcceptable, poblationParam, randomNum, seed, shift, time, dupLabels
Métodos
Train, GetFuzzyProblem, get_knowledge_base, LoadModel, ReadSetTFG, Load_KnowledgeBase, Load_RuleBase, export_rules, Test, main, initParameters, initParametersTFG, executeNSLVOrd, writeSyntax, executeNSLVOrdPredict, executeNSLVOrdPredictTFG, executeNSLVOrdTFG, checkSyntax, getParametersKeel, getCostMatrix, executeLearning, executeLearningTFG, executePredict, executePredictTFG, executeInference, loadLearning, saveLearning

Tabla 11.11: Clase NSLVOrdJava

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 (Tabla11.12) sirve como intermediaria para que en los algoritmos se pueda configurar las reglas para ser mostradas.

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 11.12: Clase RulesVisual

11.5.2. Java

Clase VisualRules

La clase *VisualRules* (Tabla11.13) es la encargada de crear la ventana en la que se representa las reglas y permite cambiar el número de antecedentes por linea 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 11.13: Clase VisualRules

Clase Rule

La clase *Rule* (Tabla11.14) 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 11.14: Clase Rule

Clase ConditionCategoric

Esta clase (Tabla11.15) 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 11.15: Clase ConditionCategoric

Clase ConditionFuzzyLogic

Esta clase (Tabla11.16) 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 11.16: Clase ConditionFuzzyLogic

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 (Tabla11.17) sirve como intermediaria para que en los algoritmos se pueda configurar las reglas para ser exportadas.

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 11.17: 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 Tabla11.19, 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 Tabla11.18 la clase *FuzzySystemType* ya que es de la que hereda la clase *FuzzyInferenceSystem* y donde se encuentran los métodos que utiliza esta.

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

Tabla 11.18: Clase FuzzySystemType

Clase: FuzzyInferenceSystem
Clase que representa al sistema que contiene las reglas.
Hereda: FuzzySystemType
Datos
Métodos

Tabla 11.19: Clase FuzzyInferenceSystem

Clase KnowledgeBaseType

La clase *KnowledgeBaseType* (Tabla11.20) 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 11.20: Clase KnowledgeBaseType

Clase MamdaniRuleBaseType

La clase *MamdaniRuleBaseType* (Tabla11.22) 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 Tabla11.21 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 11.21: Clase RuleBaseType

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

Tabla 11.22: Clase MamdaniRuleBaseType

Export

En la Tabla11.23 y Tabla11.24 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 11.23: 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 11.24: 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 Figura12.1 se muestra un esquema de la interfaz que servirá para explicar las diferentes secciones de esta, además, en la Figura12.2 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 linea.

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.

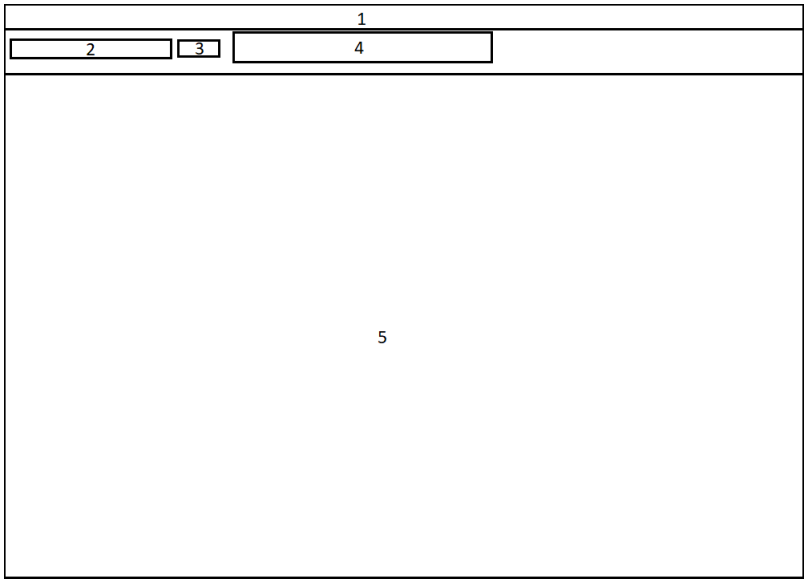


Figura 12.1: Esquema de la interfaz gráfica.

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 este. Por ello, en este capítulo se pretende mostrar que 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 de las salidas esperadas.

13.1. Datos para las pruebas

Datasets

En la Tabla13.1 se encuentran los *datasets* que se usarán para las pruebas. Los *datasets* y la tabla están sacados del TFM de Juan Carlos Gámez Granados [5, 1], ya que se pretende hacer la comparación con los resultados de este. 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 13.1: Datasets de prueba.

sistema de reglas

A continuación, se mostrará un sistema de reglas sencilla con las que se harán las pruebas en las que se necesite uno de estas. En este, se intentará mostrar 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 sido para adaptarlo a la nueva forma de leer los datos, por lo que no es necesario hacerles muchas pruebas. Sin embargo, se han hecho pequeñas pruebas de caja blanca con las que se ha podido comprobar que estos cambios actúan de manera correcta.

13.3. Read File

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

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* funcionen de manera correcta con los diferentes parámetros que afectan a su ejecución.

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, sin embargo, 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 le han hecho no han afectado al funcionamiento normal del algoritmo y que la 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.

13.6. Rule View

Para este módulo, se ha 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 ha hecho pruebas de caja negra para el módulo completo en las que se creaban una serie de reglas específicas y se comprobaba que estas se mostrasen como se esperaba, tal y como se puede ver en la Figura13.1 con respecto al sistema de reglas de la sección 13.1.

Figura 13.1: 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 con las que se pretendía que con unas reglas específicas, los ficheros que se generasen las representaran de forma correcta. A continuación, se mostrará como sería el fichero en formato JFML que generaría el sistema de reglas de la sección 13.1:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fuzzySystem xmlns="http://www.ieee1855.org">
 <knowledgeBase>
  <fuzzyVariable name="A" scale="" domainleft="0.0"
                 domainright="3.0" type="input">
   <fuzzyTerm name="uno" complement="false">
    <trapezoidShape param1="0.0" param2="0.0" param3="0.0"
                                                 param4="0.0"/>
   </fuzzyTerm>
   <fuzzyTerm name="dos" complement="false">
    <trapezoidShape param1="1.0" param2="1.0" param3="1.0"
                                                 param4="1.0"/>
   </fuzzyTerm>
   <fuzzyTerm name="tres" complement="false">
    <trapezoidShape param1="2.0" param2="2.0" param3="2.0"
                                                 param4="2.0"/>
   </fuzzyTerm>
   <fuzzyTerm name="cuatro" complement="false">
    <trapezoidShape param1="3.0" param2="3.0" param3="3.0"
                                                 param4="3.0"/>
   </fuzzyTerm>
  </fuzzyVariable>
  <fuzzyVariable name="B" scale="" domainleft="0.0"
                 domainright="3.0" type="input">
   <fuzzyTerm name="S2" complement="false">
    <trapezoidShape param1="0.0" param2="0.5" param3="0.5"
                                                 param4="1.0"/>
   </fuzzyTerm>
   <fuzzyTerm name="S1" complement="false">
    <trapezoidShape param1="0.5" param2="1.0" param3="1.0"
                                                 param4="1.5"/>
```

```
    </fuzzyTerm>
    <fuzzyTerm name="CE" complement="false">
     <trapezoidShape param1="1.0" param2="1.5" param3="1.5"
                                                  param4="2.0"/>
    </fuzzyTerm>
    <fuzzyTerm name="B1" complement="false">
     <trapezoidShape param1="1.5" param2="2.0" param3="2.0"
                                                  param4="2.5"/>
    </fuzzyTerm>
    <fuzzyTerm name="B2" complement="false">
     <trapezoidShape param1="2.0" param2="2.5" param3="2.5"
                                                  param4="3.0"/>
    </fuzzyTerm>
   </fuzzyVariable>
   <fuzzyVariable name="C" scale="" domainleft="0.0"
                        domainright="2.0" type="output">
    <fuzzyTerm name="a" complement="false">
     <trapezoidShape param1="0.0" param2="0.0" param3="0.0"
                                                  param4="0.0"/>
    </fuzzyTerm>
    <fuzzyTerm name="b" complement="false">
     <trapezoidShape param1="1.0" param2="1.0" param3="1.0"
                                                  param4="1.0"/>
    </fuzzyTerm>
    <fuzzyTerm name="c" complement="false">
     <trapezoidShape param1="2.0" param2="2.0" param3="2.0"
                                                  param4="2.0"/>
    </fuzzyTerm>
   </fuzzyVariable>
  </knowledgeBase>
  <mamdaniRuleBase name="" activationMethod="MIN" andMethod="MIN"
                                                  orMethod="MAX">
   <rule name="R1" andMethod="MIN" connector="and"
                                        weight="0.5">
    <antecedent>
     <clause>
      <variable>A</variable>
      <term>uno</term>
     </clause>
    </antecedent>
    <consequent>
     <then>
```

```
      <clause>
75     <variable>C</variable>
       <term>c</term>
      </clause>
     </then>
    </consequent>
80  </rule>
    <rule name="R2_1" andMethod="MIN" connector="and"
                                      weight="0.837">
     <antecedent>
      <clause>
85     <variable>A</variable>
       <term>dos</term>
      </clause>
      <clause>
       <variable>B</variable>
90     <term>CE</term>
      </clause>
     </antecedent>
     <consequent>
      <then>
95     <clause>
        <variable>C</variable>
        <term>a</term>
       </clause>
      </then>
100  </consequent>
    </rule>
    <rule name="R2_2" andMethod="MIN" connector="and"
                                      weight="0.837">
     <antecedent>
105   <clause>
       <variable>A</variable>
       <term>dos</term>
      </clause>
      <clause>
110    <variable>B</variable>
       <term>B1</term>
      </clause>
     </antecedent>
     <consequent>
115   <then>
```

```
        <clause>
         <variable>C</variable>
         <term>a</term>
        </clause>
120    </then>
      </consequent>
     </rule>
     <rule name="R3_1" andMethod="MIN" connector="and"
                                       weight="0.234">
125   <antecedent>
       <clause>
        <variable>B</variable>
        <term>S2</term>
       </clause>
130   </antecedent>
      <consequent>
       <then>
        <clause>
         <variable>C</variable>
135      <term>c</term>
        </clause>
       </then>
      </consequent>
     </rule>
140  <rule name="R3_2" andMethod="MIN" connector="and"
                                       weight="0.234">
      <antecedent>
       <clause>
        <variable>B</variable>
145     <term>B2</term>
       </clause>
      </antecedent>
      <consequent>
       <then>
150     <clause>
         <variable>C</variable>
         <term>c</term>
        </clause>
       </then>
155   </consequent>
     </rule>
     <rule name="R4" andMethod="MIN" connector="and"
```

```
                                        weight="0.9459">
      <antecedent>
160     <clause>
          <variable>B</variable>
          <term>S2</term>
        </clause>
        <clause>
165       <variable>A</variable>
          <term>cuatro</term>
        </clause>
      </antecedent>
      <consequent>
170     <then>
          <clause>
            <variable>C</variable>
            <term>b</term>
          </clause>
175     </then>
      </consequent>
    </rule>
  </mamdaniRuleBase>
</fuzzySystem>
```

13.8. Sistema

Una vez hechas 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, se ha visto que salen las correctas al compararlas con las que se sacaron en la prueba de la sección 13.5.

14 Conclusiones y Futuras Mejoras

Para acabar este proyecto, en este capítulo se hablará sobre las conclusiones que se han sacado 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

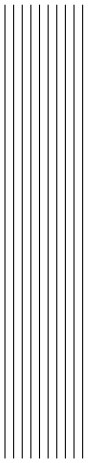
Durante el desarrollo de este trabajo se han obtenido diferentes conceptos, métodos y habilidades que se podrán aplicar en futuros trabajos:

- 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 un poco más.
- 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, accessed: 2020-06-06

[2] Grupo de Invertigación en Aprendizaje y Redes Neuronales Artificiales. Universidad de Córdoba. http://www.uco.es/grupos/ayrna, accessed: 2020-06-06

[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] 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

[7] 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)

[8] 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

[9] Soto-Hidalgo, J.M., Alonso, J.M., Acampora, G., Alcala-Fdez, J.: Jfml: A java library to design fuzzy logic systems according to the ieee std 1855-2016. IEEE Access 6(1), 54952–54964 (December 2018)