Generalisation par points de vue et apprentissage de concept

Joël Quinqueton, Jean Sallantin

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.: (3) 954 90 20

Rapports de Recherche

N° 265

GÉNÉRALISATION PAR POINTS DE VUE ET APPRENTISSAGE DE CONCEPT

Joel QUINQUETON
Jean SALLANTIN

Janvier 1984

GENERALISATION PAR POINTS DE VUE

ET

APPRENTISSAGE DE CONCEPT

Joel QUINQUETON
I.N.R.I.A.
B.P. 105
78153 LE CHESNAY CEDEX

Jean SALLANTIN
C.N.R.S. G.R.22
2, place Jussieu
75005 PARIS

RESUME:

Nous présentons ici une méthode d'apprentissage statistico-syntaxique, qui cherche à induire des formules presque exactes, en utilisant des opérateurs logiques et des procédures de comptage.

Ces règles sont ensuite considérées comme des "points de vue" sur le concept à apprendre, et la généralisation est faite par un "remue méninges" sur l'ensemble des points de vue.

ABSTRACT:

We present in this paper a learning technique which is both syntactical and statistical. Its aim is to learn almost true formulas from a set of examples, using logical operators and counting procedures.

These rules are in a second step considered as "beliefs" about the concept to be learnt, and the generalization is done by a "rule-storming" on this set of beliefs.

GENERALISATION PAR POINTS DE VUE ET APPRENTISSAGE DE CONCEPT

0.INTRODUCTION

En Analyse de Données, un problème d'apprentissage est ramené, soit à un problème de discrimination /1/, soit à un problème de régression. Certaines méthodes d'induction logique /4/ font appel à des notions métriques. Mais la plupart des méthodes utilisées en Intelligence Artificielle /2,3/ sont de type syntaxique, c'est à dire qu'elles se placent dans un contexte purement logique.

Le problème qui se pose alors est celui de la généralisation, c'est à dire de l'application des formules logiques induites à des objets qui ne sont pas dans l'ensemble d'apprentissage. Ce problème est de plus en plus étudié, et des solutions originales ont été proposées, basées sur la notion de filtre topologique sur l'espace des objets /5,6,8/.

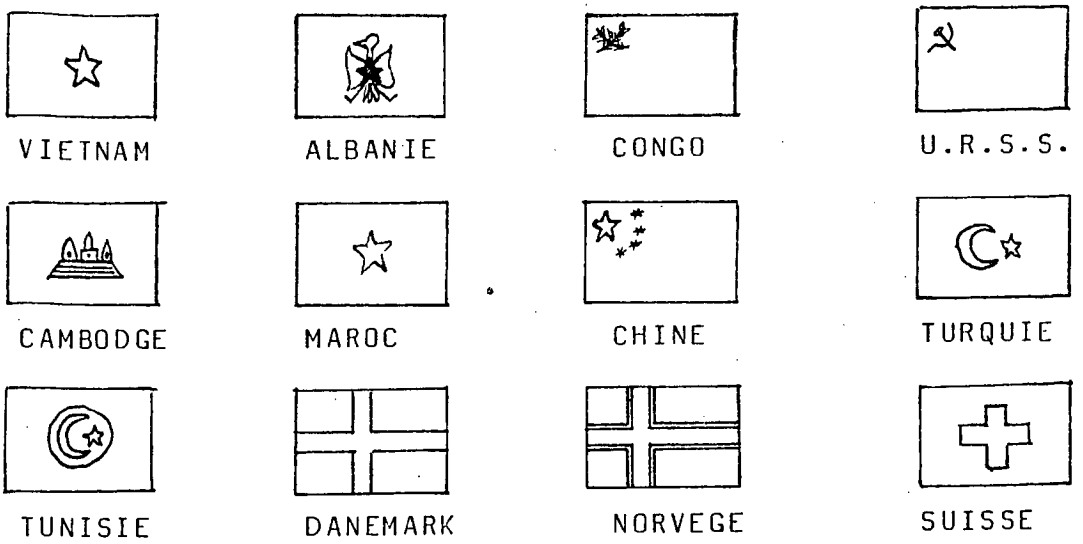
Notre méthode d'apprentissage cherche a induire des formules presque exactes, en utilisant des opérateurs logiques et des procédures de comptage.

Elle a été décrite brièvement dans /9/. Nous en donnons ici une description détaillée et des exemples de résultats.

1.LE PROBLEME

On considère un ensemble d'objets décrit par un ensemble de descripteurs, qui sont des questions (ou variables binaires) ayant une valeur sur chacun des objets.

Un descripteur est donc défini par l' ensemble de ses valeurs et par un identificateur. Nous donnons sur la figure 1 un exemple d'école qui nous servira a illustrer la méthode.



Description possible:

```
((vietnam motif motif_central etoile)
 (albanie motif motif_central etoile)
 (congo motif motif_haut_gauche)
 (urss motif motif_haut_gauche)
 (cambodge motif motif_central)
 (maroc motif motif_central etoile)
 (chine motif motif_haut_gauche etoile)
 (turquie motif motif_central etoile croissant)
 (tunisie motif motif_central etoile croissant rond)
 (danemark croix)
 (norvege croix)
 (suisse motif motif_central croix))
```

==> descripteurs obtenus:

```
((motif 1 1 1 1 1 1 1 1 1 0 0 1)
 (motif_haut_gauche 0 0 1 1 0 0 1 0 0 0 0 0)
 (motif_central 1 1 0 0 1 1 0 1 1 0 0 1)
 (etoile 1 1 0 0 0 1 1 1 1 0 0 0)
 (croissant 0 0 0 0 0 0 0 1 1 0 0 0)
 (rond 0 0 0 0 0 0 0 0 1 0 0 0)
 (croix 0 0 0 0 0 0 0 0 0 1 1 1))
```

FIGURE 1: EXEMPLE DE DESCRIPTION DES DRAPEAUX A FOND ROUGE

(Source : Petit Larousse Illustré)

Le but de la méthode proposée est de décrire cet ensemble d'une maniere qui autorise une généralisation logiquement fondée, c'est à dire l'extension du concept appris à des objets ayant une description différente de ceux de l'ensemble d'apprentissage.

La démarche que nous adoptons ici est comparable, sur le plan épistémologique, à celle qui consiste à représenter une fonction par le début de sa série de Taylor.

En effet, en logique booleenne, il existe toujours une combinaison des descripteurs à l'aide de 2 opérateurs distributifs ("et","ou"), qui soit vraie si et seulement si l'objet appartient à l'ensemble d'apprentissage.

Malheureusement, une telle description n'autorise pas de généralisation autre que l'identification.

Notre approche consiste donc à construire des approximations (que nous appelons "points de vue") de cette description, mais en nous placant dans une logique non distributive. Un seul opérateur est utilisé ("et" en général) dans les points de vue, qui sont ensuite combinés par un operateur non distributif avec le premier.

La définition en compréhension de l'ensemble d'apprentissage ainsi obtenue n'est alors plus identique à sa définition en extension (description initiale), ce qui autorise une véritable généralisation.

2.METHODE PROPOSEE

Chaque point de vue est associé à l'un des descripteurs. Le point de vue est un ensemble de règles prédisant la valeur de ce descripteur (appelé variable à expliquer) à partir des autres.

La construction d'un point de vue consiste à itérer un algorithme qui se compose de 3 modules: expansion, selection, compression.

L'expansion consiste à combiner les descripteurs 2 à 2 à l'aide d'un opérateur logique, afin d'obtenir un nouvel ensemble de descripteurs.

La sélection consiste à éliminer, dans ce nouvel ensemble, les descripteurs qui ne sont pas assez "proches", au sens d'une certaine mesure, de la variable à expliquer.

La compression intervient alors pour classifier les descripteurs selectionnés en fonction de leur "ressemblance". Chaque classe est ensuite représentée par un (ou un petit nombre) des descripteurs qui la composent.

Si nous représentons chaque descripteur par un point du plan, ces 3 étapes peuvent etre visualisées comme sur la figure 2 page suivante. Remarquons que ce type de représentation est à considérer avec un certain recul, car nous nous placons dans un contexte logique. Mais c'est néanmoins un bon moyen de visualisation.

Nous allons maintenant décrire chacun de ces modules.

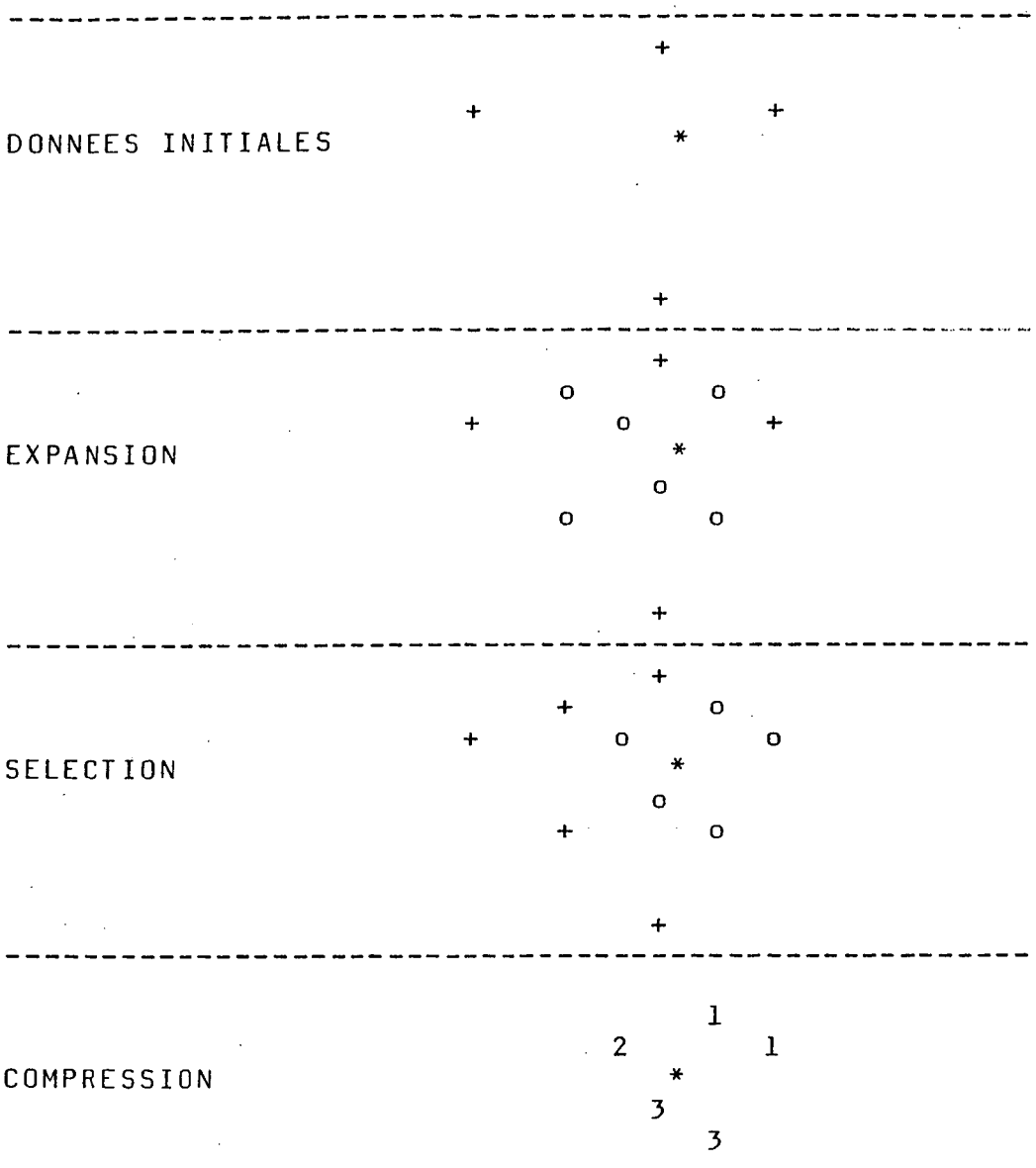


FIGURE 2 : ILLUSTRATION DE LA METHODE PROPOSEE

3.EXPANSION

Cette étape consiste à composer 2 à 2 les descripteurs afin de construire un nouvel ensemble. Nous allons présenter, dans ce paragraphe, les contraintes et les choix possibles de l'opérateur d'expansion.

3.1 Les opérateurs d'expansion

Etant donnée la méthodologie utilisée, l'opérateur doit etre associatif, faute de quoi il faudrait construire toutes les combinaisons de descripteurs 3 à 3, puis 4 à 4, etc..., ce qui donnerait à la méthode une complexité prohibitive.

D'autre part, lorsque l'on construit le descripteur a*b, il faut aussi construire les descripteurs:

(non a)*b; a*(non b); (non a)*(non b).

En effet, le fait de permuter les valeurs 0 et 1 ne doit pas influer sur le résultat de l'expansion.

Ces considérations suggèrent deux opérateurs possibles: la conjonction logique "et", et l'équivalence logique "id".

Avec la conjonction, 4 descripteurs sont construits pour chaque paire de descripteurs initiaux:

a et b; (non a) et b; a et (non b); (non a) et (non b).

Si l'ensemble initial contenait n descripteurs, nous en obtiendrons, après expansion:

$$N = n + 4\,(n\,(n-1)\,/2) = n\,(2n - 1).$$

Cette technique a cependant quelques inconvénients: peu économique, elle introduit, de facon implicite, un "ou" classique, car (non a et non b) = non (a ou b).

Une autre technique consiste à "polariser" les descripteurs, c'est à dire à choisir "a" ou "non a", celui qui est vrai sur le plus grand nombre d'objets. Ceci est possible grace à l'effet contractant de l'opérateur "et", et ne presente pas les inconvénients précédents. Elle peut cependant éliminer trop de descripteurs.

Entre ces deux extremes, il existe plusieurs possibilités. Par exemple, on peut interdire au "non" d'opérer sur un descripteur composé, c'est à dire qui ne fait pas partie de la liste initiale.

Pour l'équivalence logique, le problème est un peu plus simple, car on n'a pas besoin de construire toutes les combinaisons. En effet, on peut observer que:

a id b = (non a) id (non b);

(non a) id b = a id (non b) = non (a id b).

D'après les remarques faites au début de ce paragraphe, il suffit donc de construire "a id b". Si l'ensemble initial contenait n descripteurs, le nombre de descripteurs après expansion sera:

N = n + (n (n-1) /2) = n (n + 1) /2.

On vérifie facilement que ces opérateurs obéissent aux contraintes décrites au début de ce paragraphe.

3.2 L'expansion en LISP

Comme nous l'avons dit dans l'introduction, ces algorithmes ont été programmés en MACLISP. Le module d'expansion se compose de 3 fonctions, que nous allons maintenant décrire.

La fonction "opera" est l'opérateur d'expansion choisi par l'utilisateur. C'est une lambda-expression à 2 arguments (x y), qui sont les valeurs binaires des descripteurs, et dont la valeur est la combinaison des deux par l'opérateur choisi.

Par exemple, la conjonction et l'equivalence, dont nous avons parlé plus haut, pourraient etre définies de la facon suivante:

```
(defun et (x y) (cond ((zerop x) 0) (t y)))

(defun id (x y) (cond ((= x y) 1) (t 0)))
```

La fonction "etoil" est une lambda-expression à 2 arguments (desc lisdesc). Le premier, desc, est un descripteur, et lisdesc est une liste de descripteurs. Sa valeur est une liste de descripteurs, composition par "opera" de desc avec chacun des descripteurs de la liste lisdesc.

Cette fonction peut etre definie en LISP de la facon suivante:

```
(defun etoil (desc lisdesc)
       (cond
       ((null lisdesc) ())
       (t (cons (cons
                (list (car desc) opera (car (car lisdesc)))
                (mapcar opera (cdr desc) (cdr (car lisdesc))))
          (etoil desc (cdr lisdesc))))))
```

La dernière fonction réalise l'expansion proprement dite, en concaténant les résultats successifs de "etoil", de manière à ne pas construire plusieurs fois le meme descripteur. Elle peut etre définie de la facon suivante:

```
(defun expan (lisdesc)
       (cond
       ((null lisdesc) ())
       (t (append
          (etoil (car lisdesc) (cdr lisdesc))
          (expan (cdr lisdesc))))))
```

De plus, la liste initiale est concaténée au résultat de l'expansion.

Une autre strategie d'expansion est possible, qui consiste à la définir comme une maniere de combiner 2 listes de descripteurs. Elle peut etre définie en LISP de la facon suivante:

```
(defun expan (lisdesl lisdes2) (cond
       ((null lisdesl) ())
       (t (append
          (etoil (car lisdesl) lisdes2)
          (expan (cdr lisdesl) lisdes2)))))
```

Mais cette stratégie implique que les descripteurs de lisdesl et ceux de lisdes2 soient differents, c'est à dire le plus décorrélés possibles. Nous reviendrons sur ce point au cours de l'étape de compression.

4.SELECTION

4.1 Les critères de sélection

Le but de la sélection est de comparer un descripteur à la variable à expliquer. Le moyen le plus naturel pour comparer 2 descripteurs est de regarder leurs occurences simultanées

Cette comparaison peut se faire à l'aide de la liste (N00 N01 N10 N11) des co-occurences des différentes valeurs des descripteurs, c'est à dire dans notre cas du descripteur à sélectionner et de la variable à expliquer.

Plusieurs critères de sélection sont alors possibles, que l'on peut classer en 2 catégories: des critères de recouvrement et des critères informationnels.

Un recouvrement signifie que le descripteur a une valeur pour au moins une partie des objets d'une classe et l'autre valeur pour au plus une partie des objets de l'autre classe. Les seuils correspondants sont choisis par le programmeur.

On obtient donc un critère du type:

N11 > min1 et N10 < max0

ou

N11 < max1 et N10 > min0

Les critères informationnels se distinguent des précédents par le fait qu'ils ne sont pas utilisés de la meme facon. Ce sont des mesures d'information sur le descripteur, et les descripteurs sont ensuite ordonnés en fonction de cette mesure, ce qui permet de sélectionner les k meilleurs, pour une valeur de k choisie par le programmeur. Ils sont donc surtout utiles lorsque les critères de recouvrement semblent insuffisants pour réduire le nombre des descripteurs.

Plusieurs critères sont possibles. Citons par exemple la divergence de Kullback, la distance de Mahalanobis, ou encore le khi2 de contingence.

Leur expression à partir des co-occurences est la suivante:

DIVERGENCE = (N10/(N00+N10) - N11/(N01+N11)).LOG(N10.N01/N00.N11)

DISTANCE-MAHALANOBIS = ABS(N10-N11)/(N10+N11)**0.5

$$KHI\text{-}DEUX = N.(N00.N - (N00+N01).(N00+N10))^{**2} / (N00+N01)(N00+N10)(N11+N01)(N11+N10)$$

où $N = N00+N01+N10+N11$

4.2 La sélection en LISP

La première liste à calculer est la liste des co-occurences:

(N00 N01 N10 N11)

En fait, on n'a besoin que des valeurs N10 et N11, de maniere à les comparer aux seuils de sélection definis au paragraphe précédent. Ceci est fait par une fonction à valeur booléenne, "test", dont l'argument est un descripteur. Elle peut etre définie comme suit:

```
(defun test (desc)
       (progn
        (fillarray 'ntab '(0))
        (mapcar '(lambda (x y) (cond ((= x 1)
                                      (store (ntab y) (1+ (ntab y))))))
                (cdr desc) (cdr varexpl))
        (cond
         ((> (ntab 0) min0) (< (ntab 1) max1))
         ((< (ntab 0) max0) (> (ntab 1) min1)))))
```

La mise en oeuvre effective de la sélection se fait alors simultanément à l'expansion. Il suffit pour cela de remplacer, dans la lambda-expression de "expan", la fonction MACLISP de concaténation, "append", par une concaténation conditionnelle, "selec", qui depend du résultat de "test":

```
(defun selec (lisdes1 lisdes2)
       (cond
       ((null lisdes1) lisdes2)
       ((test (car lisdes1))
        (cons (car lisdes1) (selec (cdr lisdes1) lisdes2)))
       (t (selec (cdr lisdes1) lisdes2))))
```

En ce qui concerne les critères informationnels, le plus simple à mettre en oeuvre est la distance de Mahalanobis. Il peut etre defini en LISP à l'aide d'une fonction de tri.

Nous terminerons ce paragraphe par une remarque concernant le but de la sélection. En effet, celui-ci est de sélectionner des descripteurs qu'il semble utile de soumettre de nouveau à "expan". Le critère de sélection ne doit donc pas etre trop sévère, sous peine d'éliminer des descripteurs intéressants, mais doit aussi etre suffisamment sélectif, sous peine d'explosion combinatoire.

Nous donnons sur la figure 3 un exemple d'expansion-sélection.

Variable a expliquer:

(motif 1 1 1 1 1 1 1 1 1 0 0 1)

Descripteurs initiaux:

((motif_haut_gauche 0 0 1 1 0 0 1 0 0 0 0 0)
((non motif_haut_gauche) 1 1 0 0 1 1 0 1 1 1 1 1)
(motif_central 1 1 0 0 1 1 0 1 1 0 0 1)
((non motif_central) 0 0 1 1 0 0 1 0 0 1 1 0)
(etoile 1 1 0 0 0 1 1 1 1 0 0 0)
((non etoile) 0 0 1 1 1 0 0 0 0 1 1 1)
(croissant 0 0 0 0 0 0 0 1 1 0 0 0)
((non croissant) 1 1 1 1 1 1 1 1 0 0 1 1 1)
(rond 0 0 0 0 0 0 0 0 1 0 0 0)
((non rond) 1 1 1 1 1 1 1 1 1 0 1 1 1)
(croix 0 0 0 0 0 0 0 0 0 1 1 1)
((non croix) 1 1 1 1 1 1 1 1 1 1 0 0 0))

Expansion-Selection:

1.1.0.0.1.1.0.1.1.0.0.1.
motif_central(2. 3. 0. 7.)

1.1.0.0.0.1.1.1.1.0.0.0.
etoile(2. 4. 0. 6.)

1.1.1.1.1.1.1.1.1.0.0.0.
(non croix)(2. 1. 0. 9.)

1.1.0.0.1.1.0.1.1.0.0.1.
((non motif_haut_gauche) et motif_central)(2. 3. 0. 7.)

1.1.0.0.1.1.0.1.1.0.0.0.
((non motif_haut_gauche) et (non croix))(2. 4. 0. 6.)

1.1.0.0.1.1.0.1.0.0.0.1.
(motif_central et (non rond))(2. 4. 0. 6.)

1.1.0.0.1.1.0.1.1.0.0.0.
(motif_central et (non croix))(2. 4. 0. 6.)

1.1.0.0.0.1.1.1.1.0.0.0.
(etoile et (non croix))(2. 4. 0. 6.)

1.1.1.1.1.1.1.0.0.0.0.0.
((non croissant) et (non croix))(2. 3. 0. 7.)

1.1.1.1.1.1.1.1.0.0.0.0.
((non rond) et (non croix))(2. 2. 0. 8.)

FIGURE 3: EXPANSION-SELECTION SUR LES DONNEES DE LA FIGURE 1

5.COMPRESSION

Cette étape va consister à résumer l'ensemble des descripteurs selectionnés, en tenant compte de leurs corrélations, mesurées par une certaine fonction.

Il s'agit donc de réaliser une classification automatique des descripteurs en k classes. Il faut donc choisir une distance entre descripteurs, qui soit une mesure de leur décorrélation, puis de chercher la partition qui rende minimale la somme des distances intra-classe. Nous allons décrire successivement ces différents aspects de la compression.

5.1 Optimisation d'une partition

Dans ce paragraphe, on se pose le problème d'optimiser, au sens d'un certain critère donné, une partition en k classes. Comme nous l'avons dit dans l'introduction, on suppose choisie une distance entre descripteurs, qui mesure leur décorrélation.

Le critère à optimiser est alors la somme des distances du descripteur i aux descripteurs de sa classe. Soit donc D(i j) la somme des distances du descripteur i aux descripteurs de la classe j. Soit j(i) la classe à laquelle appartient le descripteur i. On pose:

$$W(i) = D(i\ j(i)) - \min\ (D(i\ j)\ j=1,k)$$

Soit alors i0 tel que:

$$W(i0) = \max\ (W(i)\ i=1,\ldots)$$

L'algorithme va alors effacer i0 de sa classe et l'ajouter à la classe j0 telle que:

$$D(i0\ j0) = \min\ (D(i0\ j)\ j=1,k)$$

Ce processus est ensuite itéré jusqu'à ce que la partition soit stable, c'est à dire que W(i)=0 pour tous les descripteurs. Cet algorithme est bien un algorithme d'optimisation locale, car on vérifie aisément que le critère W(i0) décroit à chaque itération. La partition de départ peut etre tirée au hasard ou choisie par le programmeur. Nous allons maintenant revenir sur ce dernier point, afin de définir une stratégie de compression.

5.2 Algorithme de compression

Une particularité intéressante de l'algorithme précédent est qu'il peut fonctionner meme si une classe est vide. Cette remarque permet de proposer une stratégie de compression, en se donnant au départ le nombre maximum de classes.

L'algorithme part d'une partition en une classe. Le critère est évidemment nul dans ce cas.

Lorsque la meilleure partition en k-1 classes a été trouvée par l'algorithme décrit au paragraphe précédent, on crée une classe vide et on cherche la meilleure partition en k classes.

On arrete lorsque k est le nombre maximum de classes. On peut remarquer que si la somme des distances intra classe est nulle pour k<nmc, alors les classes supplémentaires resteront vides.

Une fois obtenue la partition désirée, chaque classe est résumée par un (ou un petit nombre) des descripteurs qui la composent. Nous reviendrons sur ce point, ainsi que sur le choix de la di

stance, après
avoir décrit la programmation des algorithmes précédents.

5.3 Compression en LISP

Deux fonctions sont supposées définies au départ, qui sont le calcul de la distance, "dista", ainsi que le résumé d'une compression, "resum". Nous reviendrons plus en détails sur ces fonctions dans les paragraphes suivants. Signalons simplement que "dista" a pour arguments (l1 l12), où l1 est un descripteur, et où l12 est une liste de descripteurs. Sa valeur est la liste des distances de l1 aux descripteurs de la liste l12. La valeur de "resum" est la liste des descripteurs qui ont été choisis pour résumer leur classe.

La compression proprement dite utilise 3 fonctions, que nous allons maintenant décrire. Ces fonctions utilisent 4 tableaux qui sont communs et initialisés au début du programme:

vclas: tableau contenant, pour chaque descripteur, le numero de classe qui lui est affecté à un instant donné:

(j(1) j(2) ...)

vcrit: tableau contenant le critere d'évaluation, c'est à dire les distances d'un descripteur à une classe:

((D(1 1) D(1 2) ... D(1 k)) (D(2 1) ... D(2 k)) ...)

où les D(i j) sont les sommes des distances du descripteur i aux descripteurs de la classe j, tels que définis au paragraphe précédent.

vmini: tableau contenant, pour chaque descripteur, le numero de la classe de laquelle il est le plus proche, au sens du critere.

vdist: tableau contenant les distances d'un descripteur à chacun des autres.

Le tableau "vcrit" est initialisé par la fonction "inicr", qui peut etre définie en LISP de la facon suivante:

```
(defun inicr (i lisdesc) (cond
            ((null lisdesc) ())
            (t (store (vcrit i 0)
                      (somme (dista (car lisdesc) donnees)))
               (store (vclas i) 0)
               (do ((j 1 (1+ j)))
                   ((= j nmcla) (inicr (1+ i) (cdr lisdesc)))
                   (store (vcrit i j) 0)))))
```

La fonction "ameli" a pour role de changer un descripteur de classe. Son argument (numdesc) est le numero du descripteur à changer de classe. Elle peut etre definie en LISP de la facon suivante:

```
(defun ameli (numdesc) (progn
       (fillarray 'vdist
                  (dista (nieme numdesc donnees) donnees))
       (do ((i 0 (1+ i)))

           ((= i nbrdesc)
            (store (vclas numdesc) (vmini numdesc)))
           (progn
           (store (vcrit i (vclas j))
                  (- (vcrit i (vclas j)) (vdist i)))
           (store (vcrit i (vmini j))
                  (+ (vcrit i (vmini numdesc)) (vdist i)))))))
```

La fonction (nieme n ls) a pour valeur le (n+1)eme élément de la liste ls.

La fonction "compr" est la compression proprement dite, qui appelle les fonctions précédentes. C'est un programme, dont l'argument (nc) est le nombre de classes deja construites, et sa valeur est le résumé de la partition trouvée. Son expression LISP est la suivante:

```
(defun compr (nc) (cond
       ((= nc maxclas) (resum 0))
       (t (setq critere 0 numdesc 0)
          (do ((i 0 (1+ i)))
              ((= i nbdes) (cond
                           ((zerop critere) (compr (1+ nc)))
                           (t (ameli numdesc) (compr nc))))
              (progn
              (setq n 0 m infini)
              (do ((j 0 (1+ j)))
                  ((> j nc) (store (vmini i) n))
                  (cond ((< (vcrit i j) m)
                         (setq n j m (vcrit i j)))))
              (setq n (- (vcrit i (vclas i)) m))
              (cond ((> n critere)
                     (setq critere n numdesc i)))))))))
```

Le paramètre maxclas est donné par l'utilisateur et la constante "infini" est un nombre qui est aussi grand que possible.

5.4 Choix de la distance

Il s'agit ici de comparer 2 descripteurs. Comme pour l'étape de sélection, nous partirons de la liste des co-occurences:

(N00 N01 N10 N11)

Nous avons retenu 2 mesures de décorrélation entre descripteurs, qui correspondent à 2 types de relations que l'on peut rencontrer: une distance d'équivalence et une de comparabilité.

La première correspond au nombre d'objets qu'il faudrait enlever pour que les 2 descripteurs soient logiquement équivalents (ou opposés). Sa valeur est donnée par:

min (N00+N11 N01+N10)

Cette distance est calculée au moyen d'une fonction "equiv" dont les arguments (des1 des2) sont les 2 descripteurs et dont la valeur est la liste (N00+N11 N01+N10). Cette fonction est définie récursivement sur des1 et des2.

La seconde distance, de comparabilité, est le nombre d'objets qu'il faudrait enlever pour que les descripteurs (ou leurs compléments) soient logiquement comparables, c'est à dire que:

des1 ==> des2 ou des1 ==> (non des2)

Deux descripteurs comparables peuvent etre considérés comme les valeurs successives d'une variable ordinale, et ce type de compression correspond à un "dépliage" (réduction de dimension) du nuage des objets, et a été étudiée dans des travaux antérieurs /7/.

La valeur de cette distance est donnée par:

min (N00 N01 N10 N11)

Elle est calculée au moyen de la fonction "compa" dont les arguments (des1 des2) sont les 2 descripteurs et dont la valeur est la liste (N00 N11 N01 N10). Cette fonction est définie récursivement sur des1 et des2.

La distance proprement dite est calculée au moyen d'une fonction "dista", dont les arguments (desc lisdesc) sont un descripteur et une liste de descripteurs, et dont la valeur est la liste des distances de desc aux éléments de lisdesc. Elle est définie récursivement sur lisdesc, et son expression peut etre la suivante:

```
(defun dista (desc lisdesc)
       (cond
       ((null lisdesc) ())
       (t (cons
          (apply 'min (opera desc (car lisdesc)))
          (dista desc (cdr lisdesc))))))
```

La fonction "opera" est choisie par l'utilisateur parmi les fonctions "equiv" et "compa" précédemment définies.

5.5 Résumé d'une compression

Le résumé d'une compression consiste à représenter chaque classe de la partition trouvée par un (ou un petit nombre) de ses éléments. Il dépend bien sur de la distance choisie.

Dans le cas de la distance d'équivalence, on peut choisir, à priori, n'importe quel élément de la classe, puisqu'ils sont censés etre, à quelques objets près, logiquement équivalents. Il semble judicieux de choisir celui de complexité minimale, afin de ne pas augmenter inutilement la longueur des formules obtenues.

Nous donnons sur la figure 4 un exemple de compression par ce critère.

Dans le cas de la distance de comparabilité, le problème est un peu plus compliqué. En effet, ce qui caractérise les éléments d'une classe, c'est qu'ils sont, à quelques objets près, 2 à 2 comparables. Ce type de relation peut etre resumé par un tri des descripteurs, puis un arbre de décision dichotomique.

Considérons en effet le cas suivant:

ds1 ==> ds2 ==> ds3 ==> ds4 ==> ds5 ==> ds6 ==> ds7

Cette classe peut etre résumée par les 3 descripteurs suivants:

v1 = "ds4"

v2 = "si ds4 alors ds2 sinon ds6"

```
v3 = "si ds4 alors
            si ds2 alors ds3 sinon ds1
            sinon
            si ds6 alors ds7 sinon ds5"
```

Cependant, cette méthode pose le problème de la complexité des formules obtenues. On peut bien sur se limiter aux premiers (v1 et v2, par exemple ci dessus), mais le résumé risque de faire disparaitre des descripteurs interessants.

En fait, si l'on regarde de près la signification de ce type de distance entre descripteurs, on constate que ce qui caractérise les

éléments d'une meme classe, c'est qu'ils ne doivent pas etre combinés 2 à 2. En effet, le résultat d'une telle combinaison serait l'un des 2 descripteurs initiaux. C'est donc à l'étape d'expansion suivante que se fait le véritable resume de la classification. Nous reviendrons plus loin sur ce point.

6.CRITERE D'ARRET

Nous disposons maintenant d'outils pour construire des formules logiques par apprentissage. Il reste à définir un bon critère d'arret de l'algorithme.

Le premier critère qui vient à l'esprit est de fixer un nombre maximum d'itérations: c'est prudent et, en outre, cela permet de limiter la complexité des formules construites.

Mais on peut trouver d'autres critères, en énumérant les cas où il est inutile de continuer:

-liste des descripteurs vide ou réduite à un seul élément: cela peut venir d'un mauvais choix des paramètres de sélection, ou du fait qu'il n'y a rien à apprendre.

-la liste des descripteurs reste inchangée après un nouveau cycle expansion, sélection et compression (critère de "stabilité"): il est clair que les itérations suivantes donneront le meme résultat.

Enfin, si l'un des descripteurs est logiquement equivalent à la variable à expliquer (formule exacte), il est clair que notre mécanisme d'apprentissage va etre perturbé.

Il faut donc, dans ce cas, soit arreter les iterations et supprimer la variable a expliquer de la liste des descripteurs, soit effacer de la liste des descripteurs construits celui qui est équivalent à la variable à expliquer.

Nous avons maintenant decrit tous les composants de notre méthode, il reste à définir la ou les manières de combiner ces composants.

```
  1.1.0.0.1.1.0.1.1.0.0.1.
  motif_central(2. 3. 0. 7.)

  1.1.0.0.0.1.1.1.1.0.0.0.
  etoile(2. 4. 0. 6.)

  1.1.1.1.1.1.1.1.1.0.0.0.
  (non croix)(2. 1. 0. 9.)

  1.1.0.0.1.1.0.1.1.0.0.1.
  ((non motif_haut_gauche) et motif_central)(2. 3. 0. 7.)

  1.1.0.0.1.1.0.1.1.0.0.0.
  ((non motif_haut_gauche) et (non croix))(2. 4. 0. 6.)

  1.1.0.0.1.1.0.1.0.0.0.1.
  (motif_central et (non rond))(2. 4. 0. 6.)

  1.1.0.0.1.1.0.1.1.0.0.0.
  (motif_central et (non croix))(2. 4. 0. 6.)

  1.1.0.0.0.1.1.1.1.0.0.0.
  (etoile et (non croix))(2. 4. 0. 6.)

  1.1.1.1.1.1.1.0.0.0.0.0.
  ((non croissant) et (non croix))(2. 3. 0. 7.)

  1.1.1.1.1.1.1.1.0.0.0.0.
  ((non rond) et (non croix))(2. 2. 0. 8.)

-liste des numeros de classe: (3. 1. 4. 3. 0. 5. 0. 1. 2. 2.)

  1.1.0.0.1.1.0.1.1.0.0.0.
  ((non motif_haut_gauche) et (non croix))(2. 4. 0. 6.)

  1.1.0.0.0.1.1.1.1.0.0.0.
  etoile(2. 4. 0. 6.)

  1.1.1.1.1.1.1.0.0.0.0.0.
  ((non croissant) et (non croix))(2. 3. 0. 7.)

  1.1.0.0.1.1.0.1.1.0.0.1.
  motif_central(2. 3. 0. 7.)

  1.1.1.1.1.1.1.1.1.0.0.0.
  (non croix)(2. 1. 0. 9.)

  1.1.0.0.1.1.0.1.0.0.0.1.
  (motif_central et (non rond))(2. 4. 0. 6.)
```

FIGURE 4: EXEMPLE DE COMPRESSION SUR LES DONNEES DE LA FIGURE 1

7.STRATEGIE D'APPRENTISSAGE

Récapitulons les différents éléments de notre méthode, afin de définir les choix possibles de l'utilisateur.

Tout d'abord, regardons les parametres de chaque étape:

-expansion: choix de l'opérateur
choix de la stratégie

-selection: seuils taul et tau2

-compression: choix de la distance
nombre de classes

-ensemble: nombre maximum d'itérations.

D'autre part, il est clair que la première étape peut etre, soit une expansion-sélection, soit une compression, et que l'utilisateur doit pouvoir choisir en fonction du problème posé. Par exemple, le nombre d'objets par rapport au nombre de descripteurs intervient certainement dans ce choix.

Enfin, le résultat final de l'apprentissage peut etre, au choix de l'utilisateur, un ensemble de règles décorréles (c'est à dire après compression) ou non (à la dernière itération, on s'arrete à l'expansion).

L'ensemble de ces paramètres constitue ce que nous appelons la stratégie d'apprentissage.

Afin que l'utilisateur puisse effectivement choisir entre toutes les possibilités offertes, l'algorithme a été implémenté au moyen de 2 procédures qui s'appellent mutuellement. C'est l'utilisateur qui choisit celle qui est appelée en premier.

La procédure "expansion" appelle tout d'abord la fonction "expan" définie plus haut, puis appelle la procedure "compression".

La procédure "compression" initialise les tableaux définis au paragraphe 5.3 puis appelle la fonction "compr" afin de realiser la compression, et termine en appelant "expansion" si aucun des tests d'arret n'est vérifié.

8.METHODE DE GENERALISATION

8.1 Définition d'un "point de vue"

Nous appelons "point de vue" l'ensemble des regles trouvées pour prédire la valeur d'une variable à expliquer. C'est un opérateur, qui peut agir sur tout objet décrit avec la liste de descripteurs utilisee pour l'apprentissage.

En effet, si nous considérons une des règles qui composent le point de vue, il peut se présenter plusieurs cas lorsque l'on essaie de l'appliquer à un objet:

-les prémisses de la regle sont vérifiées et la conclusion aussi: dans ce cas la reponse est "oui".

-les prémisses de la règle sont vérifiées mais pas la conclusion: dans ce cas la réponse est "non".

-les prémisses ne sont pas vérifiées: la réponse est "bof" (je ne sais pas).

Cette procedure peut etre définie en LISP de la facon suivante. Soit une regle exprimee sous la forme classique des regles de production:

(... desl... et ... (non des2) ... => varexpl)

La réponse d'une telle règle peut etre evaluee par la fonction "reponse", qui prend la liste précédente comme argument, et suppose definie une fonction "non" pour la negation:

```
(defun reponse (li)
   (cond (
      (null
         (cdr li))
      (eval
         (car li))) (
      (=
         (eval
            (car li)) 1)
      (reponse
         (cdr li)))
      (t nil)))

(defun non (x) (cond ((zerop x) 1) (t 0)))
```

La réponse du point de vue sera une décision majoritaire sur les réponses des différentes règles.

On dira que le point de vue JUSTIFIE la description s'il y a une majorité de "oui", qu'il la CONTESTE si la majorité est "non", et dans les autres cas (majorité de "bof"), on parlera de SILENCE. La règle majoritaire utilisée peut etre, par exemple:

```
(NB(oui)-NB(non))  > seuil  (JUSTIFICATION)
     ou bien ...   < seuil  (CONTESTATION)
                            (sinon SILENCE)
```

où le seuil est donné par l'utilisateur.

En LISP, un point de vue est une liste de règles. La réponse du point de vue est donc la liste des réponses des règles. On peut alors définir la fonction "vote" sur cette liste:

```
(vote (mapcar 'reponse point_de_vue))
```

La figure 5 donne un exemple de cette proc'dure.

8.2 Le "remue-méninges"

Le "remue méninges" peut etre vu comme un jeu entre les points de vue. Le jeu le plus simple (niveau 0) est le vote entre les points de vue, la decision finale étant la reconnaissance de la description comme faisant partie du concept ou non.

Le jeu de niveau 1 consiste à ce que les points de vue qui contestent la description proposent de transformer cette description de manière à la justifier. La transformation consiste à changer la valeur de la variable à expliquer, c'est à dire de la conclusion commune aux règles qui composent le point de vue.

La figure 5 montre un exemple de cette transformation.

L'ensemble des descriptions ainsi obtenues (lorsque tous les points de vue ont opéré) est ensuite juge par un vote entre les points de vue.

Le jeu de niveau k consiste à itérer k fois cette procédure de petites déformations, le nombre k étant choisi par l'utilisateur.

Cette stratégie peut etre décrite, sur le plan formel, comme la construction d'une topologie sur l'ensemble d'apprentissage. En effet, la succession des jeux de niveau k définit une base de filtre autour de chacune des descriptions possibles /9/.

LISTE DES DESCRIPTEURS:

(motif motif_haut_gauche motif_central etoile croissant rond croix)

POINT DE VUE:

(((non motif_haut_gauche) et (non croix) => motif)

(etoile => motif)

((non croissant) et (non croix) => motif)

(motif_central => motif)

((non croix) => motif)

(motif_central et (non rond) => motif))

GENERALISATION:

description: (motif motif_central etoile)

code: (1 0 1 1 0 0 0)

reponse: (oui oui oui oui oui oui)

description: (motif motif_haut_gauche)

code: (1 1 0 0 0 0 0)

reponse: (bof bof oui bof oui bof)

description: (etoile)

code: (0 0 0 1 0 0 0)

reponse: (non non non bof non bof)

transformation: (motif etoile)

nouveau code: (1 0 0 1 0 0 0)

reponse: (oui oui oui bof oui bof)

FIGURE 5: EXEMPLE DE GENERALISATION SUR LES DONNEES DE LA FIGURE 1

9.VALIDATION D'UN CONCEPT APPRIS

Plusieurs techniques peuvent etre utilisées pour valider logiquement le concept appris. Ce sont ces techniques que nous allons maintenant décrire rapidement. Pour les aspects theoriques de ces techniques de validation, nous renvoyons le lecteur à la littérature /5,10/.

9.1 Controle sur l'ensemble d'apprentissage

C'est le premier type de controle qui vient à l'esprit. Il semble en effet raisonnable de vérifier qu'un pourcentage suffisant de l'ensemble d'apprentissage est bien reconnu par la règle de décision que nous avons construite.

On peut aussi effectuer ce controle sur un ensemble test, comme cela se fait classiquement en analyse de données /1/. Lorsqu'il n'y a pas trop de descripteurs, on peut meme produire une liste exhaustive des descriptions reconnues. Il suffit pour cela d'engendrer les 2**n descriptions possibles et d'appliquer le "remue-méninges" à chacune d'entre elles.

En LISP, cela peut etre fait à l'aide de la fonction "suivant", qui a pour argument la liste des noms des descripteurs:

```
(defun suivant (li)
    (cond (
       (null li) nil) (
       (zerop
          (eval
             (car li)))
       (set
          (car li) 1))
       (t
          (set
             (car li) 0)
          (suivant
             (cdr li)))))
```

9.2 Idempotence

Ce test consiste à vérifier l'idempotence de l'opération de généralisation elle meme.

Soit X l'ensemble des 2**n descriptions possibles (s'il y a n descripteurs). Soit A l'ensemble d'apprentissage. Soit G(A) le sous ensemble de X formé par les descriptions reconnues par la regle de décision construite précedemment sur l'ensemble A.

Une condition de consistence de la procédure d'apprentissage est que G(G(A)) = G(A), c'est à dire que le champ de généralisation n'est pas arbitraire.

9.3 Négation

Ce test consiste à réaliser un apprentissage sur les SILENCES, c'est à dire sur X - G(A). L'apprentissage, dans ce cas, peut faire apparaitre un concept, ou non (pas ou peu de regles dans les points de vue). Dans le premier cas, le problème était en réalité un probleme de discrimination entre au moins 2 concepts.

9.4 Réduction de l'ensemble d'apprentissage

Au cours de la procedure de jeu de niveau k que nous avons définie, les descriptions contestées sont "attirées" pas à pas par des descriptions qui sont justifiées.

Il est donc possible de sélectionner, parmi les descriptions appartenant à A, celles qui sont les plus attractives dans le jeu de niveau k. Soit R ce sous ensemble.

Cet ensemble R peut etre considéré comme une représentation réduite du concept appris, à condition de choisir k de facon que G(R) soit aussi proche que possible de G(A).

10.RESULTATS OBTENUS

10.1 Complexité de l'algorithme et performances

La complexite du module d'apprentissage (expansion-selection et compression) est quadratique par rapport au nombre de descripteurs n, mais elle est linéaire en fonction du nombre N d'objets dans l'ensemble d'apprentissage.

Quant à la generalisation, sa complexité depend surtout du niveau k de la procédure de remue-meninges. Comme tout processus arborescent, cette complexite est exponentielle.

Nous donnons ci-dessous, pour fixer les id'es, des exemples de cout CPU de la version actuellement implémentee sous Multics 68DPS. Ces chiffres viennent d'un exemple ou le nombre initial de descripteurs est n=12 et le nombre d'objets N=60.

expansion-selection		compression	
n	cout	n	cout
22	3.15	17	3.40
22	3.59	7	0.51
22	3.38	8	0.62
22	2.81	5	0.02
22	3.32	9	0.47
22	3.73	9	0.78
22	3.24	4	0.07
22	3.27	1	0.00
22	3.20	9	0.96
22	3.41	13	3.01
22	3.64	8	0.77
22	2.93	8	0.27

10.2 Application en biologie moléculaire

Dans cette application, il s'agit d'etudier le repliement des molecules d'ARN. Ces molecules peuvent etre decrites comme des sequences de 4 types de bases: T, C, A, G. Les donnees du probleme sont 3 fichiers contenant chacun une de ces sequences. Leurs longueurs sont respectivement 6508, 5574 et 5386.

Dans ces sequences, se présentent, en plusieurs endroits, un CODON, c'est à dire une sous séquence "ATG". Ce sont ces occurences qui sont les objets de notre apprentissage.

Ces objets sont decrits par la sous chaine de n (parametre fixé) bases qui les precede. Les 2n descripteurs sont obtenus par un codage de chaque base sur 2 bits, de la facon suivante:

	b1	b2
T	0	0
C	0	1
A	1	0
G	1	1

Ces objets sont de 2 types, suivant qu'ils entrainent ou non un repliement de la chaine a cet endroit. Il s'agit de caracteriser ceux qui entrainent un repliement.

Nous avons utilise la premiere sequence(6508) pour l'apprentissage, et les 2 autres comme ensemble test. Le taux de bonne reconnaissance variait entre 95 et 98%, ce qui est assez satisfaisant. De plus, les regles qui composent les points de vue trouves sont tres bien interpretables par les biologistes concernes.

10.3 Application en psychologie

Il s'agit de chercher a expliquer l'engagement d'une psychothérapie à partir d'un questionnaire décrivant différents aspects du contexte social et des antécédents psychologiques d'un ensemble de 40 individus. Le nombre d'individus dans la première classe (pas de psychothérapie) est de 32, et de 8 dans l'autre classe.

Les questions sont utilisées comme descripteurs, et nous avons, dans un premier temps, utilisé séparément 2 sous ensembles de questions. Le premier contenait 11 questions et portait sur l'environnement social, et le second sur les antécédents psychologiques avec 14 questions.

Les reponses aux questions sont des variables ordinales, pouvant prendre 4 valeurs (sans, peu, moyen, tres). Nous avons donc associe 2 descripteurs a chaque variable (00, 01, 10, 11) comme dans l'exemple precedent.

Etant donne le faible nombre d'objets, nous avons conduit l'apprentissage en 2 temps:

-Caracteriser les 40 objets dans leur ensemble

-Caracteriser les 32 objets de la le classe

D'autre part, nous nous sommes limites aux regles de longueur 2 (une seule etape expansion selection compression).

11.CONCLUSION ET PERSPECTIVES

L'algorithme que nous avons présenté dans ce rapport est la première version d'un outil pour des méthodes d'apprentissage. Nous travaillons maintenant à son amélioration et à son intégration dans un système complet d'apprentissage. Nous pensons que les résultats présentés démontrent une efficacité suffisante et que le champ d'application potentiel est assez vaste.

Les possibilités d'extension de la méthodologie que nous avons définie sont assez importantes. Citons, par exemple, le traitement de séquences (ordre externe sur les descripteurs ou sur les objets), le traitement de données hétérogènes (modalités multiples, variables ordinales), et, au dela, la logique du premier ordre.

Le but est, a terme, de définir un langage de formulation des problèmes d'apprentissage basé sur la méthodologie décrite dans ce rapport.

12.REFERENCES

/1/ G.Celeux and Y.Lechevallier, "Non Parametric Decision Trees by Bayesian Approach", COMPSTAT 1982, Physica Verlag, Vienne, 1982.

/2/ Y.Kodratoff and R.Loisel, "Learning Complex Structural Descriptions from Examples", ICPR 1982, Munich (RFA).

/3/ R.Loisel, "Apprentissage de Descriptions Structurelles Complexes", These d'Etat, Université Paris 6, Oct. 1981.

/4/ R.S.Michalski, "Pattern Recognition as Rule Guided Inductive Inference", IEEE trans on PAMI, Vol.2, no.4, 1981.

/5/ R.S.Michalski, Artificial Intelligence, Vol 20, No 2, pp 111-161, Fevrier 1983

/6/ T.Mitchell, "Version Spaces: a Candidate Elimination Approach to Rule Learning", IJCAI 1979, Tokyo (Japon).

/7/ J.Quinqueton, "Intrinsic Dimensionality of Ordinal Data", ICPR 1980, Miami Beach (USA).

/8/ J.Sallantin and J.Quinqueton, "Expansion and Compression of Binary Data to Build Features by Learning", ICPR 1982, Munich (RFA).

/9/ J.Sallantin and J.Quinqueton, "Algorithms for learning logical Formulas", IJCAI 1983, Karlsruhe (RFA).

/10/ D.Singer, "La Logique de l'Apprentissage", These de 3e cycle, Universite Paris 6, Janv. 1984

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique