Some learning approaches with FisPro

Fuzzy inference system (FIS) design from learning data follows two main steps:

- Fuzzy partitioning of input and output variables, section 1
- Rule induction, section 2.

Partitions as well as rules can then be adjusted, section 4. A FIS can also be simplified (section 5), to obtain incomplete rules (see glossary).

Learning programs are available as command lines and are also interfaced in Java. The execution time is likely to increase very fast with the data file size. This is particularly true for the optimization and simplification procedures. In that case, it is preferable to use the command line versions. All the command lines versions can run in the background with the ouput redirected to a file.

Since *FisPro* version 3.4, all learning programs, when called from command line, have a -wl option, that allows a silent (wordless) mode. Only error messages will appear on the output.

Contents

Alphabetical list of the C++ programs available as command lines:

The following programs: **genrules, fislinks, perf, readdata, sample** are utility programs. The **other ones** are *learning* programs.

The **fis** program is aimed for developers of new Fispro modules. it corresponds to the **testfis.cpp** source program, located in the cpp/base subdirectory of fispro. it gives example calls of Fispro C++ functions (refer to source comments for more details).

- **dist** : create a distance matrix
- **fis**: example of use of a FIS and a data file
- **fisimple**: simplify a FIS (see section 5)
- **fistree**: generate fuzzy decision trees and the corresponding FIS (see section 2)
- **fpa**: generate the FIS rule conclusions (see section 2)
- **genrules**: generate the FIS rule premise from known input partitions (see section 2)
- **hfpcfg**: create a HFP configuration file (see section 1)
- **hfpfis**: generate the FIS configuration file from the HFP configuration file (see section 1)
- **hfpselect**: select the number of membership functions (MFs) per variable (see section 1)
- **hfpsr**: generate a FIS configuration file without rules(see section 1)
- **hfpvertex**: compute the MF centers from a HFP configuration file and a data file (see section 1)
- **fislinks**: compute representative elements of links between a FIS rule premise and data from a file (see section **??**)
- **looptoptims**: optimize parts of a FIS with cross validation (see section 4)
- **ols**: generate rules using OLS (orthogonal least squares (see section 3)
- **perf**: compute the FIS performance relative to a data file (section 9)

- **readdata**: check the reading of a text field separated data file (see section 10)
- **sample**: generate learning and test samples from a data file(section 8).
- **sethfpfis**: edit a HFP configuration file
- **vocreduc**: reduce the FIS output vocabulary (see section 6)
- **wm**: generate FIS rules (premise and conclusion) (see section 2)

Note valid for all FisPro C++ programs:
A call without argument displays the program required and optional arguments.

We start with a brief description of the different steps to follow, then we illustrate the procedure on a data set. For each procedure, we give the command line arguments in C++, and the corresponding option in the (java) graphical user interface. User documentation gives more details about the interface.

The result files mentioned in command lines are stored in the EXAMPLES subdirectory.

Notations:

FIS=fuzzy inference system

MF=membership function

1. **Partitioning**

   Partitions induced by FisPro are standardized fuzzy partitions (see glossary), which guarantees that each fuzzy set corresponds to a linguistic label. There is one exception: the Gaussian MF fuzzy partitions generated by the OLS with the original algorithm from Hohensohn and Mendel

   Three partition types are available: hfp, k-means, regular.

   The easiest way is to use the 'Generate a FIS without rules' option, which allows to build input and output fuzzy partitions from data. A partition type, common to all inputs, must be specified, as well as an output partition type.

   The *Learning* menu has two options to generate the partitions: *Generate a FIS without rules* and *HFP MF*.

The easiest way is to use the *Generate a FIS without rules* option, which allows to build input and output fuzzy partitions from data. A partition type, common to all inputs, must be specified, as well as an output partition type. This option is also available in the *FIS* menu and in the *OLS* learning option.

The *HFP MF* option allows to choose the number of MFs per partition according to the data. The algorithm starts from the simplest configuration, complexifies it by adding a fuzzy set to the partition of one input variable, and searches for the best performing configuration among the most compact ones. In that case, the number of fuzzy sets per variable is not set a priori.

2. **Rule induction**

   Several methods are available in FisPro to generate fuzzy rules:

   - Generate all possible rules, then initialize rule conclusions using the *FPA* method
   - Use the Wang & Mendel (*wm*) algorithm
   - Build a fuzzy decision tree and prune it if necessary
   - Use the *HFP MF* algorithm
   - Use the *OLS* algorithm

   The first three methods all need both a FIS configuration file, used to define the fuzzy partitions, and a data file.

   The HFP MF option is another method, that only needs a data file and allows to generate the rules corresponding to the HFP MF option (Learning/Partition menu). It needs two specific files: a HFP configuration file and a vertex file, described in section 1. The algorithm starts from the simplest configuration, and complexifies it by adding each time a MF in one input dimension, and to search for the most accurate configuration among the most compact ones. The number of MFs in each input dimension is not known beforehand.

3. **Generate partitions and induce rules using OLS**

   The *OLS* method can work using only data, first generating the partitions, either standard fuzzy partitions made of triangular and trapezoidal MFs, or Gaussian MF partitions, then generating the rules. It can also use a FIS configuration file, then using the FIS partitions, and only generating the rules.

4. **Optimization**

   The optimization option of the *Learning* menu allows to adjust the fuzzy set parameters associated to the input or output variables, and/or to optimize the rule conclusions.

5. **Simplification**

   Last of all, the *simplification* option of the *Learning* menu attempts to simplify the FIS by grouping rules together, or by removing some variables from the rule premises.

6. **Reduce the output vocabulary**

   By output vocabulary, we mean the distinct fuzzy rule conclusions. Learning procedures often generate rules with as many conclusions as rules. For more interpretability, these conclusions can be slightly modified to reduce the number of distinct ones.

7. **Links**

   We try to quantify links between fuzzy rules and data, by calculating the rule cumulated matching degree for a data file, by giving the matched rules for each row, and conversely by enumerating the examples matching each rule.

8. **Sampling**

   The *Generate sample* option of the *Data* menu creates random learning/validation samples from a data file.

9. **Performance and data**

   This program, or option, allows to measure the FIS performance on a data set.

   Two test data sets are included.

   The iris data (Fisher, 1936) are made up of four input variables: sepal length, sepal width, petal length and petal width. The output is the iris species: Setosa, Virginica or Versicolor. The sample includes 50 items of each species.

   The rice data (Ishibuchi, 1994) are sensory test data collected from a subjective evaluation of many different kinds of rice by plural panelists. The sample includes 105 items. The following five factors constitute the input data: flavor, appearance, taste, stickiness, toughness, and the output factor is the overall evaluation of quality.

10. **Check data**

    The **readdata** program checks the data file reading: number of rows, number of columns, heading, field separator.

11. **Learning result files**

    Each learning procedure creates several result data files. Two of these files share a common format, and are detailed in section 11.

1 Partitioning

1. **hfpsr** generates a configuration file for a FIS. The number of fuzzy sets must be specified for each input, and for the output, as well as the hierarchy types to use for inputs and output. The created FIS has no rules.

   Java interface:

   Learning menu, *Partitions* submenu, *Generate a FIS without rules* option.

   Command line, hfpsr program:

   Arguments:

   - the data file name
   - the number of fuzzy sets for each input variable (string argument within which numbers are delimited by spaces)
   - the input hierarchy type: 1 for hfp, 2 for k-means, 3 for regular
   - if value<1, the tolerance value, used to group input data into unique values, or, if value>1, the number of groups for the k-means algorithm (only used in the hfp hierarchy).
   - the number of fuzzy sets for each output
   - the output hierarchy type: 1 for hfp, 2 for k-means, 3 for regular
   - the defuzzification operator: area, MeanMax or sugeno
   - the disjunction operator: sum or max
   - if value<1, the tolerance value, used to group output data into unique values, or if value>1, the number of groups for the k-means algorithm if value>1 (only used in the hfp hierarchy)

   Options:

   -oFIS' where FIS is the output configuration file name (default: 'data file name'-sr.fis)

-oConj where Conj is the disjunction operator(default: prod)

-vVertexFile where VertexFile is the vertex file (see hfpvertex, default: vertices are computed and a file is created).

-f: sets the Classif='yes' output option

Command line example:

hfpsr iris '3 3 3 3' 1 0.01 3 3 MeanMax sum 0.01

Creates the files iris.hfp, iris.vertex and iris-sr.fis with 3 fuzzy sets per input variable, the hfp hierarchy type, and a regular grid of 3 fuzzy sets for the output.

or

hfpsr iris '3 3 3 3' 2 0.01 3 3 MeanMax sum 0.01 -oiriskm.fis

Creates the files iris.hfp, iris.vertex and iriskm.fis, where the input partitions are generated by the k-means procedure.

hfpsr iris '2 2 3 3' 3 0 0 3 sugeno sum 0 -oirisreg.fis -f

Creates the files iris.hfp, iris.vertex and irisreg.fis, with 2 MF regular grids for the fist two inputs, 3 MF regular grids for the other two, and a crisp classification output.

2. **Select partitions**

   This method requires several programs.

   Java interface:

   (a) *Learning* menu, *Partitions* submenu, *HFP MF* submenu, *Generate a HFP file* and *Edit a HFP file* options

   (b) *Learning* menu, *Partitions* submenu *HFP MF* submenu, *Generate vertices* option.

   Vertices can be viewed with the *View vertices* option. The maximum number of vertices generated for each input is specified in the HFP (default value=7).

   (c) *Learning* menu, *Partitions* submenu *HFP MF* submenu, *Select partition* option.

   Command lines:

   (a) **hfpcfg** to create the HFP configuration file

   Arguments:

- data file name
- number of columns to ignore. Typically 1, which means to ignore the last column, representing the output.

Two optional parameters:

The ouput hfp file name (default value the data file name.hfp)

The hierarchy type:1 for hfp, 2 for k-means, 3 for a regular grid (default value)

Command line examples:

hfpcfg iris 1

Creates the file iris.hfp with a regular grid hierarchy

or

hfpcfg iris 1 iriskm.hfp 2

Creates the file iriskm.hfp with a k-means hierarchy.

The default generated output is crisp, with respective aggregation and defuzzification operators 'sum' and 'sugeno', and without the classification option.

These parameters can be modified by editing the iris.hfp file. For the iris data, the classification option should be chosen, as the output is a variety.

(b) **hfpvertex** to calculate the fuzzy sets bounds in the various partitions

Arguments:

- data file name
- HFP configuration file name

Two optional parameters:

-oVertexFile where VertexFile is the name of the output vertex file (default: vertices.'hierarchy type', i.e. regular, kmeans ou hfp)

-kn This option only concerns the hfp hierarchy. If given, n is the number of groups used to form the initial partition in the call to the k-means algorithm. Otherwise, the initial partition is formed by grouping data into unique values, according to the tolerance given in the hfp configuration file.

Command line example:

hfpvertex iris iris.hfp

Creates the file vertices.regular, if the hierarchy is a regular grid

hfpvertex iris iriskm.hfp -oiris.vertices

Creates the file iris.vertices for the kmeans hierarchy

(c) **hfpselect** to automatically select the number of MFs per variable.
Arguments:

- the FIS configuration file name
- the data file name

Optional parameters:

-r: choose wm as rule induction method (default: fpa is used)

-tx where x is the strategy to determine the data subset used to initialize the rule conclusion, 0 for MIN, 1 for DEC (default value).

-my where y is the minimum matching degree (default value: 0.3)

-ez where z is the minimum cardinality (default value: 3)

-oFIS where FIS is the output FIS configuration filename (default value 'system name'.fis)

-sw where w is the minimum cumulated weight for a rule to be generated (0.0 default value)

-bc where c is the minimum coverage level(default value: 1.0, 100 %)

-nf where f is the initial number of fuzzy sets per variable (default value: 1)

-ig ehere g is the maximum number of iterations (default value: 10)

-lFileV where FileV is the name of the vertex file, created by hfpvertex (default value: vertices.Hierarchy)

-pNum where Num is the output number (défault: 0)

-vFileTest where FileTest is the name of a data file used for validation (default value: the filename given in the second argument)

Command line example:

hfpselect iris iriskm.hfp -b0.7 -e2 -m0.3 -liris.sommets

This program creates two files called result and result.min. The first one has as many lines as the number of attempts to complexify the FIS. In the second one, only the configurations kept at each step appear

Reminder of the command lines for the partition selection procedure:

- *hfpcfg iris 1 iriskm.hfp 2*

- *hfpvertex iris iriskm.hfp*

Set the output Classif flag to 'yes' in the iriskm.hfp file.

- *hfpselect iris iriskm.hfp -b0.7 -e2 -m0.3*

Note: As described in the user documentation, the four fields called In1, In2, In3, In4 indicate the number of MFs per input variable.

2 Rule induction using the current FIS

The rule induction methods presented here use a FIS configuration file. All existing FIS rules are ignored.

1. **FPA method**

   Two C++ programs are required, **genrules** and **fpa**. In Java, they are combined in the *FPA* option (*Learning* menu, *Induction* submenu).They can also be used independently, by first generating the rules, then their conclusions.

   Java interface:

   - *FIS* menu, *Generate rules* option.
     This option behaves differently if there is a current data file or not.
   - *FIS* menu, *Generate conclusions* option (*fpa* is the default method.

   Command lines:

   - **genrules** to generate the rules
     Argument: FIS configuration file name
     Optional arguments:

     -fFISFile where File is the output FIS configuration filename (default value config.fis)

     -r to remove work files

     other options: associated to the call with a data file name (call genrules without argument to get their description).

     The created rules have a conclusion equal to one. They are stored in a file called 'system name'.rules, this filename appears in the Rules section of the output FIS configuration filename.

     This program also creates an intermediate file called info.gen.

     Command line example:

     genrules iris-sr.fis

     Creates the files config.fis (FIS) and info.gen.

     or

     genrules iriskm.fis -firiskmr.fis

     Creates the file iriskmr.fis.

     other example:

     genrules irisreg.fis -firisregr.fis

     Creates the file irisregr.fis

- **fpa** to set the rule conclusions

  Arguments:

  - the FIS configuration file name
  - the data file name

  Optional arguments:

  -a for detailed display

  -fFile where File is the output FIS configuration filename (default value: config.fpa)

  -sx where x is the strategy to determine the data subset used to initialize the rule conclusion, 0 for MIN, 1 for DEC (default value).

  -dy where y is the minimum matching degree for rule generation (default value: 0.3).

  -ez where z is the minimum cardinality (default value: 3)

  -tFileV data file name used for performance computation (default : data file)

  -u activity threshold for performance computation (default value: 0.1)

  -r to remove work files

  The meaning of these parameters is given in the user guide, section Learning Menu, Generate conclusions. The number of generated rules is highly dependent on them.

  Command line example:

  fpa iriskmr.fis iris

  Creates the file configfpa.fis, which has 26 rules, while there were 81 (3*3*3*3) rules in iriskmr.fis.

  or

  fpa iriskmr.fis iris -d0.1 -e1 -firiskmrfpa.fis

  which creates the file iriskmrf.fis with 55 rules.

  other example:

  fpa irisregr.fis iris -d0.0 -e1 -firisregfpa.fis

  which creates the file irisregfpa.fis with 36 rules..

  . These configuration files are usable in the inference process.

2. **Wang & Mendel**

   Java Interface:

   Learning menu, *Rule induction* submenu, *wm* option

Command line

The **wm** program requires the following arguments:

- the FIS configuration file name
- the data file name

Optional parameters:

-tfile data file used for performance calculation (default is second argument)

-oFIS the output FIS file name (default value 'system name'wm.fis).

-sThresh activity threshold for performance calculation (default value=0.2)

-l 'No limit for output distinct values in data file' (default=false)

Command line example

wm iriskmr.fis iris

Creates the file iriswm.fis with 20 rules

or

wm iriskmr.fis iris -owmiris.fis

which creates the same file called wmiris.fis

3. **ols** will be presented in the section 3.

4. **Fuzzy decision tree**

   Java Interface: Learning menu, Rule induction submenu, Tree option

   Command line, :fistree program

   The **fistree** program requires two arguments and has several optional arguments.

   Arguments:

   - the FIS configuration file name
   - the data file name

   Optional parameters:

   -oNum where Num is the output number (Default Value: 0, first output)

-sx where x is the minimum membership for an item to be considered as attracted by the node for entropy calculations (default value 0.2)

-xCard where Card is the minimum leaf cardinality (default=min(10,#rows/10))

-ty where y is the tolerance on the membership to the majority class (default value 0.1)

-dn where n is the tree maximum depth (default value: 0, means no limit)

-gval where val is the minimum relative entropy/deviance gain for splitting nodes (default value 1e-6)

-e0 absolute entropy gain, -e1 relative gain (default=absolute)

-gval where val is the minimum entropy/deviance relative gainen entropie/deviance required to create a branch

-p0 no pruning, -p1 full split pruning according to a performance criterion, -p2 leaf pruning according to a performance criterion (default is no pruning)

-lw where w is the relative performance loss tolerated during tree pruning (default value 0.0)

-vValidFile where ValidFile is the validation file name for performance calculation during pruning (default is data file)

-a detailed display

-a0 semi detailed display

Tree building, as well as tree pruning, creates two files. Output filenames are generated from the FIS configuration filename. The file with the .tree suffix contains a tree summary, which can be viewed in the java interface, or studied in a text editor. The second file is the corresponding FIS configuration file. It has a .fis suffix.

A file called result.fistree is also created. It is similar to the file result.simple created by the fisimple program (see 5), with some extra information specific to fuzzy trees.

The tree is a regression tree (used criterion=deviance) if the output declared in the FIS configuration file is fuzzy, with non discrete MFs. Otherwise it is a classification tree (used criterion=entropy), with inferred values corresponding to crisp classes if the output is crisp, with the flag classif=yes, or else to fuzzy classes.

Command line example:

fistree iriskmr.fis iris -s0.3

Creates iriskmr.fis.sum.tree and iriskmr.fis.tree.fis, which has 9 rules.

fistree iriskmr.fis iris -s0.1 -p1 -l0.1

Creates the two files above, plus two files for the pruned tree, iriskmr.fis.prunedsum.tree and iriskmr.fis.prunedtree.fis. In this case, the minimum membership threshold is lower than in the previous example, which produces a tree with 34 rules before pruning and 4 after pruning.

5. **HFPFIS**

   To generate the FIS configuration file corresponding to a given MF combination, two C++ programs are required, sethfpfis et hfpfis. In Java, they are combined in a single option.

   In C++:

   - **sethfpfis** to create a customized hfp file, which can later be used as an argument of hfpfis.

     Three arguments are required:

     – the input hfp configuration file
     – the number of fuzzy sets per variable, given as a character string
     – the output hfp configuration file

     Command line example:

     sethfpfis iriskm.hfp "5 2 3 3" irissel.hfp

     Creates the file irissel.hfp

     Java Interface:

     Learning menu, Partitions submenu, HFP MF submenu, Edit HFP file option

   - **hfpfis** to generate a FIS configuration file from the hfp configuration file.
     It requires the same arguments than hfpselect (except those specific to iterative search in hfpselect).

     Arguments:

     the FIS configuration file name

     the data file name

     Optional arguments:

     -r: choose wm as rule induction method (default: fpa is used)

-tx where x is the strategy to determine the data subset used to initialize the rule conclusion, 0 for MIN, 1 for DEC (default value)

-my where y is the minimum matching degree (default value: 0.3).

-ez where z is the minimum cardinality (default value: 3)

-oFile where File is the output FIS configuration filename (default 'system name'.fis)

-sw where ww is the minimum cumulated weight for a rule to be generated (0.0 default value)

-lFileV whereFileV is the name of the vertex file created by hfpvertex (default value: vertices.'Hierarchy')

-pNum whereNum is the output number (default value: 0, first output)

Command line example:

hfpfis iris irissel.hfp -e2 -m0.3 -oirissel.fis -liris.sommets

Creates the file irissel.fis with 13 rules.

3 Generate partitions and induce rules using OLS

The **ols** (Orthogonal Least Squares) program transforms each example from a data file into a fuzzy rule, and selects the most important rules according to a least square criterion. It uses linear regression and Gram-Schmidt orhtogonalization. Once the rules selected, a second passage of the algorithm is done to optimize the rule conclusions. This algorithm is well suited to regression problems.

Java interface:

Learning menu, *Rule induction* submenu, *OLS* option.

Command line, ols program:

A single argument:

- data file name

Options:

-nNumOutput: NumOutput to consider (default: 0=first output)

-oNbOutputs: NbOutputs is the number of outputs (default: 1)

-j: to use original algorithm (Hohensohn and Mendel) (default: not original algo)

-tTol: Tol is a parameter used for partition generation.
2 cases: in the original algorithm, Tol is the Standard deviation used for Gaussians(default: 0.05), otherwise Tol is the tolerance threshold to determine the MF centers. (default: 0.25)

-s: does not impose standard fuzzy partitions (default=standard fuzzy partitions)

-fFISfile: FISfile is the input FIS configuration file
In that case the fuzzy partitions are kept
The rules are ignored, and will be replaced by generated rules, except if the -x option is given.

-x: this option is useful only if the previous option is given (-fFISfile).
The program keeps the FIS partitions and rules, it only changes the rule conclusions by a least square optimization procedure based on the data file. This option allows to optimize the rule conclusions of any FIS.

-gTotV: Total unexplained variance stop condition (default: 1 percent not explained)

-qNrules: Number of selected rule stop condition (default: 10000)

-v: to reduce output vocabulary (default: no reduction)

-i: this option is useful only with the previous option -v, 2 possibilities to set rule conclusions (voir section 6)

i=0: the new rule conclusions will be chosen according to the data file
i=1 (default value): the new rule conclusions will be chosen according to the initial rule conclusions.

-dPerfLoss: PerfLoss is the relative performance degradation allowed by vocabulary reduction (default: 0.1)

-mNConc:
NConc is the maximum number of elements allowed in the reduced vocabulary (default=no limit)
IF BOTH PerfLoss AND NConc ARE GIVEN, PRIORITY TO NConc.

-l: fuzzify the ouput (works only if vocabulary is reduced).

-pTestFile: the data file name for the second pass (default: the same than for the 1st pass)

-cOutputFIS: OutputFIS is the generated FIS file name (default: 'DataFileName'.fis)

-r to remove temporary files ('data'.mat and summary.ols)

-btestdatafile: testdatafile is the data file used to compute performance (default: 'DataFileName')

-eMumin: Mumin is the activity threshold to compute performance (default: 0.2)

-a: intermediate display (default: no display)

Command line example:
ols rice
The program creates 5 files:

- **rice.ols**.

```
Num Index VarExp VarCum

1,18,0.305448,0.305448,
2,1,0.176927,0.482375,
3,85,0.110070,0.592445,
4,2,0.086740,0.679184,
5,25,0.070750,0.749935,
6,31,0.038250,0.788185,
7,28,0.035713,0.823898,
8,64,0.025145,0.849042,
9,4,0.019027,0.868070,
10,8,0.015855,0.883925,
11,49,0.013378,0.897303,
12,12,0.012828,0.910131,
13,100,0.012922,0.923054,
14,34,0.012031,0.935085,
15,35,0.007077,0.942162,
16,66,0.007068,0.949231,
17,42,0.006960,0.956190,
18,5,0.006750,0.962941,
19,79,0.005379,0.968320,
20,22,0.004892,0.973212,
21,20,0.005113,0.978325,
22,71,0.004085,0.982410,
```

```
23,6,0.003055,0.985465,
24,102,0.002667,0.988132,
25,53,0.001796,0.989928,
```

A line per induced rule.

For each line, the following columns:

1. rule number
2. data file row number used to build the rule
3. percentage of variance explained by the rule
4. cumulated percentage of explained variance

- **rice.mat**: data matching degree matrix
- **irisrice.fis**: generated FIS configuration file
- **perf.ols**: performance results (see section 11).
- **result.ols** summary (see section 11).

4 Optimization

Java Interface: *Learning* menu, *Optimization* submenu: Two options:

1. *Custom Solis & Wets* option.
2. *Standard Solis & Wets* option.

Command line, loopoptims program:
The **loopoptims** program requires the following arguments:

- the FIS configuration file name
- the data file name

default=optimize rule conclusions
Optional parameters:

-b the basename of the resulting FIS configuration file;(default optim)

-it the number of iterations; (default 100)

-e the range of the gaussian noise; (default 0.005)

-v the max number of constraints violations before it counts as an iteration (default 1000)

-f the max number of failure steps in an algorithm (default 1000)

-d the equality center distance threshold (default 1e-6)

-ns Number of sample pairs to generate from data file (default 10-put -ns 0 for no sampling)

-cs Draw samples with respect to class ratio in data file (class is last column, default no))

-rs Ratio learning/all pairs (default 0.75, maximum 0.9)

-s integer to set seed value for parker miller random;(default 0:new sampling each time);

-in "x y ", the string of input numbers to optimize (starting at 1, order is important, default no input optimization)

-r optimize rule conclusions

-o optimize fuzzy output (default false)

-n Output number to consider (default 0: first output);

-m minimum membership (default 0.1)

-l1 Solis Wetts Constant 1 (default 0.4)

-l2 Solis Wetts Constant 2 (default 0.2)

-l3 Solis Wetts Constant 3 (default 0.5)

-u Number of loops for optimizing (default 2)

-c relative tolerated loss of coverage (default 0.10 ; 1.=10.0%)

-nc minimum distance between MF centers (default=1e-3)

-g create intermediate files (default false)

-a for intermediate display (default false)

-wl for wordless.(default is not wordless)

-nc distmin to impose a minimum distance between the neighboring MF kernels (default 0.001).

Command line example:

loopoptims rice.fis rice -ns 2 -s 102 -b optimfis

which optimizes the rule conclusions for the FIS configuration file rice.fis, with a cross-validation procedure including 2 pairs of learning(75%)-test (25 %) samples, and stores the optimized FIS for each learning sample, in the optimfis-lrn.sample0-final.fis and optimfis-lrn.sample1-final.fis files. The med.fis file is also built using the median parameters of the optimized FIS.

or

loopoptims rice.fis rice -ns 2 -s 102 -b optimfis -in '1 3' -r

which optimizes the fuzzy set parameters of the input variables 1 and 3, as well as the rule conclusions for the FIS configuration file rice.fis. The generated files have the same names as above.

A perf.res file is written, that includes the accuracy of the initial FIS, of each optimized FIS and of the medain FIS, calculated for the initial dataset, each test sample and each learning sample.

5 Simplification

Java Interface:

Learning menu, *Simplification* option.

Command line, fisimple program: The **fisimple** program tries to group rules together, to remove rules or to eliminate some variables from a rule.

Arguments:

- the FIS configuration file name
- the data file name

Optional arguments:

-t performance test file name (default: data file name)

-pNum where Num is the output number (default value: 0, first output)

-bAbsCov where AbsCov is the absolute coverage level: tolerated percentage of blank samples (default: 0.1, 10

-cCovLoss where CovLoss is the relative tolerated loss of coverage during rule removal (default: 0.0

-dAbsPerf where AbsPerf is the absolute maximum performance index (default: -1.0)

-ePerfLoss where PerfLoss is the tolerated loss of performance, percentage of initial error (default: 0.1, 10

-hHomog where Homog is the output homogeneity threshold

-k do not keep the last rule with a given class or MF label as conclusion (default: keep the last rule)

-r to remove work files

-a for detailed display

-q for rule removal

-l for variable removal

-tFileTest where FileTest is the name of a validation data file used for performance calculations only (default value: the data file name)

-sMuMin where MuMin is the activation threshold for an item not to be blank (default: 0.2), used for performance calculations only.

Remarks: the arguments -b and -c are exclusive, -b has priority over -c. The arguments -d and -e are exclusive, -d has priority over -e.

Command line examples:

fisimple wmiris.fis iris

The program displays the message
'The most simple FIS: irissel.fis.jb.2'. The whole set of simplified FIS configurations is given in the file result.simple. The simplest FIS is wmiris.fis.jb.2 which has eleven rules.

or

fisimple iriskmr.fis.tree.fis iris

The simplest FIS is iriskmr.fis.tree.fis.jb.2 which has eleven rules.

fisimple rice.fis rice -q -l

The simplest FIS is rice.fis.jb.23 which has 9 rules.

6 Reducing the output vocabulary

In the case of a crisp regression output, the rule conclusion values are all different from each other. Reducing the output vocabulary improves the readability of the rule base.

Two choices are available. With the first one (default one), a clustering is performed using the rule conclusions, with the second one it is done by using the data file output values. The clustered values are chosen as the new rule conclusions.

The number of distinct conclusions can be set, or the tolerated loss of performance. Indeed reducing the voabulary usually goes with a loss of accuracy.

Java interface:
FIS menu, *Reduce the output vocabulary* option.
Command line, vocreduc program:
Argument:

- FIS configuration filename
- data filename

Options:

-oNumOutput where NumOutput is the output number (default: 0=first output)

-dType data used to make vocabulary reduction

- -d1: rule conclusions are generated by clustering the initial rule conclusions
- -d0: rule conclusions are generated by clustering the output data values in the data file

-lPerfLoss: PerfLoss is the relative performance loss allowed by vocabulary reduction (default: 0.1)

-cConc: Conc is the number of elements in the reduced vocabulary (default: 10000).

Remark: these 2 argument default values are 0.1 for PerfLoss, which yields an automatic determination of Conc.

-sMuMin where MuMin is the activity threshold for an item not to be blank (default: 0.2)

-a: detailed output

Command line example
vocreduc rice.fis rice
The number of conclusions passes from 25 to 6.

7 Links

This utility allows to view the links between rules and sample items, and also the links between the various rules.

Java interface:
Data menu, *Links* option.
Les fichiers sont générés et visualisés dans des fenêtres texte.
Command line, fislinks program:
Argument:

- the data file name

Command line example:
fislinks iris.fis iris

It creates several work files, with a default name prefix equal to the data filename.

- rules.items

  This file has as many lines as there are rules in the system, plus 2.

  * First line: number of rules
  * Second line: maximal number of items that activate a rule
  * Third line: description of the first rule, i.e. the rule number (starting from 1), cumulated weight, number of items that activate a rule (beyond a threshold parameter with default value equal to 1e-6) followed by their number.

- items.rules

  This file has as many lines as there are items in the data sample.

  For each line, the item number (starting from 1), followed by the rule numbers that are activated by it.

- rules.links

  The notion of link between rules is useful to appreciate the consistency of a rule base. If two rules are strongly linked, and have different conclusions, then the inconsistency can come from an insufficiently specific input range rule coverage. This situation can also correspond to an exception in the data sample. The linkage level of the i_{th} rule with the j_{th} rule is calculated as follows:

$$L_{i,j} = \frac{N_{i,j}}{N_i}$$

N_i is the cardinal of the subset E_i of the items that activate the i_{th} rule, $N_{i,j}$ is the cardinal of $E_i \cap E_j$.

This file is formatted as a square matrix, whose size is equal to the number of rules. The i, j cell gives the corresponding linkage level. Note: the matrix is not symmetric.

- rules.sorted

  if the sort option is selected, rules are sorted by cumulated weight (which represents their influence in the data sample).

Note: links do not depend on the FIS outputs.

8 Sample generation

Sampling generates learning and test samples from a data file.

Java interface: *Data* menu, *Generate sample* option.

The files are generated in the FisPro working directory, or in the bin subdirectory of the FisPro root directory, depending on the operating system.

Command line, sample program:

Argument:

- the data file name

Optional arguments:

-nNs where Ns is the number of sample pairs (default value: 1 creates on learning sample and one test sample

-pApp where App is the data file size ratio used to determine the learning sample size (default value: 0.75)

-sSeed
the same Seed value will reproduce the same samples
(default value, 0 gives new random samples at each call)

-c to create samples which respect the class proportion in the data file

-oNumC
used with the -c option, to give the column number used to assign classes in the data file (default value, last column)

-eTol
used with the -c option, Tol=tolerance to assign classes (default value: 1e-2)

-a for detailed display

Command line example:
sample iris -n4 -c
generates the 8 following files using the iris data file:
iris.lrn.sample.0,iris.lrn.sample.1,. . . , iris.tst.sample.0,iris.tst.sample.1, . . . , with class proportions identical to the iris data (species 1,2,3).
The iris.lrn.sample.0 and iris.tst.sample.0 files form one learning-test pair, and so on.

9 Performance

Java Interface:
Data menu, *Infer* option.
Command line, perf program:
The **perf** program calculates the FIS performance on a data set.
Arguments:

- the FIS configuration file
- the data file name

Optional arguments:

-nNum where Num is the output number (default value: 0, first output)

-sVal where Val is the minimum activity threshold to coonsider an item as non blank (default value: 0.2)

-a for detailed display.

Command line example:
perf irissel.fis iris
returns the value 4 (misclassified items)
or

perf riceoptrules.fis rice

returns 0.004819 (index based on the quadratic error)

Perf creates the perf.res file, whose structure is described in the user documentation.

10 Check a data file

Note: only available as command line utility.

Command line, readdata program:

The **readdata** program reads a data file and detects the field separator.

Argument:

- the data file name.

Result:

The program gives the number of rows and columns and the column names found in the heading (if present).

11 Result files of learning programs

The learning procedures cited below always create 2 files: *result.nomproc* and *perf.nomproc*. The table gives the name correspondence between the procedure names and the file names.

program name	result file	performance file
fisimple	result.simple	perf.simple
fisopt	result.fisopt	perf.fisopt
fistree	result.fistree	perf.fistree
fpa	result.fpa	perf.fpa
ols	result.ols	perf.ols
perf	result.perf	perf.res
wm	result.wm	perf.wm

Summary file

result.nomproc is a summary file. It has only one line which describes the generated FIS:

first column: initial FIS filename

second column: performance index

third column: coverage index

fourth column: max error

following columns: rule base characteristics

Rule base characteristics

- maxR : maximum possible number of rules according to the fuzzy partitions
- nR : number of rules
- maxVr: maximum variable number within a rule
- meanVr: average variable number within a rule
- nVar : maximum number of distinct variables used by a rule
- meanMF: average number of distinct variables used by a rule
- nClass: number of output classes/MFs
- nRc : number of rules per class/MF

Performance file

General Format

The first line of the result file is a column header. The possible labels are defined in the file fis.h:

- "OBS": The observed output, part of the data set
- "INF": The inferred output
- "Al": Alarm risen while inferring (see below)
- "ClINF": Inferred class label, for crisp output and classification flag
- "CLAl": Alarm risen while inferring (see below)
- "Err": Difference between inferred and observed output
- "Bl": Blank flag, 1 if this item is considered as blank, o otherwise
- "CErr2": square cumulated error over the previous examples.

WARNING: The cumulated error does not take into account the blank examples.

The first column holds the observed output, if it is available in the data file, as inference can also be done without an observed output.
The second one gives, when the observed output value is available, the difference between observed and inferred values.
A variable number of columns follows, depending on the output nature.

Output	Classif	Defuzzification	Field #	Available fields			
crisp	yes	sugeno	4	infered v.	Alarm	infered class	Alarm
crisp	no	sugeno	2	inferred v.	Alarm		
crisp		MaxCrisp	2	inferred v.	Alarm		
fuzzy	yes	sugeno/area	n+2	infered v.	Alarm	$\mu_1, \mu_2 \ldots \mu_n$	
fuzzy	no	sugeno/area	2	infered v.	Alarm		
fuzzy	yes	MeanMax	n+2	infered v.	Alarm	$\mu_1, \mu_2 \ldots \mu_n$	
fuzzy	no	MeanMax	2	infered v.	Alarm		

Note: $\mu_1, \mu_2 \ldots \mu_n$ are available if the example activates at least one rule. For a fuzzy output defuzzified with *sugeno* ou *area* operators, they are equal to the inferred output value membership degrees to MF $1, 2 \ldots n$. For a fuzzy output defuzzified with a *MeanMax* operator, or for a crisp output obtained through a *MaxCrisp* operator, they represent the matching degree of all possible outputs: MF values for a fuzzy output, or real values for a crisp output.

The column *Bl* is 1 if the item is inactive, 0 otherwise. Finally, the last column gives the current value of the performance index.

Available alarm values:
Integer values which depend on the defuzzification operator:

- NOTHING (Value 0): All types. Everything is normal.
- NO_ACTIVE_RULE (Value 1): All types. The example does not fire any rule of the rule base.
- AMBIGUITY (Value 2): SugenoClassif (Crisp output) and MaxCrisp. The difference between the two main classes is less than a threshold (default value AMBIGU_THRES = 0.1).
- NON_CONNEX_AREA (Value 3): WeArea - Set when the area defined by the fired fuzzy sets (threshold set to MIN_THRES = 0.1 by default) is non connex.

- NON_CONNEX_SEGMENT (Value 4): MeanMax - Set when the max corresponds to two fuzzy sets (with a tolerance threshold set by default to EQUALITY_THRES = 0.1) and the resulting segment is non connex.

The performance function also computes the coverage index, which depends on the minimum matching degree given as an argument.