

Prototypage flou

Fuzzy prototyping

L. Orseau P.Y. Glorennec

INSA de RENNES/ IRISA

Département Informatique, 35 043 RENNES Cedex

lorseau@irisa.fr

glorenne@irisa.fr

Résumé :

Une méthode de classification basée sur des prototypes à structure variable est proposée. L'algorithme d'apprentissage permet d'extraire les propriétés intrinsèques d'une classe et peut être utilisé pour la sélection de variable.

Mots-Clés :

Classification, prototypes flous, sélection de variable, apprentissage supervisé.

Abstract:

A classification method based on prototypes with variable structure is proposed. The training algorithm makes it possible to extract the intrinsic properties from a class and can be used for variable selection.

Keywords:

Classification, fuzzy prototypes, variable selection, supervised learning.

1 Introduction

Le problème général de la classification peut se résumer comme suit : étant donné un espace \mathcal{E} à N dimensions, trouver une application f de \mathcal{E} dans un espace discret $C = \{c_1, \dots, c_K\}$. Il existe de très nombreuses méthodes dont beaucoup sont basées sur la notion de prototype, [4, 5]. Un prototype est un vecteur de \mathcal{E} qui est représentatif d'une classe. Dans l'apprentissage compétitif, par exemple, les prototypes sont construits progressivement à partir de la définition d'une distance.

Dans cet article, l'appartenance d'un prototype à l'espace \mathcal{E} n'est plus nécessaire : les prototypes vont comporter de une à N dimensions,

selon les classes. Ceci va permettre de mieux mettre en évidence les propriétés intrinsèques de chaque classe en éliminant les variables qui n'apporteraient que peu d'information.

L'objectif de ces prototypes flous est de devenir la couche de base d'un réseau de neurones comportant un symbolisme interne. Ils ont une certaine similarité avec les réseaux à base radiale [8], mais possèdent d'avantage de paramètres : des trapèzes sont utilisés à la place de rayons, leur conférant une zone utile pouvant être tout aussi locale mais aussi plus globale. En effet, les trapèzes possèdent un sommet de longueur variable, ce qui est intéressant, notamment au niveau de la sélection de variables. Les prototypes flous sont par ailleurs moins sensibles au nombre de variables.

D'autres méthodes de sélection de variables existent, comme par exemple les algorithmes génétiques [9], les arbres de décision flous [7], la construction de règles incomplètes [3], les C moyennes floues [1], ...

L'algorithme présenté utilise une méthode génétique simple pour effectuer la sélection (et non pas l'optimisation du placement des trapèzes). Les arbres de décision flous, quant à eux, diffèrent par le fait que les règles possèdent obligatoirement une ou plusieurs variables en commun. Ceci est un avantage certain pour l'interprétabilité, mais c'est aussi une restriction liée aux besoins de lisibilité. La construction de règles incomplètes est une des méthodes

les plus abouties dans le domaine de l'interprétabilité, et notre méthode n'apporte rien de plus de ce côté-ci. Il existe aussi un algorithme possibiliste efficace, basé sur les C moyennes floues [6], mais il nécessite beaucoup de ressources calculatoires.

L'interprétabilité n'était donc pas notre but premier, mais cette méthode permet néanmoins d'effectuer une sélection de variables et d'extraire facilement de la connaissance .

Un algorithme d'apprentissage original et calculatoirement simple est proposé et est testé sur un problème industriel.

2 Prototypes flous

Un prototype flou (cf fig. 1) englobe une zone de l'espace d'entrée et représente une unique classe. C'est un hyper-trapézoïde dont chaque trapèze, considéré comme une fonction d'appartenance, recouvre tout ou partie du domaine de variation d'une variable. Il possède initialement un trapèze par variable et ce nombre sera ensuite souvent réduit par la sélection de variables.

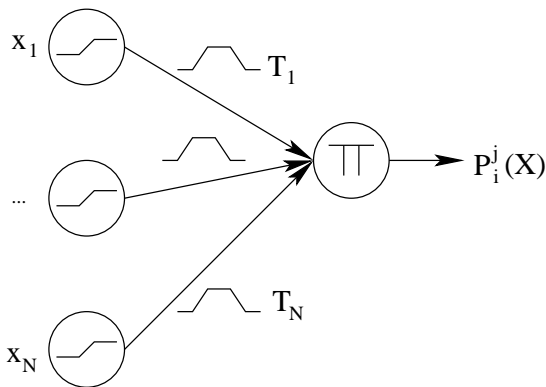


Figure 1 – Un prototype flou : la donnée passe par une phase de normalisation-bornage, puis dans les trapèzes d'appartenance, dont on fait enfin le produit.

Au préalable, le vecteur d'entrée $X = (x_1, \dots, x_N)^t$ passe par une phase de normalisation-bornage. C'est le rôle de la fonction de saturation à pente linéaire, à valeur dans $[0 ; 1]$. C'est à l'utilisateur de fixer les bornes *min* et *max* et il n'y a pas d'optimisation

de celles-ci.

L'appartenance à la classe est ensuite calculée par le produit de ces fonctions d'appartenance. On définit le degré d'appartenance $P_i^j(X)$ du vecteur X à la classe i représentée par le prototype j :

$$T_n(x_n) \in [0 ; 1], P_i^j(X) \in [0 ; 1]$$

$$P_i^j(X) = \prod_{n=0}^N T_n(x_n) \quad (1)$$

$T_n(x_n)$ étant le degré d'activation du trapèze T_n par la variable x_n .

Il n'y a pas de partitionnement du domaine d'entrée, donc les zones qu'englobent les différents prototypes peuvent se chevaucher, qu'ils soient de la même classe ou non.

Pour déterminer la classe à laquelle le système suppose que la donnée appartient, la solution la plus simple serait de choisir le prototype dont le degré d'appartenance est le plus fort. Cela pose un certain nombre de problèmes, car, par exemple, plusieurs prototypes peuvent avoir un degré d'appartenance de 1. La raison principale est détaillée plus loin.

On somme donc les degrés d'appartenance de tous les prototypes d'une même classe. À ce niveau là, on ne peut plus parler de degré d'appartenance, car la somme peut dépasser 1. Aussi utiliserons-nous le terme d'activation.

La réponse du système est la classe de plus forte activation.

3 Apprentissage

Initialement, le système ne comporte qu'un unique prototype par classe.

C'est à l'utilisateur de proposer au système d'ajouter un prototype, et celui-ci sera inséré automatiquement lorsque ceux existant sont

jugés trop loin de la donnée actuelle, selon une notion de distance définie en 3.1.

Nous définissons ensuite un facteur d'exclusion, permettant de restreindre la portée des prototypes aux zones les plus représentatives de la classe considérée. Ainsi il est possible d'ajouter un prototype qui modélisera rapidement les autres zones mal modélisées, car l'erreur y est plus grande.

L'apprentissage est supervisé et se fait donnée par donnée.

On pose $A_i(X) = 1$ si X appartient à la classe i , 0 sinon.

Soit $C_i(X)$ la somme des activations des prototypes P_i^j de la classe i pour le vecteur d'entrée X :

$$C_i(X) = \sum_j P_i^j(X) \quad (2)$$

L'erreur sur la prédiction E_i pour la classe i est :

$$E_i(X) = A_i(X) - C_i(X) \quad (3)$$

Définie ainsi, le domaine théorique de E_i est $[-\infty ; 1]$. Si l'on avait choisit le maximum d'activation d'un prototype comme critère de décision, l'apprentissage de chaque prototype aurait été indépendant des autres, sans prise en compte de ce qu'ils modélisent : chacun aurait dû représenter au mieux la classe considérée et chacun se serait positionné sur la zone de l'espace d'entrée où les données sont les plus représentatives de la classe. Ce qui pose alors problème pour recouvrir les autres zones. Mais l'optimisation va faire tendre l'erreur définie par l'équation 3 vers 0, ayant pour conséquence de limiter les zones de chevauchement des prototypes. E_i prendra alors peu de valeurs autres que dans $[-1 ; 1]$.

Il faut donc que l'apprentissage d'un prototype se fasse en fonction des autres : si un prototype modélise correctement une zone de l'espace d'entrée, cette zone ne doit plus être considérée comme étant à modéliser.

C'est pourquoi on répartit l'erreur de prédiction E_i sur tous les prototypes concernés, c'est à dire tous ceux qui sont activés. On définit l'erreur E_i^j du prototype j de la classe i en fonction de sa propre activation, considérée ici comme une contribution plus ou moins forte à la décision :

$$E_i^j(X) = E_i(X) \frac{P_i^j(X)}{C_i(X)} \quad (4)$$

Ainsi, $C_i(X)$ va tendre vers 1 si X appartient à la classe i , quelque soit le nombre de prototypes concernés.

3.1 Optimisation d'un prototype

Avec cette erreur $E_i^j(X)$, on effectue une modification sur chacun des trapèzes d'un même prototype. Les deux points du trapèze qui sont affectés sont ceux qui encadrent la valeur de la variable correspondante. Ils se décalent ensuite proportionnellement à la proximité de la donnée X et à l'erreur commise, de manière à réduire celle-ci.

Au fur et à mesure, les quatre points du trapèze vont donc se placer de sorte à avoir un équilibre entre erreurs positives (la prédiction est inférieure à la réalité : $E_i(X) > 0$) et négatives.

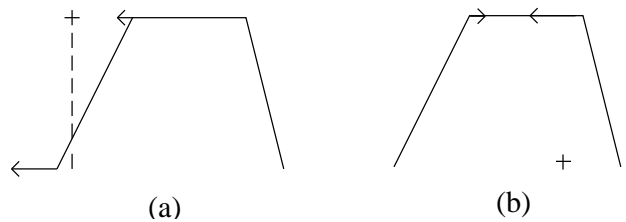


Figure 2 – Modifications d'un trapèze d'un prototype P_i^j en fonction d'une entrée x_n (+) correspondante. (a) $E_i^j(X) > 0$, X est de la même classe que P_i^j ; (b) $E_i^j(X) < 0$, X n'est pas de la même classe que P_i^j .

Lorsqu'une donnée n'active aucun prototype, l'optimisation porte sur le plus proche voisin, selon la notion de distance entre le vecteur X et un prototype P :

$$D(X, P) = 1 - \prod_{n=0}^N (1 - Dist(x_n, T_n)) \quad (5)$$

avec

$$Dist(x_n, T_n) = \begin{cases} 0 & \text{si } x_n \in T_n, \\ T_n^{inf} - x_n & \text{si } x_n < T_n^{inf}, \\ x_n - T_n^{sup} & \text{si } x_n > T_n^{sup} \end{cases} \quad (6)$$

T_n^{inf} et T_n^{sup} représentant les bornes inférieures et supérieures du trapèze T_n (du prototype P) correspondant à la variable x_n .

Grâce à la normalisation (cf. fig. 1), cette notion de distance nous certifie d'avoir une valeur dans $[0; 1]$. Une distance nulle signifie que l'on se trouve à l'intérieur du prototype, auquel cas on ne se sert pas du plus proche voisin. Pour obtenir une distance de 1 (donc un produit nul), il faudrait qu'un des trapèzes soit réduit à un point sur l'une des bornes du domaine d'entrée, et que la valeur de la variable soit exactement la borne opposée, ce qui très improbable : on voit donc que les valeurs proches de 1 sont beaucoup moins probables que celles proches de 0.

L'erreur E_i^j utilisée pour l'optimisation des trapèzes est alors multipliée par $(1 - D(X, P))$.

3.2 Facteur d'exclusion

Un facteur d'exclusion est défini dans $]0; 1[$ permettant de prendre en compte plus fortement les erreurs soit positives (cf. fig. 3), soit négatives (cf. fig. 4). On redéfinit ainsi l'erreur d'un prototype, utilisée pour l'optimisation des trapèzes :

$$\begin{aligned} &\text{si } (E_i^j(X) > 0) \text{ alors} \\ & \quad Exc_l_i^j(X) = E_i^j(X) \cdot (1 - EXCL); \\ &\text{sinon} \\ & \quad Exc_l_i^j(X) = E_i^j(X) \cdot EXCL; \end{aligned}$$

$EXCL$ étant la valeur du facteur d'exclusion. On peut ainsi, par exemple, forcer à n'englober presque que des exemples positifs, avec le

moins d'erreurs possible, quitte à laisser plus de données de côté. On tente en fait de déplacer l'équilibre de l'erreur moyenne commise par le prototype. Elle sera nulle pour une valeur d'exclusion de 0.5, positive pour une valeur supérieure, et négative sinon.

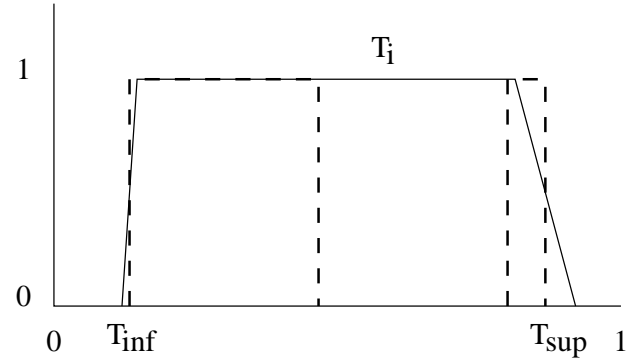


Figure 3 – Facteur d'exclusion à 0.1; en pointillés : une des variables que l'on cherche à modéliser (exemples positifs d'une même classe); en trait continu : le trapèze correspondant obtenu. Les zones sans pointillés sont considérées comme des exemples négatifs.

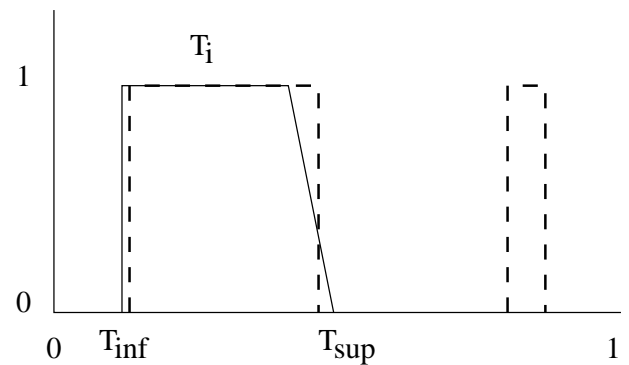


Figure 4 – Facteur d'exclusion à 0.9; en pointillés : une des variables que l'on cherche à modéliser (exemples positifs d'une même classe); en trait continu : le trapèze correspondant obtenu. Les zones sans pointillés sont considérées comme des exemples négatifs.

Sur les figures 3 et 4, le vecteur d'entrée est réduit à une dimension.

Le fait que le trapèze de la figure 4 ne soit pas vertical sur le bord droit est expliqué par l'attirance des données de droite via le plus proche voisin. Pour la figure 3, ce sont les exemples négatifs compris entre les deux rectangles qui fléchissent le trapèze.

Le facteur d'exclusion ne doit pas atteindre les bornes 0 et 1. En effet, en 1, l'algorithme d'optimisation va exclure *tous* les exemples négatifs sans se préoccuper des positifs. On risque donc d'obtenir un trapèze bien trop réduit, qui n'engloberait pas complètement un groupe d'exemples uniquement positifs. De même, en 0, en ne prenant en compte que les exemples positifs, le trapèze produit ne serait pas capable d'exclure des exemples négatifs, même si ceux-ci se trouvent sur les bords du trapèze.

Il est donc nécessaire de toujours laisser une marge, même faible, permettant la modification dans les deux sens : agrandissement *et* rétrécissement du trapèze.

Il semble que la meilleure technique consiste à prendre une exclusion initialement très faible, et à l'augmenter au fur et à mesure de l'apprentissage, ayant pour effet d'englober d'abord le maximum de données non encore modélisées, puis de se restreindre à une des zones restantes les plus représentatives de la classe considérée. Sur la figure 4, on voit qu'une zone n'est pas modélisée, car le facteur d'exclusion est fort et le prototype exclut donc une partie des données, de manière à engendrer moins d'erreurs négatives. Si l'on propose d'effectuer un ajout, le nouveau prototype va mettre très peu de temps à modéliser le rectangle de droite. De plus, l'ancien prototype va aussi continuer à s'optimiser, mais cette fois sans tenir compte de l'attraction de la deuxième zone par le plus proche voisin.

Avec un facteur d'exclusion faible (cf fig. 3), le prototype doit engendrer moins d'erreurs positives et donc inclure les deux zones pointillées. L'ajout d'un prototype est plus délicat car les deux zones sont déjà modélisées. Le prototype existant devrait alors subir des modifications plus importantes.

4 Sélection des variables

Une fois que l'utilisateur considère que l'erreur moyenne commise est suffisamment faible, il peut passer à une phase de sélection, de manière

à ne garder uniquement les variables dignes d'intérêt.

Distinguons deux types de variables inutiles :

- les variables de *bruit*, qui troublent l'apprentissage et empêchent la plupart du temps d'obtenir des résultats meilleurs,
- les variables *redondantes*, qui n'apportent donc pas d'information supplémentaire, ralentissent un peu l'apprentissage (moins que les variables de bruit), mais ne modifient pas la qualité des résultats.

Il serait donc intéressant de se séparer de ces variables par un procédé automatique.

Nous décrivons par la suite deux modes de sélection :

- en interne, c'est à dire que c'est l'algorithme d'apprentissage qui va faire en sorte que ces variables n'aient plus d'influence sur le résultat,
- en externe, où l'on permet à l'utilisateur de connaître ces variables et de les enlever.

4.1 Sélection interne

Le fait que l'on ne partitionne pas le domaine d'entrée ainsi que la méthode d'apprentissage permettent une sélection interne des variables.

Les trapèzes des variables de bruit vont avoir tendance à s'élargir pour finalement englober tout le domaine d'entrée, sous condition d'avoir suffisamment de données.

Pour le cas des variables redondantes, c'est un peu différent. Supposons que deux entrées sont identiques, et prennent donc pour chaque vecteur X les mêmes valeurs. Dans ce cas, l'optimisation va modifier les deux trapèzes exactement de la même manière.

Supposons maintenant qu'une des deux variables est plus bruitée que l'autre. Le trapèze correspondant à la plus utile des deux va se placer correctement, mais comme on effectue un produit entre les trapèzes, ce qui équivaut à un *ET*, une fois que ce trapèze sera bien positionné, l'autre n'aura plus de nécessité d'être modifié et fera donc une modélisation

moins bonne. Néanmoins, cela ne gêne pas les résultats.

Intrinsèquement, l'algorithme fait en sorte que les variables de bruit ne gênent pas les résultats, sans pour autant être capable de *discerner* l'utilité des variables. Il ne fait aucune distinction entre elles, qu'elles soient utiles, bruitées, ou redondantes, et il n'est par conséquent pas possible de *supprimer* en interne celles qui ne nous intéressent pas.

4.2 Sélection externe

Il appartient à l'utilisateur de trouver un critère permettant de définir la ligne de séparation entre les variables que l'on souhaite évincer et celles que l'on juge utiles (voir l'exemple dans la section suivante). Il doit favoriser les classifications correctes et pénaliser les erreurs. De manière à restreindre le nombre de variables utilisées, il est aussi nécessaire de favoriser le retrait d'une d'entre elle, ou de pénaliser le système proportionnellement à ce nombre.

Ce critère sera donc la base de l'algorithme de sélection externe.

A chaque trapèze, on attribue un booléen Actif, qui, lorsqu'il est à vrai délivre la réponse habituelle du trapèze, mais lorsqu'il est à faux, délivre constamment 1, comme si la variable n'était pas connectée au prototype, ou que le trapèze englobait tout le domaine d'entrée. Par ailleurs, il faut que le prototype réponde 0 au lieu de 1 dans le cas où tous les trapèzes sont désactivés.

Le booléen Actif représente donc l'utilité de la variable pour un prototype donné.

On peut ensuite utiliser un algorithme génétique, mais une succession de modifications sur un unique agent semble largement suffisante.

Cet algorithme va uniquement servir à déterminer quelles sont les variables jugées utiles, et non à optimiser le placement des prototypes. L'agent se compose des booléens

Actif. Au début, ils sont tous à vrai, car toutes les variables sont utiles *a priori*.

On garde en mémoire le meilleur agent. On l'altère légèrement à chaque génération, en inversant la valeur d'un nombre restreint des booléens Actif. Le critère nous donne ensuite la valeur de ce nouvel agent, potentiellement meilleur, auquel cas on le garde en mémoire à la place de l'ancien. On recommence jusqu'à ce qu'il se soit passé suffisamment de générations sans qu'aucune amélioration n'ait été détectée.

À la fin de l'algorithme, seules les variables dont Actif est vrai sont considérées comme utiles. Si trop d'information a été perdue, c'est qu'il faut améliorer le critère.

Une fois la sélection automatique effectuée, il est préférable de continuer quelque peu l'optimisation du système, étant donné que les variables que l'on vient d'enlever gênaient peut être les résultats, empêchant les trapèzes des variables restantes de mieux se positionner.

5 Application à la détection de bactéries

L'appareil de la société X. cherche à détecter des bactéries, mais, par précaution, provoque de nombreuses fausses alarmes : un élément détecté comme bactérie peut se révéler être autre chose. Les éléments détectés appartiennent à cinq catégories : les particules, les bactéries et des éléments notés de type 1, 2 ou 3.

Une observation est classée comme particule si elle échoue à un ou plusieurs tests sur une série de 10 ou 12 tests. Dans le cas contraire, l'appareil diagnostique une bactérie. Cette méthode de classification nécessite une analyse systématique au microscope pour déterminer la classe réelle de l'élément observé.

Dans le fichier à analyser, 3152 observations ont été diagnostiquées comme étant des bactéries, mais l'examen au microscope a permis de

déterminer :

- 2215 vraies détections de bactéries, soit un taux de succès de 70%,
- 264 particules,
- 568 éléments de type 1,
- 105 éléments de type 3.

Aucune observation ne correspond à un élément de type 2, dont la détection ne sera donc pas abordée dans cette étude.

L'objectif de l'étude est double :

- réduire drastiquement le nombre de fausses alarmes (appelés aussi les "faux positifs"),
- limiter le recours au microscope.

Dans l'approche présentée [2], seul un sous-problème est traité : la caractérisation d'une bactérie avec un coefficient de confiance (ou coefficient d'appartenance à la catégorie des bactéries). Un système complet, capable de traiter les quatre cas possibles, pourrait être développé selon la même méthodologie.

5.1 Utilisation d'un réseau de neurones

Une première modélisation, avec un réseau de neurones (RN) (perceptron multicouches avec 5 neurones dans la couche cachée), a tenté de séparer les bactéries des autres composants (particules, éléments de type 1 et 3). Pour cela, deux fichiers ont été extraits aléatoirement du fichier initial, l'un, de 934 observations, pour l'apprentissage, l'autre, de 442 observations, pour tester les capacités de généralisation du RN obtenu. Les données ont été séparées en deux classes :

- la classe 1, pour les bactéries,
- la classe 0, pour les autres composants.

La répartition dans les ensembles d'apprentissage, de test et total sont de l'ordre de 70% de bactéries et 30% d'autres types. En prenant des seuils de $\frac{1}{3}$ et $\frac{2}{3}$, on définit les faux positifs, les faux négatifs et les indécisions de la façon suivante :

- un faux négatif (FN) correspond à une

réponse neuronale inférieure à $\frac{1}{3}$ alors que la réponse attendue est 1 (le RN ne confirme pas qu'il s'agit d'une bactérie),

- un faux positif (FP) correspond à une réponse neuronale supérieure à $\frac{2}{3}$ alors que la réponse attendue est 0 (le RN annonce à tort une bactérie),
- une indécision (R) correspond à une réponse neuronale comprise entre les deux bornes précédentes.

Tableau 1 – Résultats du RN sur l'ensemble total.

	Bactéries	Particules
Bactéries	89,4%	4,8% (FP)
Particules	4% (FN)	83%
Rejets (R)	6,6%	12,2%

Les résultats du tableau 1 ne sont pas satisfaisants, car la présence de faux négatifs reste trop importante et impose le recours au microscope chaque fois que l'appareil croit détecter une bactérie.

5.2 Prototypage flou

Les valeurs min et max et la fonction de saturation pour chaque variable ont été fixées par une analyse rapide des données de sorte que seules quelques valeurs aberrantes ont été exclues de la pente linéaire.

Un seul prototype par classe a été utilisé car l'ajout de nouveaux prototypes n'a pas révélé une augmentation suffisamment importante de la qualité des résultats pour le justifier.

On a ajouté un seuil de décision pour rejeter le plus possible d'erreurs. Cela n'est pas toujours utile en temps normal, puisque les prototypes sont limités dans l'espace.

Les résultats sur l'ensemble total sont regroupés dans le tableau 3, à partir de différents essais et différents paramètres sur l'ensemble d'apprentissage.

Grâce au facteur d'exclusion, les faux négatifs

Tableau 2 – Tableau de contingence des résultats par prototypage flou sur l'ensemble total.

	Bactéries	Autre
Bactéries	87,1%	5,4% (FP)
Autre	<0,1% (FN)	36,2%
Rejets (R)	12,9%	58,4%

Tableau 3 – Résultats par prototypages flous sur l'ensemble total.

seuil	0.01	0.1	0.2
exclusion	0.998	0.998	0.98
faux positifs (FP)	69	44	32
faux négatifs (FN)	3	1	3
bien classés (BC)	2502	2330	2170
rejets	578	777	947

ont été considérablement réduits, au détriment d'une augmentation des rejets de particules.

Ici, le critère servant à déterminer l'utilité des variables est :

$$Cr = BC - 10.FP - 20.FN - 40.NA \quad (7)$$

BC étant le nombre de bactéries et les particules qui ont été correctement classées et NA étant le nombre de trapèzes dont le booléen Actif est vrai. Cette formule permet de pénaliser plus fortement le nombre de faux négatifs par rapport aux faux positifs et aux composants bien classés. On désire par ailleurs avoir le moins possible de trapèzes actifs, de sorte à obtenir un système plus simple et donc plus compréhensible.

Le prototype des bactéries n'utilise que 2 variables parmi les 10. Celui des non-bactéries utilise 5 variables distinctes des précédentes.

La sélection de variables a donc permis de mettre en évidence que la plupart des variables utilisées n'est pas nécessaire à une bonne modélisation de ce problème. Les résultats sur cette application ont bien montré une bonne capacité de sélection. Il est ensuite possible de fa-

cilement interpréter la connaissance représentée par les prototypes par des règles du type :

si $x_1 \in T_1$ et $x_2 \in T_2$ et ...
alors X est de la classe 1

T_i étant le numéro du trapèze d'un prototype correspondant à la variable x_i .

6 Conclusion

L'élaboration des prototypes flous s'inscrit dans un projet de DEA, poursuivi en thèse, visant à créer une structure neuronale comportant un symbolisme interne.

Il est, en outre, possible d'effectuer une sélection de variables, de manière à augmenter l'interprétabilité pour l'utilisateur et d'éliminer les variables de bruit.

L'application à un problème industriel a montré de bons résultats tant au niveau de l'optimisation que de la sélection.

Références

- [1] Dunn J. C., A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters, *J. Cybernetics*, vol. 3, num. 3, p. 32-57, 1973.
- [2] Garcia P., Glorennec P.Y., Orseau L., Détection de bactéries. *Rapport de convention*, 2003.
- [3] Guillaume S., Induction de règles floues interprétables, *Rapport de thèse au LAAS, Toulouse*, 2001.
- [4] Kohonen T., The self-organizing map, *Proc. of the IEEE*, vol. 78, p. 1464-1480, 1990.
- [5] Krishnapuram R., Generation of membership functions via possibilistic clustering, *Proc. of IEEE World Congress on Computational Intelligence*, p.902-908, 1994.
- [6] Krishnapuram R., Kim J., A note on the gustafson-kessel and adaptative clustering algorithms, *IEEE Transaction on Fuzzy Systems*, vol 7, num. 4, p. 453-461, 1999.
- [7] Marsala C., Apprentissage inductif en présence de données imprécises : construction et utilisation d'arbres de décision flous, *Rapport de thèse à l'université de Paris 6*, 1998.
- [8] Moody J., Darken C., Fast learning in networks of locally-tuned process units, *Neural Computation*, vol. 1, p. 281-294, 1989.
- [9] Russo M., Fugenesys - a fuzzy genetic neural system for fuzzy modeling, *IEEE Transactions on Fuzzy Systems*, vol. 7, num.3, p.373-388, 1998.