



UNIVERSIDAD
DE CÓRDOBA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

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

Manual de Usuario

Autor

Federico García-Arévalo Calles

Directores

Pedro Antonio Gutiérrez Peña

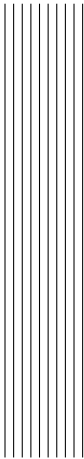
Juan Carlos Gámez Granados

ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba



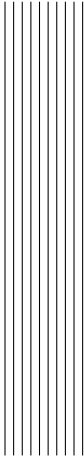
ESCUELA POLITÉCNICA SUPERIOR

—
Córdoba, mes de 2020



ÍNDICE GENERAL

Índice General	III
Listings	V
1. Introducción	1
1.1. Descripción del Producto	1
1.2. Instalación y Desinstalación	2
2. Ejecución de NSLVOrd	3
2.1. Fichero de Configuración	3
2.2. Ejecución del Experimento	5
3. Exportar las Reglas	7
3.1. Ejecución en NSLVOrd	11
4. Visualizar las Reglas	13
4.1. Ejecución en NSLVOrd	16



Listings

2.1. Configuración de un experimento	4
3.1. Ejemplo de código para exportar un sistema de reglas	10
3.2. Exportar un sistema de reglas	11
4.1. Ejemplo de visualizar las reglas	15
4.2. Visualizar las reglas	16



1 Introducción

Este documento corresponde al manual de usuario del Trabajo Fin de Grado “*Implementación de una interfaz para el algoritmo NSLVOrd en la biblioteca ORCA*”, en el que se explicará como se debe usar el producto de este trabajo.

1.1. Descripción del Producto

La finalidad de este producto es la incorporación del algoritmo NSLVOrd en ORCA. Pero además, se han añadido nuevas características a ORCA, que son:


- **Añadir datos de entrada categóricas:** con los que se usan archivos de formato Weka.
- **Visualizar las reglas:** se ha añadido la opción para hacer que los algoritmos incluyan una forma de visualizar las reglas que generan.
- **Exportar las reglas:** se ha añadido la opción para hacer que los algoritmos incluyan una forma de exportar las reglas que generan.

1.2. Instalación y Desinstalación

Para la instalación de este producto, se debe descomprimir el archivo “orcaTFG.zip” que viene junto a los documentos de este proyecto. Con esto, sería suficiente para tener ORCA en el sistema, sin embargo, para ciertos algoritmos que contiene hay que seguir unos pasos para poder compilar lenguaje C, pero eso no se incluirá en este manual ya que solo se tratará lo relacionado con el producto de este trabajo.

Tanto el algoritmo NSLVOrd, el visualizador de reglas y JFML, con lo que se exportarán las reglas, son archivos de Java, por lo que para poder usarlos se debe tener instalado Java en el sistema antes de su uso.

Por otro lado, para su desinstalación solo se deberá borrar la carpeta que se descomprimió.



2 Ejecución de NSLVOrd

Para una ejecución del algoritmo NSLVOrd, se realizará un experimento con ORCA, por lo que se usará Matlab u Octave.

2.1. Fichero de Configuración

Antes de realizar el experimento, se creará un fichero que contendrá la configuración de este. Como se puede ver en el fichero 2.1, la configuración del experimento se puede dividir en diferentes apartados:

1. Primero, se pondrá el nombre que recibirá el experimento (línea 2).
2. A continuación, se configura los datos con los que se realizará el experimento (líneas 4-10).
3. Después, el algoritmo y los parámetros que se usarán en el experimento (líneas 13 y 14).
4. Por último, los parámetros ha optimizar, separando con comas los diferentes valores de estos (líneas 17-37).

Como se puede ver en la sección 2 del fichero 2.1, básicamente se está especificando la ruta de los *datasets* en los parámetros *basedir*, *datasets* y *archive*. En este caso, los *datasets* se encontrarían en la ruta:
`../exampledata/1-holdout/toy/weka.`

```
1 ; Experiment ID
2 [nslvord-prueba]
3
4 {general-conf}
5   ; Datasets path
6   basedir = ../exampledata/1-holdout
7   ; Datasets to process (comma separated list)
8   datasets = toy
9   archive = weka
10  standarize = false
11
12 ; Method: algorithm and parameter
13 {algorithm-parameters}
14   algorithm = NSLVOOrd
15
16 ; Method's hyper-parameter values to optimize
17 {algorithm-hyper-parameters-to-cv}
18   Seed = 1286082570
19   LabelsInputs = 5
20   LabelsOutputs = 5
21   Shift = 35
22   Alpha = 0.5
23   Population = -1
24   MaxIteration = 500
25   IniProbBin = 0.9
26   CrosProbBin = 0.25
27   MutProbBin = 0.5
28   MutProbEachBin = 0.17
29   IniProbInt = 0.5
30   CrosProbInt = 0.0
31   MutProbInt = 0.5
32   MutProbEachInt = 0.01
33   IniProbReal = 0.0
34   CrosProbReal = 0.25
35   MutProbReal = 0.5
36   MutProbEachReal = 0.14
37   SeeRules = 1
```

Listing 2.1: Configuración de un experimento

2.2. Ejecución del Experimento

Una vez creado el fichero de configuración del experimento, este ya puede ser ejecutado con la función *Utilities.runExperiments*. Para ello, en Matlab u Octave se debe encontrar en la ruta *../orcaTFG/src*. Como ejemplo, se ejecutaría el fichero 2.1 con el siguiente comando:

```
Utilities.runExperiments('config-files/nslvord.ini').
```

Tras la ejecución del experimento, los resultados de este se encontrarán en la carpeta *../orcaTFG/src/Experiments*.



3 Exportar las Reglas

Como funcionalidad añadida a ORCA, en este producto, los algoritmos pueden añadir la capacidad de exportar a ficheros xml las reglas que se han obtenido con el entrenamiento. Para ello, cada algoritmo debe añadir en su clase de ORCA una sección de código (recomendable en una función) que se encargue de usar las funciones de la clase *RulesExport* para exportar las reglas. A continuación, se mostrará los pasos a seguir para realizar la exportación:

1. **Crea un objeto de la clase *RulesExport*:** como ejemplo, esto se hace con la línea de código “*export = RulesExport*”.
2. **Crea la base de conocimiento:** guarda todas las variables del sistema y los valores que puede tomar. Para ello, se siguen dos pasos por cada variable del sistema:
 - a) **Crea la variable:** se crea una variable haciendo uso de la función *new_variable(name, domain_left, domain_right)*, donde *name* es el nombre de la variable, *domain_left* es el valor numérico mínimo que puede tomar la variable y *domain_right* es el valor máximo. Para variables categóricas, *domain_left* y *domain_right* tomarán el valor numérico mínimo y máximo con el que se representen sus valores. La última variable creada, será la de salida.

- 8

Para tener una mejor idea de cómo sería exportar las reglas con código, se usará de ejemplo la función creada en el código 3.1. Para este ejemplo, se usarán de parámetros:

- ***dir***: dirección donde se guardarán los ficheros.
- ***name***: nombre del sistema de reglas.
- ***kb***: vector de estructuras que representan a las variables con los siguientes parámetros:
 - ***name***: nombre de la variable.
 - ***domain_left***: valor mínimo que toma la variable.
 - ***domain_right***: valor máximo que toma la variable.
 - ***terms***: vector de estructuras que representan a los valores de la variable con los siguientes parámetros:
 - ***name***: nombre del valor.
 - ***p1,p2,p3,p4***: son los valores numéricos que toman el valor.
- ***rb***: vector de estructuras que representan a las reglas con los siguientes parámetros:
 - ***name***: nombre de la regla.
 - ***weight***: la fiabilidad de la regla según los datos de entrenamiento.
 - ***antecedents***: vector de estructuras que representan a los antecedentes de una regla con los siguientes parámetros:
 - ***variable***: nombre de la variable.
 - ***term***: nombre del valor que toma la variable.
 - ***consequent***: estructura que representa al consecuente con los siguientes parámetros:
 - ***variable***: nombre de la variable consecuente.
 - ***term***: nombre del valor que toma la variable consecuente.

```
1 function exportfunction(dir,name,kb,rb)
    %Crea el objeto de la clase RulesExport
    export = RulesExport;

5    %Crea la base de conocimiento
    for i = 1:length(kb)
        %Crea la variable
        var = kb(i);
        export.new_variable(var.name,var.domain_left,...
10            var.domain_right);
        %Crea los valores de la variable
        terms = var.terms;
        for j = 1:length(terms)
            term = terms(j);
15            export.add_terms(term.name,term.p1,term.p2,term.p3,...
                term.p4);

        end
    end

20    %Crea la base de reglas
    for i = 1:length(rb)
        %Crea la regla
        rul = rb(i);
        export.new_rule(rul.name,rul.weight);

25        %Añade los antecedentes
        ants = rul.antecedents;
        for j = 1:length(ants)
            ant = ants(j);
30            export.add_antecedent(ant.variable,ant.term);
        end

        %Añade el consecuente
        con = rul.consequent;
35        export.new_consequent(con.variable,con.term);
    end

    export.export_rules(dir,name);
end
```

Listing 3.1: Ejemplo de código para exportar un sistema de reglas

3.1. Ejecución en NSLVOOrd

Las reglas no pueden ser exportadas durante la ejecución de un experimento de NSLVOOrd, por lo que hay que hacerlo una vez acabe. En el código 4.2 se puede ver un ejemplo de como exportar las reglas con el modelo ya aprendido siguiendo los siguientes pasos:

1. **Crear un objeto de la clase *NSLVOOrd*:** como ejemplo, esto se hace con la línea de código “*algorithm = NSLVOOrd*”.
2. **Cargar el modelo:** para esto, primero se debe guardar el modelo (*model.mat*) en una variable (*file*) con la función *load* (ej: *file = load('model.mat')*). Una vez hecho, la variable será una estructura con un único dato “*model*” que es el modelo. A continuación, se guarda dicho dato en la variable “*model*” del objeto creado en el paso 1 (ej: *algorithm.model = file.model*)
3. **Exportar las reglas:** con el modelo ya cargado, solo es necesario llamar a la función *export_rules(dir)*, siendo *dir* el directorio donde se desea guardar los ficheros.

```
% Crear instancia del objeto NSLVOOrd
algorithm = NSLVOOrd;

% Cargar el modelo
file = load('model.mat');
algorithm.model = file.model;

% Exportar las reglas
algorithm.export_rules('directorio');
```

Listing 3.2: Exportar un sistema de reglas



4 Visualizar las Reglas

Como funcionalidad añadida a ORCA, en este producto, los algoritmos pueden añadir la capacidad de visualizar en una ventana las reglas que se han obtenido con el entrenamiento. Para ello, cada algoritmo debe añadir en su clase de ORCA una sección de código (recomendable en una función) que se encargue de usar las funciones de la clase *RulesVisual* para visualizar las reglas. A continuación, se mostrará los pasos a seguir para realizar la visualización:

1. **Crea un objeto de la clase *RulesVisual*:** como ejemplo, esto se hace con la línea de código “*visual = RulesVisual*”.
2. **Añade las reglas:** añade todas las reglas del sistema. Para ello, se siguen tres pasos por cada regla del sistema:
 - a) **Crea la regla:** se crea una regla haciendo uso de la función *new_rule(name,weight)*, donde *name* es el nombre de la regla y *weight* es la fiabilidad de la regla según los datos de entrenamiento.
 - b) **Añade los antecedentes:** una vez creada una regla, los antecedentes se añaden con la función *add_antecedent(variable,term)*. *variable* es el nombre de la variable y *term* es una matriz en la que cada fila representa cada valor que toma la variable en la regla y debe contener los valores [*name_val,p1,p2,p3,p4,InfL,InfR*]. Hay que tener en cuenta que cada vez que se realice el paso 2a los nuevos antecedentes se añadirá a la última regla creada.

- ***name_val***: es el nombre que recibe el valor.
 - ***p1,p2,p3,p4***: representa los valores numéricos del valor, en caso de ser categórico, todos deberán valer lo mismo y en el caso de que se aplique lógica difusa, los valores numéricos que la representen.
 - ***InfL,InfR***: en valores categóricos no importan el valor que tomen, pero en el caso de que se aplique lógica difusa, representa si la variable tiende a infinito por la izquierda y por la derecha respectivamente. 0 significa que no tiende a infinito y 1 que sí tiende a infinito.
- c) **Añade el consecuente**: una vez creada una regla, el consecuente se añade con la función *new_consequent(variable,term)*. *variable* es el nombre de la variable de salida y *term* el nombre del valor que toma la variable para este consecuente. Hay que tener en cuenta que cada vez que se realice el paso 2a el consecuente se añadirá a la última regla creada y que solo puede haber un consecuente, por lo que el anterior se reescribirá.

3. **Visualizar las reglas**: esto se hace con la función *visual_rules(name)*, siendo *name*, el nombre que se le dará a la ventana que se genere.

Para tener una mejor idea de cómo sería visualizar las reglas con código, se usará de ejemplo la función creada en el código 4.1. Para este ejemplo, se usarán de parámetros:

- ***name***: nombre del sistema de reglas.
- ***rules***: vector de estructuras que representan a las reglas con los siguientes parámetros:
 - ***name***: nombre de la regla.
 - ***weight***: la fiabilidad de la regla según los datos de entrenamiento.
 - ***antecedents***: vector de estructuras que representan a los antecedentes de una regla con los siguientes parámetros:
 - ***variable***: nombre de la variable.
 - ***term***: matriz como la que se explica en el paso 2b.

-
- **consequent**: estructura que representa al consecuente con los siguientes parámetros:
 - **variable**: nombre de la variable consecuente.
 - **term**: nombre del valor que toma la variable consecuente.

```
1 function visualfunction(name, rules)
    % Crea el objeto de la clase RulesExport
    visual = RulesVisual;

5    % Añade las reglas
    for i = 1:length(rb)
        % Crea la regla
        rul = rules(i);
        visual.new_rule(rul.name, rul.weight);

10
        % Añade los antecedentes
        ants = rul.antecedents;
        for j = 1:length(ants)
            ant = ants(j);
15            visual.add_antecedent(ant.variable, ant.term);
        end

        % Añade el consecuente
        con = rul.consequent;
20        visual.new_consequent(con.variable, con.term);
    end

    visual.visual_rules(name);
end
```

Listing 4.1: Ejemplo de visualizar las reglas

4.1. Ejecución en NSLVOOrd

Las reglas pueden ser visualizadas durante la ejecución de un experimento de NSLVOOrd si el parametro *SeeRules* del fichero de configuración 2.1 esta a 1, pero también puede hacerse una vez esté acabado. En el código 4.2 se puede ver un ejemplo de como exportar las reglas con el modelo ya aprendido siguiendo los siguientes pasos:

1. **Crear un objeto de la clase *NSLVOOrd*:** como ejemplo, esto se hace con la línea de código “*algorithm = NSLVOOrd*”.
2. **Cargar el modelo:** para esto, primero se debe guardar el modelo (*model.mat*) en una variable (*file*) con la función *load* (ej: *file = load('model.mat')*). Una vez hecho, la variable será una estructura con un único dato “*model*” que es el modelo. A continuación, se guarda dicho dato en la variable “*model*” del objeto creado en el paso 1 (ej: *algorithm.model = file.model*)
3. **Exportar las reglas:** con el modelo ya cargado, solo es necesario llamar a la función *visual_rules()*.

```
% Crear instancia del objeto NSLVOOrd
algorithm = NSLVOOrd;

% Cargar el modelo
file = load('model.mat');
algorithm.model = file.model;

% Visualizar las reglas
algorithm.visual_rules();
```

Listing 4.2: Visualizar las reglas