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

—

Córdoba, Junio de 2020

Índice General

Índice de Código

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 cómo 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. Además, se han añadido nuevas características a ORCA, que son:

- **Añadir datos de entrada categóricos:** por lo que se incorpora la capacidad de leer archivos de formato Weka.
- **Visualizar las reglas:** se ha añadido la opción para hacer que los algoritmos basados en reglas incluyan una forma de visualizar las reglas que generan.
- **Exportar las reglas:** se ha añadido la opción para hacer que los algoritmos basados en reglas 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 o también puede localizarse en el repositorio `https://github.com/federicogac/TFG`. Una vez hecho, se crea una carpeta llamada "orca" y con esto, ya sería suficiente para tener ORCA en el sistema.

Teniendo ORCA ya guardado en el sistema, aún es necesario seguir unos pasos para su completo funcionamiento:

- **Instalar Java 8:** JFML crea los fichero xml haciendo uso de JAXP (Java API for XML Processing) el cual solo se encuentra hasta la versión 8 de Java. Sin embargo, también es posible usar versiones más recientes, pero a la hora de exportar, solo se podrá en PMML y no JFML. En caso de usar Openjdk, se aplica la misma situación.

- **Instalar Matlab u Octave:** para ejecutar las funciones de ORCA es necesario tener instalado Matlab u Octave en el sistema.

  En el caso de Matlab se puede descargar el programa y comprar la licencia en su página "`https://www.mathworks.com`". Al ejecutar el instalador, este ya se ocupa de ayudar al usuario paso a paso.

  Por otro lado, en Windows se puede descargar la versión más reciente (5.2.0) desde su página "`https://www.gnu.org/software/octave`" y en Linux, la versión 4.2.2 con el comando: ***sudo apt install octave***

- **Compilar los algoritmos en C:** para algunos algoritmos de ORCA 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. Pero para los interesados, pueden ver el tutorial de instalación en su repositorio de github "`https://github.com/ayrna/orca#installation-tutorials-and-documentation`" o usando Jupiter con el fichero "README.md" de la carpeta "orca" que se ha descomprimido anteriormente.

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

2 Ejecución de NSLVOrd

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

2.1. Fichero de Configuración

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

1. **ID del experimento:** en la linea 2 se ha especificado la ID del experimento (*[nslvord-prueba]*) que sirve para diferenciarlo de otros experimentos. En un mismo fichero pueden añadirse más de un experimento y cada uno debe tener sus propios apartados 2, 3 y 4.

2. **Configuración general**: desde la linea 4 a la 18, se realiza la configuración general del experimento con los parámetros:

   - **basedir:** indica el directorio donde se encuentra los conjuntos de entrenamiento.
   - **datasets:** indica el nombre de la carpeta con los conjuntos de entrenamiento que se usará en el experimento. Se puede añadir varios separando los nombres con comas.

- **archive:** indica el nombre de la carpeta con los ficheros de los *datasets*. De momento solo se permite el valor *matlab* y *weka*, por defecto será *matlab*. Usando los valores de *basedir*, *datasets* y *archive* del ejemplo, el directorio que contiene los archivos para leer sería *../orca/exampledata/1-holdout/toy/weka*
- **standarize:** variable que indica si se activa o no la estandarización de los datos. Por defecto es *true*.
- **num_folds:** indica el número de grupos de datos en el que se dividirá un *dataset* para hacer el proceso interno de *cross-validation*. Por defecto es 5.
- **cvmetric:** indica la métrica que servirá para juzgar que conjunto de parámetros es el más óptimo en el proceso interno de *cross-validation*. Por defecto es *MAE* (*Mean Absolute Error*).
- **seed:** semilla que se usa para la aleatoriedad al dividir los *datasets* en el proceso interno de *cross-validation*. Por defecto es 1.
- **report_sum:** indica si se quiere guardar la matriz de confusión en los resultados. Por defecto es *false*.

3. **Algoritmo del experimento:** en las líneas 21 y 22 se especifica el algoritmo que se ejecutará en el experimento.

4. **Parámetros del algoritmo:** desde la línea 25 a la 45 se especifican los parámetros que se usarán en el algoritmo. Se puede poner más de un valor en un parámetro separando éstos con comas y en ese caso se realizará el *cross-validation*. Los parámetros de NSLVOrd son:

   - **Seed:** semilla que se usará para la aleatoriedad del algoritmo.
   - **LabelsInputs:** el número de etiquetas que se crearán para una variable a la que se le haya aplicado lógica difusa.
   - **LabelsOutputs:** el número de etiquetas que se crearán si a la salida se le aplica lógica difusa.
   - **Shift:** tamaño de la etiqueta para el desplazamiento.
   - **Alpha:** parámetro de ponderación de la característica ordinal o nominal para el *fitness*.

- **Population:** valor para la población inicial del algoritmo genético.
- **MaxIteration:** número máximo de iteraciones para obtener una nueva regla.
- **IniProbBin:** probabilidad inicial para la población binaria del genético.
- **CrosProbBin:** probabilidad de cruce para la población binaria del genético.
- **MutProbBin:** probabilidad de mutación para la población binaria del genético.
- **MutProbEachBin:** probabilidad de mutación para cada elemento individual de la población binaria del genético.
- **IniProbInt:** probabilidad inicial para la población entera del genético.
- **CrosProbInt:** probabilidad de cruce para la población entera del genético.
- **MutProbInt:** probabilidad de mutación para la población entera del genético.
- **MutProbEachInt:** probabilidad de mutación para cada elemento individual de la población entera del genético.
- **IniProbReal:** probabilidad inicial para la población real del genético.
- **CrosProbReal:** probabilidad de cruce para la población real del genético.
- **MutProbReal:** probabilidad de mutación para la población real del genético.
- **MutProbEachReal:** probabilidad de mutación para cada elemento individual de la población real del genético.
- **SeeRules:** activa la opción de visualizar las reglas.
- **ExportRules:** activa la opción de exportar las reglas.

```
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   ; Datasets type
10  archive = weka
11  ; Standarize
12  standarize = false
13  ; Cross-validate param
14  num_folds = 5
15  cvmetric = MAE
16  seed = 1
17  ; Confusion matrix
18  report_sum = false
19
20 ; Method: algorithm and parameter
21 {algorithm-parameters}
22  algorithm = NSLVOrd
23
24 ; Method's hyper-parameter values to optimize
25 {algorithm-hyper-parameters-to-cv}
26  Seed = 1286082570
27  LabelsInputs = 5
28  LabelsOutputs = 5
29  Shift = 35
30  Alpha = 0.5
31  Population = -1
32  MaxIteration = 500
33  IniProbBin = 0.9
34  CrosProbBin = 0.25
35  MutProbBin = 0.5
36  MutProbEachBin = 0.17
37  IniProbInt = 0.5
38  CrosProbInt = 0.0
39  MutProbInt = 0.5
40  MutProbEachInt = 0.01
41  IniProbReal = 0.0
```

```
42   CrosProbReal = 0.25
43   MutProbReal = 0.5
44   MutProbEachReal = 0.14
45   SeeRules = 1
46   ExportRules = 1
```

Código 2.1: Configuración de un experimento

2.2. Ejecución del Experimento

Una vez creado el fichero de configuración del experimento, éste ya puede ser ejecutado con la función *Utilities.runExperiments*. Para ello, en Matlab u Octave se debe encontrar en la ruta *../orca/src*. Como ejemplo, al ejecutar el Código 2.1 con la función, se produciría la salida incluida en el Código 2.2

```
1 >> Utilities.runExperiments('config-files/nslvord.ini')
2 Setting up experiments...
3 Running experiment exp-nslvord-prueba-toy-1.ini
4 Calculating results...
5 Experiments/exp-2020-6-28-20-37-28/Results/toy-nslvord-prueba/dataset
6 Experiments/exp-2020-6-28-20-37-28/Results/toy-nslvord-prueba/dataset
7
8 ans =
9
10   'Experiments/exp-2020-6-28-20-37-28'
```

Código 2.2: Prueba de ejecución del fichero *nslvord.ini*

A continuación, se explicará el resultado de la ejecución que se muestra en el Código 2.2:

1. La primera linea (*Utilities.runExperiments('config-files/nslvord.ini')*) es la función que se usa para ejecutar el experimento. El resto de líneas ya son las salidas que muestra la ejecución de éste.

2. Mientras en la salida solo se muestre *Setting up experiments...*, ORCA esta configurando el experimento. Durante este tiempo: ORCA comprueba que exista la carpeta proporcionada por el usuario con los *datasets* de entrenamiento y prueba, habiendo uno de prueba por cada uno de entrenamiento; y crea las carpetas donde se guardarán los resultados y el fichero de configuración del experimento.

3. Durante el proceso del *Running experiment exp-nslvord-prueba-toy-1.ini* ya se está realizando la ejecución del algoritmo, tanto el entrenamiento como el test. En este caso solo hay un *dataset*, pero en el caso de que hubiese más, la salida sería así:

```
3 Running experiment exp-nslvord-prueba-toy-1.ini
4 Running experiment exp-nslvord-prueba-toy-2.ini
5 Running experiment exp-nslvord-prueba-toy-3.ini
6 Running exp...
```

4. En las líneas 4-10, la ejecución calcula los resultados haciendo las métricas y guardando los resultados en la carpeta generada en el paso 2 *../orca/src/Experiments/exp-2020-6-28-20-37-28*. El contenido de la carpeta es el siguiente:

   - **exp-nslvord-prueba-toy-1.ini:** fichero con la configuración del experimento.
   - **Results:** carpeta con los resultados de los experimentos.
     - **mean-results_train.csv:** fichero con las medias de las métricas para el conjunto de entrenamiento.
     - **mean-results_test.csv:** fichero con las medias de las métricas para el conjunto de test.
     - **mean-results_matrices_sum_train.csv:** fichero con las sumas de las matrices de confusión del entrenamiento. Solo existe en caso de tener en el fichero de configuración *report_sum* a *true*.

- **mean-results_matrices_sum_test.csv:** fichero con las sumas de las matrices de confusión del test. Solo existe en caso de tener en el fichero de configuración *report_sum* a *true*.
- **toy-nslvord-prueba:** carpeta con los resultados del experimento. En el caso de más experimentos en una ejecución, habría una carpeta por cada uno con el nombre del identificador del experimento.
  - **results_train.csv:** fichero con las métricas del entrenamiento.
  - **results_test.csv:** fichero con las métricas del tests.
  - **matrices_train.txt:** fichero que contiene las matrices de confusión del entrenamiento. Solo existe en caso de tener en el fichero de configuración *report_sum* a *true*.
  - **matrices_test.txt:** fichero que contiene las matrices de confusión del test. Solo existe en caso de tener en el fichero de configuración *report_sum* a *true*.
  - **dataset:** fichero que guarda el directorio del *dataset*.
  - **Times:** carpeta que contiene ficheros con los tiempos de ejecución del algoritmo.
  - **Predictions:** carpeta que contiene ficheros con las salidas obtenidas (categorías predichas) una vez entrenado el modelo.
  - **OptHyperparams:** carpeta que contiene ficheros con los parámetros óptimos de entrenamiento tras el proceso interno de cross-validation.
  - **Models:** carpeta que contiene ficheros con los modelos obtenidos en el entrenamiento.
  - **Guess:** carpeta que contiene fichero con las salidas esperadas de los *datasets*.

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.

   b) **Crea los valores de la variable:** una vez creada la variable, con la función *add_terms(name,p1,p2,p3,p4)* se puede añadir los diferentes valores de esta. *name* representa el nombre que recibe el valor; *p1*, *p2*, *p3* y *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. Hay que tener en cuenta que cada vez que se realice el paso $2a$ cada nuevo valor se añadirá a la última variable creada.

3. **Crea la base de reglas:** guarda 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. En el este caso, JFML no usa "or" en las reglas, por lo que si las reglas lo poseen, se deben dividir en diferentes reglas.
   b) **Añade los antecedentes:** una vez creada la regla, se añaden los antecedentes con la función *add_antecedent(variable,term)*. *variable* es el nombre de la variable y *term* el nombre del valor que toma la variable para este antecedente. Hay que tener en cuenta que cada vez que se realice el paso $3a$ los nuevos antecedentes se añadirá a la última regla creada.
   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 $3a$ el consecuente se añadirá a la última regla creada y que solo puede haber un consecuente, por lo que el anterior se reescribirá.

4. **Exporta las reglas:** esto se hace con la función *export_rules(dir,name)*, siendo *dir* la ruta donde se guardarán los ficheros xml y *name*, el nombre del sistema de reglas.

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 con los que se 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 con los que se 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)
2   %Crea el objeto de la clase RulesExport
3   export = RulesExport;
4
5   %Crea la base de conocimiento
6   for i = 1:length(kb)
7     %Crea la variable
8     var = kb(i);
9     export.new_variable(var.name,var.domain_left,...
10                        var.domain_right);
11    %Crea los valores de la variable
12    terms = var.terms;
13    for j = 1:length(terms)
14        term = terms(j);
15        export.add_terms(term.name,term.p1,term.p2,term.p3,...
16                         term.p4);
17    end
18  end
19
20  %Crea la base de reglas
21  for i = 1:length(rb)
22    %Crea la regla
23    rul = rb(i);
24    export.new_rule(rul.name,rul.weight);
25
26    %Añade los antecedentes
27    ants = rul.antecedents;
28    for j = 1:length(ants)
29        ant = ants(j);
30        export.add_antecedent(ant.variable,ant.term);
31    end
32
33    %Añade el consecuente
34    con = rul.consequent;
35    export.new_consequent(con.variable,con.term);
36  end
37
38  export.export_rules(dir,name);
39 end
```

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

3.1. Exportar las reglas de NSLVOrd

Las reglas pueden ser exportadas durante la ejecución de un experimento de NSLVOrd si el parametro *ExportRules* del fichero de configuración 2.1 esta a 1 guardándose en la carpeta con el modelo aprendido, 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 *NSLVOrd*:** esto se hace en el directorio *../orca/src/Algorithms* y como ejemplo, esto se hace con la línea de código "*algorithm = NSLVOrd*".

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')*). El modelo se encuentra en la carpeta "*Experiments/exp-aaa-mm-dd-hh-mm-dd/Results/IDdelExperimento*".

   Una vez guardado el modelo, 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 NSLVOrd
algorithm = NSLVOrd;

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

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

CÓDIGO 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:** con la regla creada, 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 importa el valor, 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 2*a* 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 con las que se 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 2*b*.

- ***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)
2   %Crea el objeto de la clase RulesExport
3   visual = RulesVisual;
4
5   %Añade las reglas
6   for i = 1:length(rb)
7     %Crea la regla
8     rul = rules(i);
9     visual.new_rule(rul.name,rul.weight);
10
11    %Añade los antecedentes
12    ants = rul.antecedents;
13    for j = 1:length(ants)
14        ant = ants(j);
15        visual.add_antecedent(ant.variable,ant.term);
16    end
17
18    %Añade el consecuente
19    con = rul.consequent;
20    visual.new_consequent(con.variable,con.term);
21  end
22
23  visual.visual_rules(name);
24 end
```

Código 4.1: Ejemplo de visualizar las reglas

4.1. Visualizar las reglas de NSLVOrd

Las reglas pueden ser visualizadas durante la ejecución de un experimento de NSLVOrd 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 *NSLVOrd*:** esto se hace en el directorio *../orca/src/Algorithms* y como ejemplo, esto se hace con la línea de código "*algorithm = NSLVOrd*".

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')*). El modelo se encuentra en la carpeta "*Experiments/exp-aaa-mm-dd-hh-mm-dd/Results/IDdelExperimento*".

   Una vez guardado el modelo, 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 NSLVOrd
algorithm = NSLVOrd;

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

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

Código 4.2: Visualizar las reglas