Démarches d'apprentissage avec FisPro

La conception d'un système d'inférence floue à partir de données d'apprentissage peut être découpée en deux étapes essentielles :
- le partitionnement des variables d'entrée et de sortie, section 1
- l'induction des règles, sections 2 et 3.

Les partitions comme les règles peuvent ensuite être optimisées, section 4. Le système résultant peut être simplifié, section 5, pour obtenir des règles incomplètes (voir glossaire).

Les programmes d'apprentissage sont disponibles en ligne de commande et sont également interfacés en Java. Le temps d'exécution du programme peut augmenter très rapidement avec la taille du fichier de données (lignes et colonnes). Cela est particulièrement sensible pour les programmes d'optimisation et de simplification. Dans ce cas, il est préférable d'utiliser les versions en ligne de commande, éventuellement en lançant les programmes en arrière plan et en redirigeant les affichages du programme dans un fichier.

Depuis la version 3.4 de *Fispro*, tous les programmes d'apprentissage disposent, en version ligne de commande, de l'option -wl qui permet un fonctionnement silencieux. Seuls les messages d'erreur restent affichés. Cette option a priorité sur l'option -a.

Table des matières

Liste (par ordre alphabétique) de tous les programmes C++ disponibles en ligne de commande :

Certains de ces programmes sont des utilitaires : **genrules, fislinks, perf, readdata, sample**. Les **autres** sont des programmes d'**apprentissage**.

Le programme **fis** est destiné aux développeurs de nouveaux modules de FisPro. Il correspond au programme source **testfis.cpp**, situé dans le répertoire cpp/base de fispro, et donne des exemples d'appel de fonctions de fispro (voir les commentaires du source pour plus de détails).

- **dist** : création d'une matrice de distances
- **fis** : exemple d'utilisation d'un SIF sur un jeu de données
- **fisimple** : simplifier un SIF (voir section 5)
- **fistree** : générer des arbres de décision flous et les SIF correspondant (voir section 2)
- **fpa** : générer les conclusions des règles d'un SIF (voir section 2)
- **genrules** : générer les prémisses des règles d'un SIF à partir de ses partitions (voir section 2)
- **hfpcfg** : créer le fichier de configuration hfp (voir section 1)
- **hfpfis** : générer le fichier de configuration SIF à partir du fichier de configuration hfp (voir section 1)
- **hfpselect** : sélectionner le nombre de fonctions d'appartenance (SEF) par variable (voir section 1)
- **hfpsr** : générer un fichier de configuration du SIF sans règles (voir section 1)
- **hfpvertex** : calculer les centres des sous-ensembles flous à partir d'un fichier de configuration hfp et d'un fichier de données (voir section 1)
- **fislinks** : calculer des éléments représentatifs des liens entre les règles d'un SIF et des données d'un fichier (voir section 7)
- **loopoptims** : optimiser des parties d'un SIF avec validation croisée (voir section 4)
- **ols** : générer des règles par OLS (orhogonal least squares (voir section 3)
- **perf** : calculer la performance d'un SIF sur un jeu de données (voir section 9)
- **readdata** : vérifier la lecture d'un fichier texte avec délimiteurs (voir section 10)
- **sample** : générer des couples d'échantillons d'apprentissage et de test à partir d'un jeu de données(voir section 8)
- **sethfpfis** : modifier un fichier de configuration hfp
- **vocreduc** : réduire le vocabulaire de sortie d'un SIF (voir section 6)
- **wm** : générer les règles d'un SIF et leurs conclusions (voir section 2)

Note valable pour tous les programmes C++ de FisPro :
Appel du programme sans argument : affiche les arguments possibles et leur

usage.

Les différentes étapes de la démarche sont dans un premier temps brièvement décrites, puis l'ensemble de la procédure est illustré sur un jeu de données. Pour chacun des programmes sont indiqués les arguments de la ligne de commande (C++) ainsi que l'accès à l'option correspondante par l'interface (Java). La documentation de l'utilisateur décrit plus en détail leur fonctionnement.

Les fichiers mentionnés dans les lignes de commande sont livrés dans le répertoire EXEMPLES.

Notations :

SIF=système d'inférence floue

SEF=sous-ensemble flou

1. **Partitionnement**

   Les partitions induites par FisPro sont des partitions floues fortes (voir glossaire) pour garantir qu'à chacun des sous-ensembles flous correspond un label linguistique. Une seule exception : les partitions de gaussiennes générées par l'OLS avec l'algorithme original de Hohensohn et Mendel.

   Trois types de partitions sont disponibles : hfp, k-means, regular.

   Le menu *Apprentissage* propose deux options pour générer les partitions : *Générer un SIF sans règles*, et *HFP SEF*.

   La méthode la plus simple est l'option *Générer un SIF sans règles*, disponible aussi dans le menu *SIF* et dans l'OLS. Dans cette option, il faut préciser le nombre de sous-ensembles flous par variable d'entrée ou de sortie, et le type de la partition.

   L'option *HFP SEF* permet de choisir automatiquement le nombre de SEF par partitions en fonction des données. Cette option est constituée d'une séquence d'étapes décrites en détail dans la section 1.

2. **Induction des règles**

   Plusieurs méthodes sont disponibles dans FisPro pour induire automatiquement les règles d'un SIF :
   - Générer l'ensemble des règles possibles puis initialiser les conclusions par la méthode *FPA*.
   - Utiliser l'algorithme *wm* de Wang & Mendel.
   - Construire un *arbre* de décision flou, éventuellement élagué.
   - Utiliser l'algorithme *HFP SIF*
   - Utiliser l'algorithme *OLS* (orthogonalisation par les moindres carrés).

   Les trois premières méthodes utilisent à la fois un fichier de configuration SIF, qui sert à définir les partitions, et un fichier de données.

   La méthode *HFP SIF* travaille uniquement à partir des données, elle a besoin de 2 fichiers particuliers : un fichier HFP et un fichier de sommets, décrits

dans la section 1, et permet de générer les règles correspondant à l'option *HFP SEF* du menu *Apprentissage/Partitionnement*. L'algorithme consiste, en partant de la configuration la plus simple, à la complexifier en ajoutant à chaque fois un SEF à la partition d'une variable, et à rechercher la configuration la plus performante parmi les plus compactes. Le nombre de sous-ensembles flous par variable n'est pas spécifié a priori.

3. **Génération de partitions et induction de règles par OLS**

   La méthode *OLS* peut travailler uniquement à partir des données, elle génère alors les partitions, soit des partitions floues fortes formées de triangles et de trapèzes, soit des partitions formées de gaussiennes, puis les règles. Elle peut aussi prendre en compte un fichier de configuration SIF, elle conserve alors ses partitions, et génère uniquement les règles.

4. **Optimisation**

   L'option *optimisation* du menu *Apprentissage* permet d'ajuster la position des sous-ensembles flous des entrées ou des sorties, ou encore d'optimiser la conclusion des règles.

5. **Simplification**

   Enfin, l'option *simplification* va tenter de simplifier le système en regroupant des règles entre elles ou en supprimant les règles les moins utiles.

6. **Réduction du vocabulaire de sortie**

   Le vocabulaire de sortie désigne les conclusions distinctes des règles dans un SIF. Les procédures d'apprentissage engendrent souvent des règles avec des conclusions toutes différentes, tirées d'un ensemble d'exemples. Pour plus d'interprétabilité, on peut modifier légèrement ces conclusions en limitant leur nombre.

7. **Liens**

   Le calcul des liens règles-exemples permet d'exprimer quantitativement le degré d'activation des règles par les exemples d'un fichier de données, de connaître les règles activées par chaque exemple, et réciproquement les exemples activant chaque règle.

8. **Echantillonnage**

   L'option *Générer échantillons* du menu *Données* permet de générer des échantillons à partir d'un fichier de données, afin de créer des jeux d'apprentissage/test.

9. **Performance et données**

   Ce programme, ou cette option, permet de mesurer la performance d'un SIF sur un jeu de données.

   Nous proposons deux jeux de données de test. Le premier est un exemple de classification, les iris, tandis que le second, le riz, est un problème de régression : la sortie est une valeur continue.

Les iris (Fisher, 1936) sont des données formées de quatre variables d'entrée : longueur et largeur des sépales, longueur et largeur des pétales. La sortie est une des trois espèces suivantes : Setosa, Virginica ou Versicolor. L'échantillon est constitué de 50 individus de chacune des classes.

Le jeu de données 'rice' (Ishibuchi, 1994) contient des appréciations sensorielles collectées sur plusieurs variétés de riz par un panel d'experts. L'échantillon est constitué de 105 individus. Les cinq caractéristiques d'entrée sont : l'arôme, l'apparence, le goût, le collant, la dureté. La sortie est une évaluation globale. La procédure est illustrée avec l'exemple des iris. Tous les fichiers manipulés sont au format texte et peuvent être modifiés par un éditeur quelconque.

10. **Vérification des données**

    Le programme **readdata** permet de vérifier que la lecture des données : nombre de lignes, de colonnes, en-tête, séparateur, est conforme à ce qui est attendu.

11. **Fichier résultats de l'apprentissage**

    Chaque procédure d'apprentissage crée un certain nombre de fichiers de résultats, dont 2 ont un format commun à toutes les procédures, et sont présentés en détail dans la section 11.

1 Partitionnement

1. **Générer un sif sans règles** permet de générer un fichier de configuration du SIF, en spécifiant le nombre de SEF pour chacune des entrées et pour la sortie, ainsi que les types de hiérarchie utilisés pour les entrées et la sortie. Le SIF créé n'a pas de règles.

   Interface Java :

   menu *Apprentissage*, sous-menu *Partitions*, option *Générer un SIF sans règles*.

   En Ligne de commande, programme hfpsr :

   Arguments :
   - le nom du fichier de données
   - le nombre de SEF pour chacune des entrées (argument de type chaîne de caractères dans laquelle les nombres sont séparés par des espaces)
   - le type de hiérarchie pour les entrées : 1 hfp, 2 kmeans, 3 regular
   - la valeur de la tolérance pour les entrées pour regrouper les valeurs uniques, si valeur<1, ou bien le nombre de groupes pour les k-means, si valeur>1 (ne sert que pour hiérarchie hfp).
   - le nombre de SEF pour la sortie (0 pour 1 sortie nette)
   - le type de hiérarchie pour la sortie : 1 hfp, 2 kmeans, 3 regular
   - l'opérateur de défuzzification : area, MeanMax ou sugeno
   - l'opérateur de disjonction : sum ou max

– la valeur de la tolérance pour la sortie pour regrouper les valeurs uniques, si valeur<1, ou bien le nombre de groupes pour les k-means, si valeur>1 (ne sert que pour hiérarchie hfp).

Options :

-oFIS où FIS est le nom du fichier de configuration créé (par défaut : Nom du système-sr.fis)

-cConj où Conj est l'opérateur de disjonction (par défaut : prod)

-vVertexFile où VertexFile est le nom du fichier de sommets des entrées (voir **hfpvertex**, par défaut : les sommets sont calculés et le fichier est crée).

-f : fixe l'option Classif='yes' dans la sortie

Exemple de ligne de commande :

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

Crée les fichiers iris.hfp, iris.vertex et iris-sr.fis avec 3 SEF par entrée de type hfp et une grille régulière de 3 SEF pour la sortie.

ou bien

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

Crée les fichiers iris.hfp, iris.vertex et iriskm.fis pour lequel les partitions des entrées sont de type k-means.

ou bien

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

Crée les fichiers iris.hfp, iris.vertex et irisreg.fis, avec des grilles régulières de 2 SEF pour les 2 premières entrées, de 3 SEF pour les 2 autres, une sortie nette de type classif.

Le fichier de configuration du SIF sans règle ainsi créé peut être utilisé comme argument pour les méthodes d'induction de règles.

2. **Sélectionner les partitions**

   Cette étape se décompose en plusieurs sous-étapes :

   Interface Java :

   (a) menu *Apprentissage*, sous-menu *Partitions*, sous-menu *HFP SEF*, options *Générer un fichier hfp* et *Editer un fichier hfp*

   (b) menu *Apprentissage*, sous-menu *Partitions*, sous-menu *HFP SEF*, option *Générer les sommets*.

   Les sommets peuvent être visualisés par l'interface (option Visualiser les sommets). Le nombre maximum de sommets générés pour chacune des entrées est celui spécifié dans le fichier de configuration hfp, NMFs=7 (7 est la valeur par défaut).

   (c) menu *Apprentissage*, sous-menu *Partitions*, sous-menu *HFP SEF*, option *Sélectionner partition*.

   En lignes de commande :

(a) programme **hfpcfg** pour créer le fichier de configuration hfp

Arguments :

– fichier de données
– nombre de colonnes à ne pas traiter. Typiquement la valeur 1 indique de ne pas traiter la dernière colonne, qui est la sortie.

Deux arguments optionnels :

le nom du fichier hfp de sortie (par défaut le nom du fichier de données.hfp)

le type de hiérarchie : 1 pour hfp, 2 pour k-means, 3 pour grille régulière (valeur par défaut)

Exemple de ligne de commande :

hfpcfg iris 1

Crée le fichier iris.hfp avec hiérarchie régulière

ou bien

hfpcfg iris 1 iriskm.hfp 2

Crée le fichier iriskm.hfp avec hiérarchie k-means

La sortie créée par défaut est nette (Nature='crisp', avec les opérateurs d'agrégation et de défuzzification 'sum' et 'sugeno' et l'option Classif='no'.

Pour les iris, la sortie est une classe (la variété), donc il est souhaitable d'activer l'option classification en éditant le fichier iriskm.hfp.

(b) programme **hfpvertex** pour calculer les centres des sous-ensembles flous

Arguments :

– fichier de données
– fichier de configuration hfp

Deux arguments optionnels :

-oVertexFile où VertexFile est le nom du fichier contenant les sommets (par défaut : vertices.'type de hiérarchie', soit regular, kmeans ou hfp)

-kn Cette option ne concerne que la hiérarchie hfp. n est le nombre de groupes pour former la partition initiale avec l'algorithme des k-means. Par défaut, la partition initiale est formée par regroupement en valeurs uniques suivant la tolérance indiquée dans le fichier de configuration hfp.

Exemple de ligne de commande :

hfpvertex iris iris.hfp

Crée le fichier vertices.regular, si la hiérarchie indiquée dans iris.hfp est de type grille régulière, vertices.kmeans si elle est de type hfp.

ou bien

hfpvertex iris iriskm.hfp -oiris.sommets

Crée le fichier iris.sommets pour la hiérarchie de type kmeans.

(c) **hfpselect** pour sélectionner le nombre de fonctions d'appartenance (SEF) par variable.

Arguments :

– le fichier de configuration fis
– le fichier de données

Et en options :

-r : choisir wm comme méthode d'induction de règles (par défaut : fpa est utilisée).
-tx : x est la stratégie pour calculer l'ensemble des exemples qui sera utilisé pour initialiser la conclusion, 0 pour MIN, 1 pour DEC (valeur par défaut).
-my : y est le degré de vérité minimum (par défaut : 0.3).
-ez : z est la cardinalité minimum (par défaut : 3)
-oFic où Fic est le nom du fichier contenant le SIF (par défaut 'nom du système'.fis)
-sv : poids cumulé minimum nécessaire pour la génération des règles (0.0 par défaut)
-bc ' : c est le niveau de couverture minimum (par défaut : 1.0, 100 %)
-nf : f est le nombre de SEF initial par variable (par défaut : 1)
-ig : g est le nombre maximum d'itérations (par défaut : 10)
-lFicV : où FicV est le nom du fichier sommets créé par hfpvertex (par défaut : vertices.Hierarchie)
-pNum : Num est le numéro de la sortie (default : 0)
-vFicTest : FicTest est le nom d'un fichier de données utilisée pour la validation (par défaut : le fichier donné en deuxième argument)

Exemple de ligne de commande :

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

Ce programme crée deux fichiers result et result.min. Dans le premier, chacune des lignes correspond à une tentative de complexifier le système, dans le second, seule les configurations gardées à chaque étape ont été conservées.

Rappel des lignes de commandes de l'étape de sélection des partitions :

- *hfpcfg iris 1 iriskm.hfp 2*

- *hfpvertex iris iriskm.hfp*

Positionner le drapeau Classif de la sortie à 'yes' dans le fichier iriskm.hfp.

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

Note : dans le fichier result.min, les champs nommés In1, In2, In3, In4 indiquent le nombre de SEF par entrée.

2 Induction de règles a partir d'un SIF existant

Les méthodes d'induction de règles présentées dans cette section travaillent à partir d'un fichier de configuration SIF. Lorsque celui ci contient des règles, elles sont ignorées.

1. **Méthode FPA**

   Deux programmes C++ sont nécessaires, **genrules** et **fpa**. Dans l'interface, ils sont combinés dans l'option *FPA* (menu *Apprentissage*, sous-menu *Induction*), ou peuvent être exécutés en séquence, d'abord générer les règles, puis générer les conclusions.

   Interface Java :

   – menu *SIF*, Option *Générer les règles*
   Le comportement de cette option est différent selon qu'un fichier de données est ouvert ou non.
   – menu *SIF*, option *Générer conclusions* (*fpa* est la méthode par défaut).

   Lignes de commande

   – programme **genrules** pour générer les règles
   Argument : fichier de configuration du SIF
   Options :
   -fFISFile où FISFile est le nom du fichier de configuration du SIF créé (par défaut config.fis)
   -r pour supprimer les fichiers de travail
   autres options : liées à l'appel avec un fichier de données (pour avoir leur description, lancer genrules sans argument).
   Le fichier de configuration créé contient maintenant, sous la balise règles, le nom du fichier contenant les règles créées avec une conclusion unique, 1.
   Ce programme crée également un fichier de travail info.genere.
   Exemple de ligne de commande :
   genrules iris-sr.fis
   Crée les fichiers config.fis (SIF) et info.gen
   ou bien
   genrules iriskm.fis -firiskmr.fis
   Crée le fichier SIF iriskmr.fis
   autre exemple :
   genrules irisreg.fis -firisregr.fis
   Crée le fichier SIF irisregr.fis
   – Programme **fpa** pour initialiser les conclusions
   Les arguments du programme sont :
   – le fichier de configuration fis
   – le fichier de données
   Et en options :
   -a pour affichage intermédiaire

-fFic où Fic est le nom du fichier contenant les règles initialisées (par défaut config.fpa)
-sx : x est la stratégie pour calculer l'ensemble des exemples qui sera utilisé pour initialiser la conclusion, 0 pour MIN, 1 pour DEC (valeur par défaut).
-dy : y est le degré de vérité minimum pour la génération des règles (par défaut : 0.3).
-ez : z est la cardinalité minimum (par défaut : 3).
-tFicV où FicV est le nom du fichier utilisé pour le calcul de performance (valeur par défaut=fichier de données).
-u : y est le degré de vérité minimum pour le calcul de performance (par défaut : 0.1).
-r pour supprimer les fichiers de travail

La signification de l'ensemble de ces paramètres est commentée dans le guide de l'utilisateur, Menu *Apprentissage*, option *Générer conclusions*. Le nombre de règles retenu dépend beaucoup de ces valeurs.

Exemple de ligne de commande :

fpa iriskmr.fis iris

Crée le fichier configfpa.fis qui contient 26 règles alors qu'il y en avait 81 (3*3*3*3) dans iriskmr.fis.

ou bien

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

qui crée le fichier iriskmrfpa.fis contenant 55 règles.

autre exemple :

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

qui crée le fichier irisregfpa.fis contenant 36 règles.

Ces fichiers de configuration peuvent être utilisés pour inférer.

2. **Wang et Mendel**

Interface Java :

- menu *Apprentissage*, sous-menu *Induction des règles*, option *wm*
  OU
- menu *SIF*, option *Générer conclusions*, cocher la méthode *wm*.

En ligne de commande, programme wm :

Les arguments du programme **wm** sont :

- le fichier de configuration fis
- le fichier de données

Et en options :

-tfile nom du fichier de données utilisé pour le calcul de performance (par défaut l'argument 2)
-oFIS le nom du fichier FIS contenant les règles (par défaut 'nom du système'wm.fis)
-sThresh seuil d'activité pour le calcul de performance (par défaut 0.2)

-l paramètre 'Pas de limite au nombre de valeurs distinctes de la sortie' (par défaut faux)

Exemples de ligne de commande

wm iriskmr.fis iris

Crée le fichier iriswm.fis avec 20 règles

ou bien

wm iriskmr.fis iris -owmiris.fis

qui crée le même fichier appelé wmiris.fis

3. **ols** sera présentée dans la section 3.

4. **Arbre de décision flou**

   Interface Java : Menu *Apprentissage*, sous-menu *Induction des règles*, option *Arbre*.

   En ligne de commande, programme fistree :

   Le programme **fistree** utilise deux arguments obligatoires et propose plusieurs options.

   Les arguments du programme sont :
   – le fichier de configuration fis
   – le fichier de données

   Les options sont les suivantes :

   -oNum où Num est le numéro de la sortie utilisée pour la construction de l'arbre (Par Défaut : 0)

   -sx où x est le seuil d'appartenance minimum pour qu'un exemple soit considéré comme appartenant au nœud pour le calcul de l'entropie (par défaut 0.2)

   -xCard où Card est le nombre minimum requis d'exemples attirés par un noeud pour développer celui-ci (default=min(10,nb exemples/10))

   -ty où y est la tolérance sur le degré d'appartenance à la classe majoritaire (par défaut 0.1)

   -dn où n est la profondeur maximum de l'arbre (par défaut : 0, soit pas de limite)

   -gval où val est le gain relatif minimum en entropie/deviance pour diviser un nœud

   -e0 gain absolu d'entropie, -e1 gain relatif (par défaut, gain absolu)

   -p0 pas d'élagage, -p1 élagage suivant un critère de performance (par défaut, pas d'élagage)

   -lw où w est la perte relative de performance tolérée lors de l'élagage (par défaut 0.0)

   -vFicValid où FicValid est le nom du fichier de validation pour le calcul de la performance à l'élagage (par défaut, fichier de données)

   -a option d'affichage détaillé

   -a0 option d'affichage intermédiaire

   -cCouvMin niveau minimal de couverture à respecter pendant l'élagage

L'induction de l'arbre, comme l'élagage, produit deux fichiers SIF dont les noms sont construits automatiquement. Un premier fichier, d'extension .tree, contient les informations de construction de l'arbre. Il peut être visualisé au moyen de l'interface ou bien à l'aide d'un quelconque éditeur de texte. Le second est le SIF correspondant à l'arbre, il porte l'extension .fis. Il crée aussi les fichiers perf.fistree et result.fistree (voir section 11).

L'arbre est un arbre de régression (critère utilisé=déviance) si la sortie déclarée dans le fichier de configuration du SIF est floue, avec des SEF non discrets. Sinon, c'est un arbre de classification (critère utilisé=entropie), avec des valeurs inférées correspondant à des classes nettes si la sortie déclarée est nette, avec l'option 'classif=yes', ou à des classes floues sinon.

Exemple de ligne de commande :

fistree iriskmr.fis iris -s0.3

Crée iriskmr.fis.sum.tree et iriskmr.fis.tree.fis. Le SIF créé comprend 9 règles.

ou bien

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

Crée, en plus des fichiers précédents, les fichiers décrivant l'arbre élagué iriskmr.fis.prunedsum.tree et iriskmr.fis.prunedtree.fis. Dans ce cas, le nombre de règles est de 34 avant élagage et de 5 après.

5. **HFPFIS**

Pour générer le fichier de configuration SIF qui correspond à une combinaison de SEF sélectionnée, il faut utiliser deux programmes C++, **sethfpfis** et textbfhfpfis. Dans l'interface, ils sont combinés dans une seule option.

Interface Java :

– menu *Apprentissage*, sous-menu *Induction de règles*, option *HFP SIF*

En ligne de commande, programmes sethfpfis et hfpfis :

– **sethfpfis** pour modifier un fichier hfp afin d'obtenir la configuration désirée, utilisable en entrée du programme **hfpfis**.
  Trois arguments sont nécessaires :
  – le nom du fichier hfp d'entrée
  – le nombre de SEF par variable sous forme de chaîne
  – le nom du fichier hfp de sortie
  Exemple de ligne de commande :
  sethfpfis iriskm.hfp "5 2 3 3" irissel.hfp
  Crée le fichier irissel.hfp
– **hfpfis** pour générer le fichier de configuration SIF à partir du fichier de configuration hfp.
  Ses arguments sont ceux de hfpselect (à l'exception des arguments spécifiques à hfpselect relatifs à la recherche itérative), soit :
    le fichier de configuration fis
    le fichier de données
  Et en options :

-r : choisir wm comme méthode d'induction de règles (par défaut : fpa est utilisée).
-tx : x est la stratégie pour calculer l'ensemble des exemples qui sera utilisé pour initialiser la conclusion, 0 pour MIN, 1 pour DEC (valeur par défaut).
-my : y est le degré de vérité minimum (par défaut : 0.3).
-ez : z est la cardinalité minimum (par défaut : 3)
-oOutFic où OutFic est le nom du fichier contenant le SIF (par défaut 'Nom du système'.fis)
-sv : v est le poids cumulé minimum nécessaire pour la génération des règles (0.0 par défaut)
-lFicV : FicV est le nom du fichier sommets créé par hfpvertex (par défaut : vertices.'Hierarchie')
-pNumO : NumO est le numéro de la sortie (par défaut : 0)

Exemple de ligne de commande :
hfpfis iris irissel.hfp -e2 -m0.3 -oirissel.fis -liris.sommets
Crée le fichier irissel.fis comprenant 13 règles générées par la méthode fpa.

3 Génération de partitions et induction de règles par OLS

Le programme **ols** (Orthogonal Least Squares) transforme chaque exemple d'un jeu de données en une règle floue et sélectionne ensuite les plus importantes d'entre elles au sens des moindres carrés, par régression linéaire et orthogonalisation de Gram-Schmidt. Une fois la sélection faite, un deuxième passage de l'algorithme est réalisé au cours duquel les conclusions des règles sont optimisées au sens des moindres carrés. Il est adapté aux problèmes de régression.

Interface Java :
Menu *Apprentissage*, sous-menu *Induction des règles*, option *OLS*.
En ligne de commande, programme ols :
Un seul argument :
– nom du fichier de données
Avec les options :
-nNumOutput
utilisé pour préciser le numéro de la sortie à traiter (par défaut 0=la première sortie)
-oNbOutputs
utilisé pour préciser le nombre de sorties (par défaut une seule sortie)
-j : utiliser l'algorithme original de Hohensohn et Mendel (partition de gaussiennes centrées sur les données)
-tTol : Tol est un paramètre pour la génération des partitions.
2 cas : si algorithme original, Tol est l'écart-type utilisé pour construire les gaussiennes (valeur par défaut=0.05), sinon Tol est le seuil de tolérance pour

déterminer les centres des SEF de la partition floue.
-s : n'impose pas la construction de partitions floues fortes (défaut= partitions floues fortes).
-fFISfile : FISfile est le nom du fichier de configuration du SIF fourni en entrée
Dans ce cas les partitions de ce fichier sont utilisées pour générer les règles. Les règles, si elles existent, sont perdues et seront remplacées par les règles induites, sauf si l'option -x est précisée.
-x : cette option ne fonctionne qu'avec l'option précédente (-fFISfile).
Le programme conserve les partitions et les règles du fichier de configuration du fis, et optimise seulement les conclusions des règles, par moindres carrés, en fonction du fichier de données fourni en 1er argument.
Elle permet donc d'optimiser les conclusions des règles d'un SIF quelconque.
-gTotV : critère d'arrêt, où TotV est la variance non expliquée en valeur relative par rapport à la variance totale (par défaut 0.01)
-qNregles : critère d'arrêt, où Nregles est le nombre maximal de règles sélectionnées (par défaut 10 000).
-v : réduction du vocabulaire de sortie. Par défaut, pas de réduction.
-i : cette option ne fonctionne qu'avec l'option précédente -v, 2 possibilités pour choisir les conclusions des règles (voir section 6)
i=0 : les nouvelles conclusions des règles seront choisies en fonction des données de la sortie du fichier de données
i=1 (valeur par défaut) : les nouvelles conclusions des règles seront choisies en fonction des conclusions des règles actuelles.
-dPerfLoss : PerfLoss est la dégradation relative de performance autorisée lors de la procédure de réduction de vocabulaire (valeur par défaut=0.1)
-mNConc :
NConc est le nombre maximal de conclusions différentes des règles autorisé lors de la procédure de rédcution de vocabulaire (par défaut, pas de limite)
si les 2 paramètres PerfLoss et NConc sont donnés, priorité à NConc.
-l : transformation de la sortie en sortie floue. Possible seulement avec la réduction de vocabulaire.
-pTestFile : TestFile est le nom du fichier de données utilisé pour la seconde passe de l'algorithme, par défaut le même fichier que dans la première passe.
-cOutputFIS : OutputFIS est le nom du fichier de configuration du FIS créé par l'OLS.
-r pour supprimer les fichiers de travail
-btestdatafile : testdatafile est nom du fichier de données utilisé pour calculer la performance du FIS résultat de l'OLS, par défaut le fichier de données.
-eMumin : Mumin est le seuil d'appartenance minimum utilisé pour calculer la performance du FIS résultat, par défaut 0.2
-a : affichage des résultats intermédiaires.

Exemple de ligne de commande :
ols rice

Le programme crée 5 fichiers résultats :

– **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,
```

Autant de lignes que de règles induites.
Signification des colonnes pour chaque ligne :

1. numéro de la règle induite
2. numéro de la ligne du fichier de données ayant servi à construire la règle
3. pourcentage de variance expliquée par la règle
4. pourcentage cumulé de variance expliquée

– **rice.mat** : matrice des degrés d'appartenance des exemples.
– **rice.fis** : fichier de configuration SIF créé par OLS.
– **perf.ols** : fichier de performance du FIS (voir section 11).
– **result.ols** résumé du SIF. Un en-tête précise les champs présents (voir section 11).

4 Optimisation

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

1. *Solis & Wets standard*.
2. *Solis & Wets sur mesure*.

En ligne de commande, programme loopoptims :
Le programme **loopoptims** requiert les arguments suivants :
– le fichier de configuration fis donné en entrée
– le fichier de données
Avec les options suivantes :
-b préfixe des fichiers de sortie (défaut=optim)
-it nombre d'itérations (défaut=100)
-e écart-type du bruit gaussien (défaut=0.005)
-v nombre maximum de contraintes qui peuvent être violées au sein d'une itération
-f nombre maximum d'échecs (défaut=1000)
-d seuil de tolérance d'égalité sur deux centres (défaut=1e-6)
-ns nombre de couples d'échantillons (défaut=10 couples, mettre 0 pour supprimer la validation croisée)
-s n où n est un entier utilisé comme germe pour le générateur de données aléatoires
-in 'x y', chaîne des numéros des entrées à optimiser, tour à tour dans l'ordre indiqué
-r optimiser les conclusions des règles
-o optimiser la sortie si elle est floue
-n numéro de la sortie (0 par défaut)
-mVal où Val est le seuil d'activité pour considérer un exemple comme non blanc (par défaut : 0.2)
-l1x où x est la constante 1 de l'algorithme Solis Wetts
-l2y où y est la constante 2 de l'algorithme Solis Wetts
-l3z où z est la constante 3 de l'algorithme Solis Wetts
-u loops, où loops est le nombre de boucles successives
-c perte relative de couverture admissible (défaut=0.10)
-g créer fichiers intermédiaires
-a affichage détaillé
-wl mode silencieux
-nc distmin pour imposer une distance minimum entre noyaux des sous-ensembles flous voisins (défaut=0.001)

Exemple de ligne de commande :
loopoptims rice.fis rice -ns 2 -s 102 -b optimfis
qui optimise les conclusions des règles du fichier rice.fis, en validation croisée sur 2 paires apprentissage (75%)-test (25 %), et enregistre les SIF optimisés pour

chaque échantillon d'apprentissage dans les fichiers optimfis-lrn.sample0-final.fis et optimfis-lrn.sample1-final.fis. De plus le fichier med.fis est construit à partir des paramètres médians des SIF optimisés.

ou bien

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

qui optimise les SEF des entrées 1 et 3, ainsi que les conclusions des règles du fichier rice.fis. Les fichiers créés portent le même nom que précédemment.

Un fichier perf.res est créé, contenant les performances du SIF initial, de chaque SIF optimisé, du SIF médian, calculées sur le jeu de données initial, sur chaque échantillon de test et sur chaque échantillon d'apprentissage.

5 Simplification

Cette étape a pour objet de regrouper des règles, d'éliminer des règles ou des variables dans une règle.

Interface Java :

Menu *Apprentissage*, option *Simplification*

En ligne de commande, programme fisimple :

Les arguments du programme **fisimple** sont les suivants :

– nom du fichier de config du FIS
– nom du fichier de données

Avec les options :

-t nom du fichier utilisé pour le calcul de la performance (défaut : fichier de données)

-pNum : Num est le numéro de la sortie (par défaut : 0)

-bBlMax : BlMax est le pourcentage maximum d'exemples blancs toléré (par défaut : 0.1)

-cCovLoss : CovLoss est la perte relative de couverture tolérée (par défaut : 0.1)

-dErr : Err est l'erreur absolue maximale admissible (par défaut 0.1)

-ePerfLoss : perte relative de performance tolérée (par défaut 0.1 pour 10%)

-hHomog : seuil d'homogénéité de la sortie (par défaut 0.8 en classification, pas de limite sinon)

-k : ne pas conserver la dernière règle qui a pour sortie une classe ou un SEF donné (par défaut la dernière règle est conservée)

-r : supprimer les fichiers de travail

-a : affichage intermédiaire

-q : phase de suppression des règles

-l : phase de suppression des variables

-tFicTest où FicTest est le nom d'un fichier de données utilisé pour les calculs de performance (par défaut : le même fichier de données)

-sMuMin : MuMin est le degré d'appartenance minimum (par défaut : 0.5). Il est utilisé uniquement pour le calcul de performance.

Remarques : les arguments -b et -c sont exclusifs, et -b a la priorité sur -c. Les arguments -d et -e sont exclusifs, et -e a la priorité sur -d.

Exemple de ligne de commande :

fisimple wmiris.fis iris

Le programme affiche *"The most simple FIS : wmiris.fis.jb.2*.

L'ensemble des configurations simplifiées se trouve dans le fichier result.simple. Le SIF le plus simple est wmiris.fis.jb.2 qui contient onze règles.

ou encore

fisimple iriskmr.fis.tree.fis iris

Le SIF le plus simple est alors iriskmr.fis.tree.fis.jb.2 qui contient dix sept règles.

fisimple rice.fis rice -q -l

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

6 Réduction du vocabulaire de sortie

Cette option permet de réduire le nombre de conclusions distinctes des règles dans un SIF.

Elle peut être lancée à la suite d'une procédure d'apprentissage, telle que l'OLS ou FPA, qui génère a priori autant de conclusions différentes que de règles.

Il faut indiquer à partir de quelles valeurs la réduction doit s'opérer. Deux choix sont possibles : soit les conclusions des règles existantes, soit les valeurs de la variable de sortie dans le fichier de données. La réduction consiste en une opération de clustering des conclusions ou des valeurs de sortie pour obtenir les valeurs finales des conclusions.

L'utilisateur peut fixer le nombre de valeurs désiré ou bien spécifier une perte de performance tolérée. En effet, la réduction de vocabulaire s'accompagne en général d'une perte de précision.

Interface Java :

Menu *SIF*, option *Réduire le vocabulaire de sortie*.

En ligne de commande, programme vocreduc :

Argument :

- le nom du fichier de configuration du SIF
- le nom du fichier de données

Options :

-oNumOutput

utilisé pour préciser le numéro de la sortie à traiter (par défaut 0=la première sortie)

-dType

- -d1 : les conclusions sont choisies parmi les conclusions des règles existantes (valeur par défaut)
- -d0 : les conclusions sont choisies parmi les valeurs de sortie dans le fichier de données

-lPerfLoss : PerfLoss est la perte relative de performance autorisée lors de la réduction de vocabulaire. Dans ce cas Conc est déterminé par le programme.
-cConc : Conc est le nombre de conclusions différentes utilisées. Dans ce cas PerfLoss est déterminé par le programme.
Remarque : la valeur par défaut de ces 2 arguments est 0.1 pour PerfLoss, et détermination automatique de Conc.
-sVal Val est le seuil d'activité pour considérer un exemple comme non blanc (par défaut : 0.2)
-a : affichage détaillé

Exemple de ligne de commande
vocreduc rice.fis rice
Le nombre de conclusions passe de 25 à 6.

7 Liens

Cette option permet de visualiser les liens, en termes de degré d'activation, qui existent entre les exemples et les règles.

Interface Java :
Menu *Données*, option *Liens*.
Les fichiers sont générés et visualisés dans des fenêtres texte.

En ligne de commande :
Argument du programme **fislinks** :
– le nom du fichier de données

Exemple de ligne de commande :
fislinks iris.fis iris
Fichiers crées :
leur nom est préfixé par défaut par celui du fichier de données.

– rules.items
  Ce fichier compte autant de lignes que de règles, plus 2.
  * Première ligne : nombre de règles
  * Deuxième ligne : nombre maximal d'exemples qui activent une règle
  * Troisième ligne : description correspondant à la première règle, soit dans l'ordre : numéro de la règle (1 pour la première), poids cumulé, nombre d'exemples qui activent la règle (au-delà d'un seuil paramétré, 1e-6 par défaut) suivi de leurs numéros.
– items.rules
  Ce fichier compte autant de lignes que d'exemples.
  Pour chaque ligne : Numéro de l'exemple (1 pour le premier), puis les numéros des règles qu'il active.
– rules.links
  La notion de liens entre règles est utile pour apprécier la cohérence d'une base de règles. Si deux règles sont fortement liées, et que leurs conclusions

sont différentes, alors l'incohérence provient du fait que les domaines d'entrée couverts par les règles ne sont pas suffisamment spécifiques. Cette situation peut aussi correspondre à la présence d'une exception dans la base d'apprentissage. L'expression mathématique du niveau de lien de la règle *i* avec la règle *j* est :

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

N_i représente le cardinal de l'ensemble E_i des exemples qui activent la règle *i*, $N_{i,j}$ est le cardinal de $E_i \cap E_j$.

Ce fichier se présente sous la forme d'une matrice carrée de la taille du nombre de règles. Chaque cellule indique le niveau de lien correspondant. Il convient de noter que cette matrice n'est pas symétrique.

– rules.sorted
Si l'option correspondante est activée, les règles sont triées suivant leur poids cumulé, c'est-à-dire leur influence dans la base d'exemples.
Remarque : les liens ne dépendent pas des données de sortie du fichier de données

8 Génération d'échantillons d'apprentissage et de test

Cette option permet de générer à partir d'un jeu de données des couples d'échantillons d'apprentissage et de test.

Interface Java :

Menu *Données*, option *Générer échantillons*.

Les fichiers générés se trouvent dans le répertoire de travail de FisPro, ou dans le répertoire bin de l'installation, selon le système d'exploitation.

En ligne de commande :

Argument du programme **sample** :

– le nom du fichier de données

Options :

-nNs où Ns est le nombre de couples d'échantillons (par défaut : 1 partage le jeu de données en 1 jeu d'apprentissage et 1 jeu de test)

-pApp où App est la fraction du jeu de données utilisée pour le jeu d'apprentissage (par défaut : 0.75)

-sSeed
il faut donner la même valeur de Seed pour reproduire le même tirage aléatoire, donc le même couple d'échantillons
(par défaut, 0 donne un nouveau tirage à chaque appel)

-c pour créer des échantillons qui respectent la proportion des classes du fichier de données

-oNumC : utilisé avec l'option -c, pour préciser le numéro de la colonne du fichier de données, utilisée pour affecter les classes (par défaut, dernière colonne)

-eTol : utilisé avec l'option -c, Tol=précision

Le paramètre Tol n'a de sens qu'avec l'option -c (échantillon de classification). Dans le cas où les classes de la sortie ne sont pas des entiers, mais des nombres réels $c_1, c_2 \ldots c_k$, les classes seront construites en regroupant les nombres $c_i \ldots c_j$ tels que $abs(c_i - c_j) <= Tol$).

-a pour affichage détaillé

Exemple de ligne de commande

sample iris -n4 -c

génère à partir du fichier iris 4 paires de fichiers :

iris.lrn.sample.0, iris.lrn.sample.1, ... iris.tst.sample.0, iris.tst.sample.1 ... qui respectent les proportions des classes (variétés 1,2,3) dans le fichier iris.

Les fichiers iris.lrn.sample.0 et iris.tst.sample.0 constituent un premier couple apprentissage-test, et ainsi de suite.

9 Performance

Calcul de la performance d'un SIF sur un jeu de données.

Interface Java :

Menu *Données*, option *Inférer*.

En ligne de commande :

Programme **perf**, arguments :

– le nom du fichier de config du fis

– le nom du fichier de données

Options :

-nNum où Num est le numéro de la sortie (par défaut : 0)

-sVal Val est le seuil d'activité pour considérer un exemple comme non blanc (par défaut : 0.2)

-a pour l'affichage.

Exemple de ligne de commande :

perf irissel.fis iris

renvoie la valeur 4 (exemples mal classés)

ou encore

perf riceoptrules.fis rice

renvoie 0.004819 (indice basé sur l'erreur quadratique)

Perf crée le fichier perf.res, dont la structure est décrite dans la documentation de l'utilisateur.

10 Vérification d'un fichier de données

Le programme **readdata** lit un fichier de données et détecte automatiquement le délimiteur de champs.

Argument :

– le nom du fichier de données

Résultat :

Le programme indique le nombre de lignes et de colonnes, et le nom des variables si le fichier a une ligne d'en-tête.

11 Fichiers résultats des programmes d'apprentissage

Les procédures d'apprentissage ci-dessous créent systématiquement 2 fichiers : *result.nomproc* et *perf.nomproc*. Le tableau donne la correspondance de noms entre les procédures et les fichiers crées.

nom du programme	fichier résultat	fichier performance
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

Fichier résumé

Le résultat d'une procédure d'apprentissage est résumé dans un fichier, *result.nomproc*. Ce fichier ne contient qu'une ligne qui décrit le système généré par la procédure :

1ère colonne : nom de fichier SIF initial

2ème colonne : indice de performance

3ème colonne : indice de couverture

4ème colonne : erreur max

colonnes suivantes : les caractéristiques de la base de règles

Caractéristiques d'une base de règles

– maxR : nombre maximal de règles possibles en fonction des partitions des entrées
– nR : nombre de règles
– maxVr : nombre maximum de variables dans une règle
– meanVr : nombre moyen de variables par règle
– nVar : nombre de variables distinctes utilisées par les règles

– meanMF : nombre moyen de MF par variable utilisée
– nClass : nombre de classes ou de MF en sortie
– nRc : nombre de règles par classe ou par MF

Fichier performance

Format général :

La première ligne de ce fichier indique le label des colonnes, les étiquettes possibles sont définies dans le fichier fis.h :

– "OBS" : Valeur de la sortie observée, décrite dans le fichier de données
– "INF" : Valeur de la sortie inférée par le SIF
– "Al" : Alarme déclenchée lors de l'inférence (voir ci dessous)
– "ClINF" : Classe inférée pour les sorties nettes avec option classification
– "CLAl" : Alarme déclenchée lors de l'inférence de la classe (voir ci dessous)
– "Err" : La différence entre la sortie inférée et la sortie observée
– "Bl" : Vaut 1 si l'exemple est inactif, 0 sinon
– "CErr2" : Le carré de l'erreur cumulée sur l'ensemble des exemples précédents

ATTENTION : L'erreur cumulée ne tient pas compte des exemples inactifs.

La première colonne est la sortie observée, si elle est disponible dans le fichier de données, car l'inférence peut aussi être faite en l'absence de sortie observée. Suivent des colonnes en nombre variable, ce nombre dépendant du type de sortie :

Sortie	Classif	Défuzzification	Nb. champs	Champs présents			
nette	oui	sugeno	4	V. inférée	Alarme	Classe inférée	Alarme
nette	non	sugeno	2	V. inférée	Alarme		
nette	oui	MaxCrisp	2	V. inférée	Alarme	$\mu_1, \mu_2 \ldots \mu_n$	
nette	non	MaxCrisp	2	V. inférée	Alarme		
floue	oui	sugeno ou area	n+2	V. inférée	Alarme	$\mu_1, \mu_2 \ldots \mu_n$	
floue	non	sugeno ou area	2	V. inférée	Alarme		
floue	oui	MeanMax	n+2	V. inférée	Alarme	$\mu_1, \mu_2 \ldots \mu_n$	
floue	non	MeanMax	2	V. inférée	Alarme		

La colonne suivante indique, lorsque la valeur observée est disponible, l'écart entre celle-ci et la valeur inférée.

Remarque : $\mu_1, \mu_2 \ldots \mu_n$ ne sont présents que si l'exemple active au moins une règle. Pour une sortie floue avec opérateurs *sugeno* ou *area*, ce sont les degrés d'appartenance de la valeur inférée aux SEF de sortie numéro $1, 2 \ldots n$. Dans le cas d'une sortie floue avec opérateur *MeanMax* ou d'une sortie nette avec *MaxCrisp* ils représentent le niveau d'activation de chacune des sorties possibles, SEF pour la sortie floue ou bien valeurs réelles pour la sortie nette.

La colonne *Bl* indique si l'exemple est inactif (valeur 1 dans ce cas, sinon 0). Enfin, la dernière colonne contient la valeur courante de l'indice de performance.

Valeurs possibles pour l'alarme :
valeurs entières qui dépendent du type d'opérateur de défuzzification.

- NOTHING (Valeur 0) : Tous types. Tout va bien, rien à signaler.
- NO_ACTIVE_RULE (Valeur 1) : Tous types. L'exemple d'entrée n'a activé aucune des règles de la base. L'activation est ici prise au sens strict ($\mu > 0$), et ne tient pas compte du seuil d'activation pour les exemples inactifs (colonne *Bl*).
- AMBIGUITY (Valeur 2) : SugenoClassif (Sortie nette) et MaxCrisp. La différence entre les deux classes les plus possibles est inférieure au seuil (valeur par défaut AMBIGU_THRES = 0.1).
- NON_CONNEX_AREA (Valeur 3) : WeArea - Est declenchée lorsque l'aire définie dans l'espace de sortie n'est pas connexe. Seuls sont considérés les SEF dont le niveau d'activation est supérieur au seuil (valeur par défaut MIN_THRES = 0.1).
- NON_CONNEX_SEGMENT (Valeur 4) : MeanMax - Se produit lorsque le segment des maxima n'est pas connexe. Deux niveaux sont considérés comme égaux, si leur différence est inférieure au seuil (valeur par défaut EQUALITY_THRES = 0.1).