Combinaisons d'automates et de boules de mots pour la classification de séquences

Frédéric Tantini Alain Terlutte Fabien Torre

(preprint à la publication RIA 2011)

Résumé

Dans cet article, nous présentons un cadre d'apprentissage général pour la classification supervisée. Ce cadre ne nécessite que la définition d'un opérateur de généralisation et fournit en particulier des méthodes d'ensemble. Pour les tâches de classification de séquences, nous montrons que l'inférence grammaticale, avec des objectifs différents, a déjà défini de tels apprenants pour certaines familles d'automates comme les réversibles ou les k-TSS. Nous proposons ensuite un opérateur de généralisation original pour la famille des boules de mots. Enfin, nous montrons au travers de différentes expérimentations que notre approche permet effectivement de résoudre des tâches de classification de séquences.

In this paper, we present a general framework for supervised classification. This framework only needs the definition of a generalisation operator and provides ensemble methods. For sequence classification tasks, we show that grammatical inference has already defined such learners for automata classes like reversible automata or k-TSS automata. Then we propose a generalisation operator for the class of balls of words. Finally, we show through experiments that our method efficiently resolves sequence classification tasks.

Mots-clefs. automates moindre généralisés, boules de mots, méthodes d'ensemble, inférence grammaticale, classification de séquences.

Keywords. least general automata, ball of words, ensemble methods, grammatical inference, sequence classification.

1 Introduction

Avec comme objectif la classification supervisée de séquences, nous allions dans cet article deux domaines de l'apprentissage automatique jusque là étrangers.

Nous considérons les opérateurs de généralisation guidés par les données et parmi ceux-là le moindre généralisé, notion qui apparaît dans différents contextes d'apprentissage [Plotkin, 1970, Ganascia, 1993, Bourgne et al., 2010]. De plus, nous souhaitons bénéficier des méthodes d'ensembles : celles-ci combinent beaucoup de classifieurs basiques pour produire une prédiction. Ces techniques font perdre la compréhensibilité de la théorie apprise, mais permettent de gagner en qualité de prédiction. Nous utilisons le système VOLATA qui propose des méthodes d'ensemble combinant des hypothèses moindres généralisées, comme cela a été montré pour des exemples représentés en attributs-valeurs et des hypothèses qui sont des hyper-rectangles [Torre, 2005].

Par ailleurs, l'inférence grammaticale est le domaine qui s'intéresse à l'identification de langages de mots en inférant des grammaires, des automates, des expressions régulières, etc. Quelle que soit la représentation choisie, un paradigme courant de l'inférence grammaticale est celui de *l'apprentissage à la limite par positifs seuls* : les mots du langage sont fournis un

par un à l'apprenant, à chaque arrivée d'un exemple l'apprenant doit proposer un langage qui contient les mots déjà vus et finalement se stabiliser sur le langage cible.

Nous voulons dans cet article généraliser la démarche VOLATA à des exemples qui sont des séquences, les classifieurs venant alors de la théorie des langages, et cela en utilisant au mieux les outils théoriques fournis par l'inférence grammaticale.

Le plan est le suivant. Section 2, nous rappelons les fondamentaux d'un apprentissage basé sur un opérateur de généralisation guidé par les données et sa mise en œuvre dans le système VOLATA. Cette méthode est instanciée pour des automates particuliers en section 3, puis pour les boules de mots section 4. Pour les automates considérés, des opérateurs de moindre généralisé existent mais cela n'est pas le cas pour les boules de mots. Nous proposons alors un calcul de généralisation de boules et discutons de ces propriétés. À la section 5, nous mettons en œuvre ces algorithmes sur des jeux de données classiques de la classification de séquences, puis sur un problème réel de reconnaissance de caractères manuscrits. Enfin, section 6, nous faisons un bilan de ce travail et en proposons les perspectives.

2 Apprentissage par moindres généralisés : le système VOLATA

Nous sommes dans le cadre de la classification supervisée dont nous rappelons les fondamentaux. Poser un problème d'apprentissage supervisé nécessite de définir $\mathcal{E}$ un langage de représentation pour les exemples et $\mathcal{H}$ un langage d'hypothèses, vérifiant ensemble l'*astuce de représentation unique* [Feigenbaum, 1977] : tout exemple est également une hypothèse ($\mathcal{E} \subset \mathcal{H}$). Sur ces ensembles, nous devons disposer d'un test de subsomption $\succeq$ indiquant qu'une hypothèse *couvre* un exemple. La couverture d'une hypothèse est l'ensemble des exemples qu'elle subsume. Cette notion de couverture permet d'étendre la subsomption aux hypothèses : une hypothèse en subsume une autre si la couverture de la première contient la couverture de la seconde, on dit aussi que la première est plus générale que la seconde. Ainsi, une relation de subsomption $\succeq$ est définie sur $\mathcal{H} \times \mathcal{H}$.

En classification supervisée, nous devons disposer d'un ensemble de classes $\mathcal{Y}$, l'objectif final étant de prédire une classe pour chaque nouvel exemple. Avant cette phase de classification, une méthode supervisée reçoit pour apprendre un échantillon, c'est-à-dire un ensemble de couples (x_i, y_i) avec $x_i \in \mathcal{E}$ et $y_i \in \mathcal{Y}$. Une manière de quantifier la réussite d'un tel apprenant est de mesurer ensuite un taux d'erreur sur un ensemble de test, cet ensemble étant disjoint de l'échantillon d'apprentissage et non disponible pour l'apprentissage. Un but est donc d'apprendre au mieux pour minimiser ce taux d'erreur.

Nous nous inscrivons dans la méthodologie d'apprentissage proposée dans [Torre, 2005] et implémentée dans le système VOLATA. Cette méthodologie consiste à se donner $\mathcal{E}$, $\mathcal{H}$ et $\succeq$, puis à définir un opérateur de généralisation sur $\mathcal{H} \times \mathcal{E}$ qui respecte la relation $\succeq$. Ces points étant fixés, le système VOLATA propose des algorithmes génériques, indépendants de la représentation des exemples et des hypothèses, comme de la relation de subsomption choisie et de l'opérateur de généralisation. Chaque méthode implémentée dans VOLATA vise à apprendre un ensemble d'hypothèses à partir d'un échantillon. En l'absence de bruit, chacune de ces hypothèses est *correcte* (parmi les exemples fournis pour l'apprentissage, elle ne couvre que les exemples de sa classe) et la disjonction formée par l'ensemble est *complet* (tout exemple d'apprentissage est couvert par au moins une règle). Les différents algorithmes d'apprentissage proposés se distinguent par l'objectif poursuivi :

— DLG [Webb and Agar, 1992] est rapide et permet d'estimer le nombre d'hypothèses nécessaire à la couverture de l'échantillon ;

— GLOBO [Torre, 2005] produit un grand nombre d'hypothèses puis opère une couverture minimale pour ne garder qu'un petit ensemble compréhensible ;

— des méthodes d'ensemble, comme le *bagging* ou le *boosting*, dont l'objectif est d'apprendre une combinaison d'un grand nombre d'hypothèses qui vont voter pour déterminer la classe d'un nouvel exemple.

Dans cet article, nous nous concentrons sur les méthodes d'ensemble et en particulier sur la plus simple d'entre elles, GLOBOOST, dont le principe est de produire aléatoirement des hypothèses correctes et de les faire voter à égalité. Pour ces méthodes en général et pour GLOBOOST en particulier, la notion de *diversité* des hypothèses est importante [Kuncheva and Whitaker, 2003] : intuitivement, le vote d'un grand nombre d'hypothèses n'a de sens que si ces hypothèses sont différentes.

Dans la suite de cette section, nous décrivons les opérateurs de généralisation qui nous intéressent, en particulier les moindres généralisés, et enfin GLOBOOST.

2.1 Opérateurs de généralisation

$(\mathcal{H}, \succeq)$ induit un graphe de recherche que nous allons parcourir à l'aide d'un *opérateur de raffinement*. Un *opérateur de raffinement* $\mathcal{O}$ est un algorithme qui, à partir d'une hypothèse de $\mathcal{H}$, calcule un nombre fini d'hypothèses de $\mathcal{H}$. Un tel opérateur suit la relation de subsomption, il est soit ascendant ($\forall h \in \mathcal{H} : \mathcal{O}(h) \subseteq \{h' : h' \succeq h\}$), soit descendant ($\forall h \in \mathcal{H} : \mathcal{O}(h) \subseteq \{h' : h \succeq h'\}$). Ainsi défini, un opérateur de raffinement n'utilise pas les exemples de l'échantillon mais uniquement la structure de l'hypothèse à généraliser. Ces opérateurs ont été étudiés, plusieurs propriétés souhaitables ont été proposées et réunies pour caractériser des *opérateurs optimaux* [De Raedt and Bruynooghe, 1993], des *opérateurs idéaux* dont il est prouvé qu'ils n'existent pas toujours car les propriétés attendues pour l'idéalité peuvent être incompatibles [van der Laag and Nienhuys-Cheng, 1994, Torre and Rouveirol, 1997], ou encore des *opérateurs parfaits* [Torre, 2000].

Nous considérons pour notre part des opérateurs ascendants et guidés par les données. Cela signifie que nous considérons des opérateurs qui généralisent, non pas une hypothèse seule, mais une hypothèse avec un exemple. Enfin, nous ajoutons que les opérateurs qui nous intéressent doivent fournir une et une seule généralisation, autrement dit nos opérateurs sont des fonctions avec le profil suivant :

$$\mathrm{G} : \begin{cases} \mathcal{H} \times \mathcal{E} & \rightarrow & \mathcal{H} \\ (h, e) & \rightarrow & h' \end{cases}$$

Cette contrainte d'unicité procure une indéniable efficacité lors de la recherche d'hypothèses : notre opérateur suit un chemin dans $(\mathcal{H}, \succeq)$, sans avoir ni de point de choix à gérer, ni de *backtrack* à effectuer, ni d'ensemble d'hypothèses candidates à maintenir. Naturellement, on s'attend à ce qu'un tel opérateur calcule une généralisation couvrant à la fois l'hypothèse de départ et l'exemple ajouté. C'est ce qui est précisé par la propriété suivante.

Définition 1 (monotonie inclusive) *Un opérateur* G *est in-monotone si et seulement si :* $\forall h \in \mathcal{H}, \forall e \in \mathcal{E} : g = \mathrm{G}(h, e) \Rightarrow (g \succeq h) \wedge (g \succeq e)$.

Pour traiter les exemples un par un, ces opérateurs sont dits *incrémentaux* et posent la question de la sensibilité à l'ordre de présentation des exemples. Nous noterons $\mathrm{G}^*(E)$ l'application successive de l'opérateur G à une séquence d'exemples $E \subseteq \mathcal{E}$: $\mathrm{G}^*(E) = \mathrm{G}(\mathrm{G}(\ldots, \mathrm{G}(E[1], E[2]), \ldots E[n-1]), E[n])$.

Définition 2 (sensibilité) *Un opérateur* G *est dit sensible à l'ordre des exemples s'il existe deux séquences* $E, E' \subseteq \mathcal{E}$ *contenant les mêmes* n *exemples mais dans des ordres différents et telles que :* $\mathrm{G}^*(E) \neq \mathrm{G}^*(E')$.

Cette sensibilité sera souhaitable ou non selon que l'on privilégie la diversité des hypothèses apprises ou un certain déterminisme du processus d'apprentissage. Nous introduisons ci-dessous un autre type de monotonie.

Définition 3 (monotonie exclusive) *Un opérateur* G *est* ex-monotone *ssi lorsque* $\forall p, n \in \mathcal{E}, \forall h \in \mathcal{H} : \text{G}(h, p) \succeq n$ *alors pour toute séquence d'exemples* $E \subseteq \mathcal{E}$ *:* $\text{G}^*(E) \succeq p \Rightarrow \text{G}^*(E) \succeq n$.

Par ailleurs, le choix de G induit la *granularité* de la recherche. Si G est trop fin, G ne généralise pas du tout et un apprentissage par cœur va survenir. Si G est trop grossier, toute généralisation de quelques exemples d'une même classe par G couvre des contre-exemples et produit donc des hypothèses incorrectes.

Une dernière propriété couramment exigée, par exemple par les opérateurs *optimaux*, est celle de *minimalité* ou de *plus petit pas* : de manière générale pour un opérateur de généralisation $\mathcal{O}$, si $h \in \mathcal{O}(g)$, il ne doit pas exister d'hypothèse h' plus spécifique que h qui subsume g. Dans notre cadre, cette propriété se confond avec la notion de moindre généralisé.

2.2 Opérateurs de moindre généralisation

Cette notion de *moindre généralisé* porte sur les hypothèses de $\mathcal{H}$.

Définition 4 (moindre généralisé) *Étant donné un ensemble d'exemples* $E \subseteq \mathcal{E}$*, une hypothèse* $h \in \mathcal{H}$ *est dite* moindre généralisée *de* E *si et seulement si* $\forall e \in E : h \succeq e$ *et il n'existe pas* h' *vérifiant* $\forall e \in E : (h' \succeq e) \wedge (h \succ h')$.

Notons que le moindre généralisé d'un seul exemple est cet exemple lui-même puisque tout exemple est une hypothèse (astuce de représentation unique). Notons également que cette définition n'implique pas l'unicité de l'hypothèse moindre généralisée d'un ensemble d'exemples, nous le vérifierons à la section 4 avec un couple $(\mathcal{H}, \succeq)$ qui induit des moindres généralisés multiples. C'est pourtant la condition première de nos opérateurs et nous allons donc supposer que tout ensemble d'exemples a une hypothèse moindre généralisée unique (cela revient à demander que $\mathcal{H}$ dotée de $\succeq$ soit un sup-demi-treillis) et noter MG l'opérateur incrémental correspondant.

Observons immédiatement que cet opérateur est insensible à l'ordre de présentation des exemples puisque le moindre généralisé d'un ensemble d'exemples a été posé unique. Par définition du moindre généralisé, tout opérateur MG est in-monotone et par son unicité, tout opérateur MG est ex-monotone. En imposant une stratégie de plus petit pas à travers un opérateur MG, nous reportons la question de la granularité induite par G sur la granularité intrinsèque de $(\mathcal{H}, \succeq)$. Les langages d'hypothèses $\mathcal{H}$ qui nous intéressent ne doivent pas permettre l'apprentissage par cœur : le moindre-généralisé d'un ensemble d'au moins deux exemples doit couvrir d'autres exemples ; autrement dit, la simple disjonction des exemples ne doit pas être représentable dans $\mathcal{H}$.

Munis d'opérateurs de généralisation, nous pouvons discuter de leur implication dans la réalisation d'un apprentissage supervisé.

2.3 Apprentissage par opérateur de généralisation et combinaisons

Étant donné un opérateur de généralisation G, l'apprentissage supervisé par VOLATA nécessite deux opérations : la construction d'hypothèses correctes et la production d'un ensemble complet d'hypothèses diverses.

2.3.1 Construction d'une hypothèse correcte

Nous devons prendre en compte les classes associées aux exemples et fournies avec l'échantillon d'apprentissage. Il s'agit d'appliquer un opérateur de généralisation aux exemples d'une même classe pour obtenir une hypothèse dédiée à cette classe (que nous désignons

comme *la classe cible*). Il vient alors naturellement qu'une telle hypothèse doit être construite en opposition aux exemples des autres classes (qui sont alors des *contre-exemples*).

Algorithme 1 Briques algorithmiques du système VOLATA

FONCTION GC

ENTRÉES : $E = [p_1, \ldots, p_n] \subseteq \mathcal{E}$ un ensemble *ordonné* de n exemples de même classe, $N \subseteq \mathcal{E}$ un ensemble de contre exemples.

SORTIE : $h \in \mathcal{H}$ généralisant en partie E et correcte vis-à-vis de N.

1: **début**
2: $g = p_1$
3: **pour** $i = 2$ to n **faire**
4: $g' = \text{G}(g, p_i)$
5: **si** $(\forall e \in N : g' \not\succeq e)$ **alors**
6: $g = g'$
7: **fin si**
8: **fin pour**
9: **renvoyer**

$$h(x) = \begin{cases} \text{classe}(E) \text{ si } g \succeq x \\ 0 \text{ sinon (abstention)} \end{cases}$$

10: **fin**

FONCTION GLOBOOST

ENTRÉES : des exemples étiquetés $(x_i \in \mathcal{X}, y_i \in \mathcal{Y})$ et T un nombre d'itérations.

SORTIE : H le classifieur final.

1: **début**
2: **pour** $t = 1$ to T **faire**
3: $cible$ = auHasardDans($\mathcal{Y}$)
4: $P = [x_i | y_i = cible]$
5: $N = [x_i | y_i \neq cible]$
6: mélanger P aléatoirement
7: $h_t = \text{GC}(P,N)$
8: **fin pour**
9: **renvoyer**

$$H(x) = \underset{y \in \mathcal{Y}}{\text{argmax}} \left(\sum_{1 \leq t \leq T}^{h_t(x)=y} +1 \right)$$

10: **fin**

Dans le cas où l'absence de bruit est admise, nous proposons GC (*généralisation correcte*) décrit à l'algorithme 1 pour contrôler G et assurer la correction de l'hypothèse produite. Le premier exemple est nommé *graine* et sert d'hypothèse initiale. C'est à cette étape (ligne 2) qu'intervient l'*astuce de représentation unique* : un exemple doit pouvoir s'interpréter comme une hypothèse. Avec cette graine, GC essaye de généraliser, à l'aide de l'opérateur G, chaque autre exemple de la classe cible ; une étape de généralisation doit être validée vis-à-vis des contre-exemples, un exemple qui amène une généralisation incorrecte n'est pas accepté ; on continue alors l'ajout des exemples de la classe cible à partir de la dernière généralisation correcte.

Basé sur un opérateur de généralisation G, GC procède de manière ascendante. Si G est in-monotone, les hypothèses produites successivement au cours d'une exécution de GC sont incluses les unes dans les autres et cela nous garantit qu'un exemple couvert à un instant le sera aussi par les hypothèses suivantes. Si G est ex-monotone, un exemple de la classe cible rejeté à une certaine étape par GC, ne sera pas couvert par l'hypothèse finalement produite. Cela découle directement de la définition de cette monotonie : si GC rejette une généralisation $\text{G}(h, p)$ car $\text{G}(h, p)$ subsume un contre-exemple n alors toute généralisation de h par G couvrant p couvrira également n, et une telle généralisation sera à son tour rejetée par GC. Une hypothèse produite par GC est en ce sens *maximalement correcte* : aucun exemple de la classe cible non déjà couvert ne peut lui être ajouté sans perdre la correction. Cela confère une certaine efficacité à GC qui en fin d'exécution connaît la couverture de l'hypothèse produite sans avoir à la calculer explicitement. Notons que GC a une dépendance à l'ordre de présentation des exemples, même si G n'y est pas sensible : selon sa position dans la liste des candidats, un exemple peut être généralisé avec l'hypothèse courante, passé le test de correction et être accepté, ou bien engendrer une généralisation incorrecte et être rejeté. Cette caractéristique est un avantage si la diversité de l'ensemble d'hypothèses appris est requise. Enfin, indiquons qu'un calcul de moindre généralisé cumule

les avantages précédents avec la minimalité en plus : GC suit un chemin dans $\mathcal{H}$ en procédant par plus petit pas à l'aide de MG, lequel vérifie nos propriétés de monotonie, et avec une diversité assurée par la sensibilité à l'ordre de GC. Il reste à itérer ce processus pour produire une combinaison d'hypothèses.

2.3.2 Production d'un ensemble d'hypothèses complet

GLOBOOST est décrit à l'algorithme 1 et vise à produire aléatoirement T hypothèses correctes qui constitueront un classifieur en votant avec un poids identique. À chacune des T étapes, une classe est choisie au hasard, les ensembles d'exemples et de contre-exemples sont constitués, puis l'algorithme GC est appelé sur ces exemples mélangés aléatoirement (ce qui va jouer sur la sensibilité à l'ordre de GC et ainsi encourager la diversité des hypothèses).

Notons que GLOBOOST n'est pas à proprement parler un algorithme de *boosting* : il ne bénéficie pas de preuve dans le cadre PAC, ni de borne sur l'erreur empirique. GLOBOOST ne s'apparente pas non plus aux algorithmes de *bagging* car il n'y a pas d'échantillonnage des exemples d'apprentissage. Finalement, GLOBOOST est simplement un algorithme *randomisé*.

2.4 Bilan VOLATA, vers la classification de séquences

Les deux algorithmes présentés, GC et GLOBOOST, sont génériques et nécessitent uniquement de définir un opérateur G utilisé par GC pour généraliser une hypothèse avec un exemple. Ainsi, la démarche VOLATA consiste à choisir $\mathcal{H}$ et $\succeq$, puis à étudier en fonction de ces choix l'existence d'un opérateur MG (rappelons que cela requiert l'unicité de l'hypothèse moindre généralisée de tout ensemble d'exemples). À défaut, en particulier dans le cas où il n'y a pas unicité, nous proposerons un opérateur de généralisation et étudierons ses propriétés : monotonies, sensibilité à l'ordre, granularité induite et diversité produite.

Dans la suite de cet article, les exemples sont des mots construits sur un alphabet Σ fixé ($\mathcal{E} = \Sigma^*$) et une hypothèse doit dénoter un langage, c'est-à-dire un sous-ensemble de Σ^*. Dans cet article, l'ensemble des hypothèses $\mathcal{H}$ est une famille particulière d'automates ou la famille des boules de mots. Ces hypothèses disposent d'un test de subsomption naturel qui est le test d'appartenance d'un mot au langage. Dans ce cadre, un opérateur MG prend un langage L et un nouveau mot m et produit le plus petit langage (au sens de l'inclusion des langages) qui contient à la fois L et m.

L'identification de langages (représentés par des automates, des grammaires, *etc.*) est le but de l'*inférence grammaticale*. Un paradigme courant de l'inférence grammaticale consiste à choisir une famille de langages puis à étudier *son apprenabilité à la limite par positifs seuls* : quel que soit le langage cible, les mots du langage sont fournis un par un à l'apprenant, à chaque arrivée d'un exemple l'apprenant doit proposer un langage qui contient les mots déjà vus et finalement se stabiliser sur le langage cible. Nous le voyons : un opérateur MG dédié à la famille choisie permet d'apprendre à la limite et constitue donc une preuve d'apprenabilité. Ainsi, à la recherche d'opérateurs MG pour certaines familles de langages, il nous faut commencer par examiner les preuves d'apprenabilité établies pour ces familles. Rappelons qu'en général la simple union des mots n'est pas exprimable dans les familles de langages qui nous intéressent, cela pour éviter que notre stratégie de plus petit pas ne conduise à un apprentissage par cœur. Dans les deux sections suivantes, nous nous penchons sur deux familles d'automates, puis sur la famille des boules de mots.

3 Généralisation des mots en automates

Dans le domaine de l'inférence grammaticale, nous nous intéressons à deux classes d'automates qui disposent de résultats d'apprenabilité à partir de positifs seuls : les k-TSS [García and Vidal, 1990] et les 0-réversibles [Angluin, 1982]. Nous allons voir dans cette

section que ces preuves d'apprenabilité fournissent les opérateurs MG que nous recherchons, *modulo* quelques adaptations.

3.1 Automates k-TSS

Grossièrement, un langage *k-testable au sens strict* (k-TSS) se définit par les séquences de taille k autorisées à apparaître dans les mots du langage et sont proches en cela des N-grammes. Formellement, un langage k-TSS est défini par plusieurs ensembles.

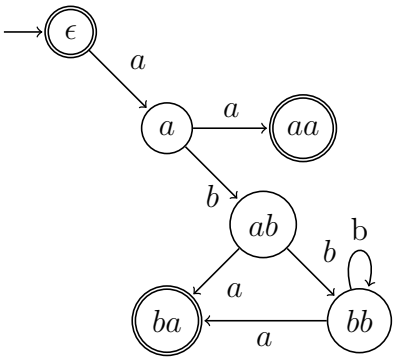
Définition 5 (langage k testable au sens strict) *Un langage k-TSS, k étant fixé et supérieur à 1, est défini par : I les segments initiaux autorisés u ($|u| = k-1$), F les segments finaux autorisés v ($|v| = k-1$), T les segments interdits, tous de taille égale à k et W les mots w du langage tels que $|w| < k-1$). Les mots du langage sont obtenus comme suit : $L(\Sigma, I, F, T, W) = W \cup (I\Sigma^* \cap \Sigma^* F) \setminus \Sigma^* T \Sigma^*$.*

Algorithme 2 Inférence d'automates k-TSS moindres généralisés

FONCTION MGTSSI
ENTRÉES : $h = (Q, \delta, q_0, F)$ un k-TSS, w un mot et k un entier
SORTIE : h' automate k-TSS généralisation minimale de h couvrant w

1: **début**
2: $q = q_0$
3: **pour** $i = 1$ à $|w|$ **faire**
4: $v = q.w_i$ { *on concatène le mot de q et la $i$ème lettre de w* }
5: **si** ($|v| = k$) **alors**
6: $v = v_{2,\ldots,|v|}$
7: **fin si**
8: $nq = v$
9: ajouter nq à Q
10: ajouter (q, w_i, nq) à δ
11: $q = nq$
12: **fin pour**
13: ajouter q à F
14: **renvoyer** $h' = (Q, \delta, q_0, F)$
15: **fin**

Avec les mots $\{\epsilon, aa, aba, abba, abbba\}$ et $k = 3$, on apprend l'automate :



soit le langage 3-TSS suivant :

$$\begin{aligned} \Sigma &= \{a, b\} \\ W &= \{\epsilon\} \\ I &= \{aa, ab\} \\ F &= \{aa, ba\} \\ T &= \{aaa, aab, baa, bab\} \end{aligned}$$

[García and Vidal, 1990] ont proposé un algorithme par exemples positifs seuls et ont prouvé que celui-ci calcule le plus petit langage k-TSS incluant l'échantillon. Avec notre vocabulaire, [García and Vidal, 1990] ont démontré l'unicité du moindre généralisé dans la famille des k-TSS et nous fournissent une méthode de calcul. Nous avons simplement évacué de cette méthode la boucle gérant le flux des exemples pour proposer une version incrémentale nommée MGTSSI (algorithme 2 dont la clef est que chaque état est représenté par les $k-1$ dernières lettres lues avant d'atteindre cet état). Cet algorithme nous permet également de satisfaire l'*astuce de représentation unique* : à un exemple-mot w correspond le plus petit langage k-TSS reconnaissant w, c'est-à-dire l'hypothèse-k-TSS obtenue par MGTSSI$((q_0, \emptyset, q_0, \emptyset), k, w)$.

3.2 Automates 0-réversibles

La famille des k-réversibles [Angluin, 1982] est souvent jugée plus intéressante que la précédente, parce que plus expressive. En particulier, les k-TSS ne permettent pas de compter et donc d'identifier des langages qui posent des conditions sur les nombres d'occurrences ou sur les positions des lettres de l'alphabet. Ainsi le langage sur $\Sigma = \{a, b\}$ des mots contenant un nombre pair de a et un nombre pair de b, est un langage 0-réversible mais pas k-TSS, quel que soit k.

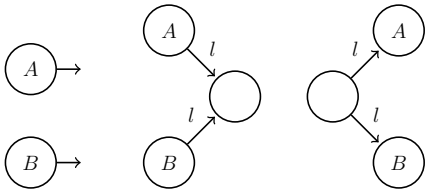
Dans cet article, nous nous restreignons à $k = 0$, par simple souci de simplicité, mais la méthode proposée s'étend sans difficulté à un k quelconque. Un automate est 0-réversible s'il est déterministe et si son miroir l'est aussi. Notons que, quand cela est nécessaire, un exemple-mot w peut être vu comme l'hypothèse-automate qui reconnaît uniquement w (il s'agit bien d'un automate 0-réversible). Un calcul de moindre généralisé dans la famille des 0-réversibles est proposé dans [Angluin, 1982] : il consiste à calculer le PTA (*Prefix Tree Acceptor*, automate qui reconnaît uniquement les mots de l'échantillon) puis à fusionner des états pour obtenir un automate réversible. Notre version incrémentale (MGZR, algorithme 3) consiste à ajouter le nouveau mot dans l'automate existant à la manière du PTA : au besoin une nouvelle branche uniquement dédiée à reconnaître ce mot est créée. Puis, MGZR opère les fusions nécessaires pour n'avoir qu'un état final, des transitions déterministes et un automate-miroir également déterministe.

Algorithme 3 Inférence d'automates 0-réversibles moindres généralisés

FONCTION MGZR
ENTRÉES : $h = (Q, \delta, q_0, F)$ un automate 0-réversible, w un mot
SORTIE : h' un automate 0-réversible, généralisation minimale de h reconnaissant w

début
$i = 1$; $q = q_0$
{ *suivi des états existants* }
tant que $\delta(q, w_i)$ est défini **faire**
  $q = \delta(q, w_i)$; $i = i + 1$
fin tant que
{ *création d'une nouvelle branche pour les lettres restantes* }
tant que $i \leq |w|$ **faire**
  créer un état q' et l'ajouter à Q
  ajouter (q, w_i, q') à δ
  $i = i + 1$
fin tant que
ajouter q à F
{ *mise en œuvre des fusions* }
fusionner tous les états de F
répéter
  si $\exists A, B \in Q$ et $\exists l \in \Sigma$ tels que $\delta(A, l) = \delta(B, l)$, fusionner A et B
  si $\exists A, B, E \in Q$ et $\exists l \in \Sigma$ tels que $\delta(E, l) = \{A, B\}$, fusionner A et B
jusqu'à plus de fusion
renvoyer $h' = (Q, \delta, q_0, F)$
fin

Les trois cas dans lesquels ZR fusionnent les états A et B ($l \in \Sigma$) :



4 Généralisation des séquences en boules de mots

4.1 Introduction aux boules de mots

Les hypothèses considérées dans cette section sont des *boules de mots*, une boule étant définie par un mot-centre o, un rayon r et notée $B_r(o)$. Une boule de mots $B_r(o)$ *contient* l'ensemble des mots à distance inférieure ou égale à r de o, c'est-à-dire, $B_r(o) = \{w \in$

$\Sigma^*|d(o,w) \leq r\}$. Autrement dit, le test de subsomption $\succeq$ entre une hypothèse $h = B_r(o)$ et un exemple e est simplement le test d'appartenance d'un mot à la boule : $h \succeq e \Leftrightarrow d(e,o) \leq r$. Le test de subsomption entre deux hypothèses $h = B_r(o)$ et $h' = B_{r'}(o')$ est l'inclusion d'une boule dans une autre : $h \succeq h' \Leftrightarrow \forall e \in h' : h \succeq e$. Pour satisfaire l'*astuce de représentation unique*, un exemple-mot w peut devenir $B_0(w)$, l'hypothèse-boule de rayon 0 centrée sur w.

La distance utilisée est la distance d'édition, ou distance de [Levenshtein, 1965], pour laquelle chaque opération d'édition parmi l'insertion, la suppression et la substitution, a un coût unitaire. On peut la définir comme étant le nombre de pas minimal de réécriture permettant de passer d'un premier mot à un second. Plus précisément, étant donnés deux mots $w, w' \in \Sigma^*$, on dit que w se réécrit en w' en un pas, noté $w \to w'$ si l'une des conditions suivantes est vérifiée :
— $w = uav$ et $w' = uv$ avec $u, v \in \Sigma^*$ et $a \in \Sigma$ *(suppression)* ;
— $w = uv$ et $w' = uav$ avec $u, v \in \Sigma^*$ et $a \in \Sigma$ *(insertion)* ;
— $w = uav$ et $w' = ubv$ avec $u, v \in \Sigma^*$, $a, b \in \Sigma$, $a \neq b$ *(substitution)*.
Nous notons $w \xrightarrow{k} w'$ si w peut se réécrire en w' à l'aide de k opérations d'édition.

Définition 6 (Distance d'édition) *La distance d'édition entre deux mots w et w', notée $d(w,w')$, est le nombre minimum d'opérations d'édition nécessaires pour réécrire w en w' ; c'est donc le plus petit entier $k \in \mathbb{N}$ tel que $w \xrightarrow{k} w'$.*

Exemple 1 *On peut montrer que $d(abaa, aab) = 2$ car abaa nécessite au moins deux pas pour être transformé en aab : $a\underline{b}aa \to aa\underline{a} \to aab$. Notons que ce n'est pas la seule façon de passer du mot abaa au mot aab : en effectuant d'abord la substitution du a par un b, la transformation devient $aba\underline{a} \to a\underline{b}ab \to aab$.*

Le calcul de la distance $d(w,w')$ est généralement effectué en remarquant que :

$$d(ua,vb) = min \begin{cases} d(ua,v) & + & 1 \\ d(u,v) & + & \text{coût(substitution)} \\ d(u,vb) & + & 1 \end{cases}$$

où coût(substitution) vaut 0 si $a = b$, 1 sinon. L'algorithme 4 effectue ce calcul en $\mathcal{O}(|w| \cdot |w'|)$ par programmation dynamique [Wagner and Fischer, 1974]. Son but est de remplir une matrice M de telle façon que $M\big[|w|+1\big]\big[|w'|+1\big]$ contienne la distance d'édition $d(w,w')$, en calculant la distance d'édition entre chaque préfixe de w et w'.

Lors de la construction de la matrice M, il peut être utile de garder un pointeur sur les cases d'où provient le résultat de $M[i][j]$: de $M[i+1][j]+1$, ou de $M[i][j]+$coût(substitution) ou encore de $M[i][j+1]+1$. Ce pointeur nous dit quelle opération effectuer pour calculer la distance d'édition et, en partant de la dernière case de la matrice, ces pointeurs nous permettent de retrouver les opérations d'édition nécessaires pour passer d'un mot à l'autre.

4.2 Opérateur de généralisation pour les boules de mots

Dans le but d'utiliser le système VOLATA décrit à la section 2, nous cherchons un opérateur de généralisation pour les boules de mots, c'est-à-dire un algorithme prenant en entrée une boule et un mot, et fournissant une nouvelle boule. À nouveau nous commençons par nous interroger sur l'existence d'un opérateur MG, en sachant que cette fois les preuves d'apprenabilité disponibles pour cette famille ne passent pas par la proposition d'un tel opérateur [Tantini, 2009].

4.2.1 Existence d'un moindre généralisé unique ?

D'emblée, notons que dans cet espace les moindres généralisés sont multiples.

Algorithme 4 Algorithme général du calcul de la distance d'édition

fonction DistanceEdition
entrées : deux mots w et w'
sortie : $d(w, w')$

début
$M[0][0] = 0$
pour $i = 0$ à $|w|$ **faire**
  $M[i+1][0] = M[i][0] + 1$
fin pour
pour $i = 0$ à $|w'|$ **faire**
  $M[0][i+1] = M[0][i] + 1$
fin pour
pour $i = 0$ à $|w|$ **faire**
  pour $j = 0$ à $|w'|$ **faire**
    $M[i+1][j+1] =$

$$\min \begin{pmatrix} M[i+1][j] + 1 \\ M[i][j] + \text{coût(substitution)} \\ M[i][j+1] + 1 \end{pmatrix}$$

  fin pour
fin pour
renvoyer

$$d(w, w') = M\big[|w| + 1\big]\big[|w'| + 1\big]$$

fin

Le calcul de la distance d'édition entre $w =$ ABACD et $w' =$ EAFCGD par cet algorithme produit la matrice M ci-contre.

On en déduit que la distance entre les deux mots est de $d(\text{ABACD}, \text{EAFCGD}) = 4$. De plus, si l'on a mémorisé les étapes de calcul, on peut retrouver les différents chemins permettant de passer d'un mot à l'autre. On peut par exemple aligner les mots de la façon suivante :

$$\begin{matrix} - & A & B & A & C & D \\ E & A & F & C & G & D \end{matrix}$$

Les opérations nécessaires pour passer de ABACD à EAFCGD sont donc l'insertion d'un E et les substitutions d'un B par un F, d'un A par un C, et d'un C par un G. Toutefois, on trouve que l'alignement suivant est lui aussi possible :

$$\begin{matrix} - & A & B & A & C & - & D \\ E & A & F & - & C & G & D \end{matrix}$$

Les opérations de réécriture sont alors différentes : insérer un E et un G, supprimer un A et substituer un B par un F.

		E	A	F	C	G	D
	0	1	2	3	4	5	6
A	1	1	1	2	3	4	5
B	2	2	2	2	3	4	5
A	3	3	2	3	3	4	5
C	4	4	3	3	3	4	5
D	5	5	4	4	4	4	4

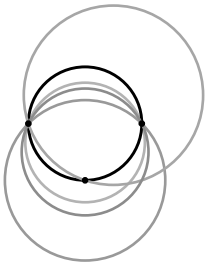


FIGURE 1 – Infinité de cercles contenant trois points et incomparables entre eux

Exemple 2 *Soient $E = [a, b, ab]$, $h = B_1(a)$ et $h' = B_1(b)$. Les deux hypothèses subsument bien les trois exemples ($h \succeq E$ et $h' \succeq E$) mais $h' \not\succeq h$ et $h \not\succeq h'$.*

Cela est intuitif dans le plan : la figure 1 exhibe trois points du plan et plusieurs hypothèses-cercles couvrant ces points mais incomparables entre elles. Elle nous convainc ainsi que nous pouvons avoir une multitude d'hypothèses moindres généralisées (une infinité dans le cas du plan). En d'autres termes, nous ne sommes manifestement pas dans le cadre idéal évoqué à la section 2.2 et dans lequel on aurait l'unicité du moindre généralisé.

Certains pourraient cependant souligner l'existence d'une *plus petite boule* (de rayon minimal) contenant les exemples à généraliser, comme cela est visible sur la figure 1 (en noir). C'est l'occasion pour nous de mettre en avant que cette notion de *plus petite boule* ne se confond pas avec celle de *moindre généralisé* : aussi petite soit-elle, cette hypothèse n'est pas contenue dans les autres moindres généralisés. Autrement dit, même si la *plus petite boule* contient bien les exemples, s'il existe une boule plus grosse contenant les exemples mais incomparable avec la *plus petite boule*, les deux boules sont des *moindres généralisés*. Et il n'y a pas de raison particulière pour que cette *plus petite boule* soit, plus qu'une autre, l'amorce du concept cible. De plus, dans le cadre qui nous intéresse, cette piste est bloquée par la complexité du calcul de la boule de rayon minimum. En effet, le théorème suivant nous indique que trouver le centre de la plus petite boule contenant un ensemble fini de mots est NP-difficile [de la Higuera and Casacuberta, 2000].

Théorème 3 *Étant donnés un ensemble fini de mots $W = \{w_1, \ldots, w_n\}$ et une constante K, le problème consistant à décider si un mot $z \in \Sigma^*$ existe tel que $\sum_{w \in W} d(z, w) < K$ (respectivement $\max_{w \in W} d(z, w) < K$) est NP-complet.*

En conclusion, même si la plus petite boule était un moindre généralisé unique, son calcul prendrait un temps exponentiel.

4.2.2 Proposition d'un opérateur GBALL

Algorithme 5 Méthode de calcul d'une boule généralisée

FONCTION GBALL

ENTRÉES : $e \in \mathcal{E}$ un exemple, $h = B_r(o) \in \mathcal{H}$ une hypothèse.

SORTIE : $g \in \mathcal{H}$ une généralisation de e et h ($g \succeq h$ et $g \succeq e$).

1: **début**
2: $c = o \xrightarrow{*} e$ { *chemin de longueur minimale* }
3: $u = \text{auHasardDans}(c)$ { $c = o \xrightarrow{x} u \xrightarrow{y} e$, $x + y = d(o, e)$ }
4: $x = d(o, u)$
5: $y = d(u, e)$
6: $k = \max(x + r, y)$
7: **renvoyer** $B_k(u)$
8: **fin**

Face à ces difficultés, nous proposons l'algorithme incrémental 5, noté GBALL, en guise d'opérateur de généralisation.

Son principe est de construire un *chemin d'édition* entre le centre de la boule à généraliser et le mot à ajouter, puis à choisir aléatoirement le centre de la nouvelle boule sur ce chemin et enfin à en déduire le nouveau rayon. Notons que les propriétés de cet algorithme sont conservées à l'identique si l'on utilise une stratégie quelconque pour choisir le nouveau centre sur le chemin d'édition : minimisation du nouveau rayon, au plus près de l'ancien centre proportionnellement au nombre de mots déjà généralisés, avantage donné aux exemples de plus grande longueur, heuristique propre au problème traité, *etc.* Le lecteur intéressé par une discussion sur ces stratégies se référera à [Tantini et al., 2010]. Dans la suite du présent article, nous considérons uniquement le tirage aléatoire du nouveau centre parmi les mots présents sur le chemin d'édition entre l'ancien centre et le nouvel exemple. Ce choix nous permet de gagner en rapidité de calcul et nous fournit une nouvelle source de diversité des boules produites.

Cet algorithme est in-monotone car une nouvelle boule produite inclut bien l'hypothèse précédente et le nouvel exemple :

— $g \succeq e$, en effet $d(u, e) = y \leq k$ et on a donc bien $e \in B_k(u)$;

— $g \succeq h$, en effet, pour tout mot $w \in B_r(o)$ on a $d(o, w) \leq r$; de plus, par l'inégalité triangulaire, $d(u, w) \leq d(u, o) + d(o, w)$ et ainsi $d(u, w) \leq x + r \leq k$; donc $w \in B_k(u)$.

Comme son exécution repose principalement sur le calcul du chemin entre le centre et le nouvel exemple, la complexité de cet algorithme est la même que celle de l'algorithme 4, c'est-à-dire polynomiale en la longueur des mots. Malheureusement, la contrepartie de cette monotonie et de ce gain en complexité est que la boule calculée n'est pas nécessairement une hypothèse moindre généralisée.

Exemple 4 *Soit $E = [a, b]$. La première hypothèse est le premier exemple, c'est-à-dire $h = B_0(a)$. Ensuite, comme le chemin entre le centre et le nouvel exemple b vaut $c : a \xrightarrow{1} b$, il existe deux possibilités pour u et finalement soit $\textsc{gball}(h, b) = B_1(a)$, soit $\textsc{gball}(h, b) = B_1(b)$. Or la boule unité centrée sur le mot vide $B_1(\lambda)$ contient bien E et est plus spécifique que les hypothèses calculées puisque $B_1(\lambda) \subseteq B_1(a)$ et $B_1(\lambda) \subseteq B_1(b)$.*

Cependant, bien que notre algorithme ne retourne pas toujours une hypothèse moindre généralisée, la complexité combinatoire des boules de mots nous empêche de fournir une meilleure construction.

Point positif pour nos méthodes d'ensemble, le résultat de GBALL est lié à l'ordre de présentation des exemples et cela indépendamment du tirage aléatoire du nouveau centre. Ainsi, si nous considérons les mots a et b et que nous supposons que le tirage conserve systématiquement l'ancien centre ($x = 0$), nous avons :

$$\textsc{gball}(B_0(a), b) = B_1(a) \neq \textsc{gball}(B_0(b), a) = B_1(b)$$

Par ailleurs, GBALL n'est pas ex-monotone de comme l'illustre l'exemple suivant.

Exemple 5 *En effet, si l'on suppose un tirage aléatoire qui fournisse systématiquement le nouveau centre à une distance de 1 de l'ancien centre ($x = 1$), si les exemples sont λ, b, a et si le seul contre-exemple est bb, alors les hypothèses successivement produites sont :*

— *$B_0(\lambda)$ est l'hypothèse initiale ;*

— *$B_1(b)$ est rejetée car elle couvre bb ;*

— *$B_1(a)$ est acceptée.*

Cette dernière couvre b alors que l'ajout de ce mot a été rejeté à l'étape précédente.

La seule conséquence de cette non-monotonie est que les implémentations des algorithmes comme AdaBoost seront moins efficaces puisque la couverture des hypothèses devra être systématiquement calculée.

Il est difficile d'évaluer la granularité de notre opérateur ; par contre, le résultat suivant (fourni par Jean-Christophe Janodet) sur la VC-dimension des boules peut alimenter notre réflexion.

Théorème 6 *Pour $|\Sigma| = 2$, la VC-dimension des boules est infinie.*

Preuve (Janodet, 2010). Nous prenons n mots, tous de longueur n, définis comme suit : le i^{e} mot n'est constitué que de `a` à l'exception de la i^{e} lettre qui est un `b`. Considérons maintenant que ces mots sont étiquetés : k positivement, $(n-k)$ négativement. On peut construire une boule qui ne capture que les positifs comme suit : le centre est le mot de longueur n qui a des `b` aux mêmes positions que les positifs et des `a` ailleurs, le rayon vaut $(k-1)$. Par construction, les positifs sont accessibles à partir du centre après $(k-1)$ substitutions, tandis que les négatifs sont à une distance strictement plus grande. □

Ce résultat indique un certain pouvoir discriminant des boules de mots : pour l'échantillon décrit dans la preuve, et quel que soit l'étiquetage choisi pour ces exemples, il existe une boule qui sépare exemples et contre-exemples. De plus, remarquons que la boule construite par cette preuve contient plus de mots que ceux de l'échantillon (signe qu'il y a eu généralisation et pas apprentissage par cœur) et que les mots sont placés à la périphérie de la boule (indiquant que la boule apprise reste relativement spécifique aux exemples de l'échantillon).

En résumé, GBALL n'avance pas par plus petits pas (exemple 4) mais produit des hypothèses correctes et est sensible à l'ordre de présentation des exemples (cette dépendance s'ajoute donc à celle, naturelle, de GC), GBALL est in-monotone mais pas ex-monotone et GC ne produit pas nécessairement des hypothèses maximales (exemple 5), $(\mathcal{H}, \succeq)$ a une granularité et une diversité intéressantes (VC-dimension infinie sans apprentissage par cœur).

Avant de passer aux expérimentations, il nous reste à préciser quelques points de notre implémentation de l'algorithme GBALL.

4.3 Compléments sur le calcul de la nouvelle boule

Dans l'algorithme 5, c est un chemin d'édition entre o et e. Or, comme nous l'avons vu avec l'exemple 1 et l'illustration de l'algorithme 4, il existe souvent plusieurs façons de passer d'un mot à l'autre. Le calcul de ce chemin est effectué en partant de la dernière case de la matrice et en remontant suivant les origines de la valeur de cette case.

Afin de toujours calculer le même chemin $c = o \xrightarrow{*} e$, nous avons effectué les choix arbitraires suivants :

— si le calcul de $M[i][j]$ peut provenir d'une suppression ou d'une autre opération, nous choisissons la suppression ;
— si le calcul de $M[i][j]$ peut provenir d'une insertion ou d'une substitution, nous privilégions l'insertion ;
— une fois les opérations d'édition trouvées, nous effectuons la réécriture de gauche à droite, c'est-à-dire dans le sens de lecture.

Fixer de tels choix permet alors à l'algorithme GC d'être dépendant de l'ordre des exemples, tout en assurant une certaine reproductibilité des expériences.

Exemple 7 *Soient les mots $w = ABACD$ et $w' = EAFCGD$. La table suivante contient le calcul des distances d'édition $d(ABACD, EAFCGD)$ et $d(EAFCGD, ABACD)$, à gauche la matrice du calcul des chemins d'édition $ABACD \xrightarrow{*} EAFCGD$, à droite celle pour $EAFCGD \xrightarrow{*} ABACD$.*

		E	A	F	C	G	D
	0	1	2	3	4	5	6
A	1	1	1	2	3	4	5
B	2	2	2	2	3	4	5
A	3	3	2	3	3	4	5
C	4	4	3	3	3	4	5
D	5	5	4	4	4	4	4

		A	B	A	C	D
	0	1	2	3	4	5
E	1	1	2	3	4	5
A	2	1	2	2	3	4
F	3	2	2	3	3	4
C	4	3	3	3	3	4
G	5	4	4	4	4	4
D	6	5	5	5	5	4

Les cases grisées retracent la sélection des opérations d'édition pour le calcul du chemin d'édition telle que défini précédemment. En appliquant les opérations de gauche à droite dans le mot, on obtient alors les chemins d'édition suivants :

$$\underline{\cdot}ABACD \rightarrow EA\underline{B}ACD \rightarrow EAF\underline{A}CD \rightarrow EAFC\underline{\cdot}D \rightarrow EAFCGD$$
$$\underline{E}AFCGD \rightarrow A\underline{\cdot}AFCGD \rightarrow ABA\underline{F}CGD \rightarrow ABAC\underline{G}D \rightarrow ABACD$$

Notre méthode étant décrite, nous pouvons passer à son évaluation expérimentale.

5 Expérimentations

Il s'agit ici d'évaluer les associations de GLOBOOST (décrit à l'algorithme 1) avec nos différents opérateurs de généralisation dédiés aux séquences, en suivant le protocole suivant : validation croisée 10 fois et GLOBOOST exécuté 10 fois sur chaque bloc de validation croisée (ainsi, un résultat donné pour GLOBOOST correspond à une moyenne sur 100 exécutions). Nous avons mis en place deux groupes d'expériences :

— le premier utilise des problèmes classiques dont la résolution n'est pas un objectif en lui-même, il s'agit simplement là de dégager des caractéristiques de notre approche et de constater si celle-ci est raisonnablement performante ;
— le second porte sur un problème réel et, dans ce cas, nous comparons notre meilleure méthode repérée pendant la première phase d'expériences à la meilleure méthode décrite dans la littérature.

5.1 Données de l'UCI Repository

Nous utilisons dans ces expérimentations les jeux de données présents dans l'*UCI Repository* [Frank and Asuncion, 2010] et indiqués comme séquentiels, à savoir :

— `tic-tac-toe` : fins de parties au jeu du morpion réparties en deux classes : celles qui voient la victoire des croix et les autres (victoires des ronds et matches nuls), les cases du jeu sont représentées de manière séquentielle par leur contenu, l'alphabet est de taille 3 (x pour une croix, o pour un rond, b pour un blanc) et tous les exemples sont donc de taille 9 ;
— `badges` : les noms des participants à une conférence répartis en deux classes (ceux qui ont une voyelle en deuxième position et les autres), l'alphabet est composé de 30 caractères et la longueur moyenne des exemples vaut 14 ;
— `us-first-name` : les prénoms américains, il s'agit d'apprendre à distinguer les prénoms féminins des masculins, l'alphabet compte 26 lettres, la longueur moyenne des exemples vaut 6 ;
— `promoters` et `splice` (dans sa version à deux classes) : jeux de données génomiques utilisant un alphabet à 4 lettres (A, T, C, G), les séquences sont chacune de taille 60.

Notons que ces jeux de données peuvent être traités par des méthodes attributs-valeurs. Les données `promoters` et `splice` ont été traitées à cette fin : les séquences ont été alignées et sont de même longueur. La tâche `tic-tac-toe` peut être directement considérée

en attributs-valeurs. Pour les autres problèmes, les exemples peuvent être arbitrairement alignés par la gauche, et complétés pour être tous de même longueur. Notons que les méthodes attributs-valeurs sont alors favorisées : elles peuvent traiter les données « colonne par colonne » sans affronter la complexité due à la séquence, et bénéficier des alignements qui, sur ces jeux de données, permettent la découverte d'un concept exprimable en attributs-valeurs. Par exemple dans le cas des données `badges`, un alignement à gauche permet de poser aisément une condition sur la deuxième lettre qui est simplement le deuxième attribut, par contre une telle condition ne peux s'exprimer par ni par un automate k-TSS, ni par une boule de mots.

Sur ces données, nous mettons en œuvre GLOBOOST instancié par nos différents opérateurs de généralisation. En particulier, MGTSSI exige de fixer une valeur k pour fonctionner : nous avons déterminé ces valeurs par validation croisée sur les ensembles d'apprentissage et finalement k varie entre 2 et 5 dans nos expérimentations. Nous ajoutons l'algorithme GLOBO qui utilise les mêmes opérateurs mais qui n'est pas une méthode d'ensemble.

Nous comparons ces algorithmes à des méthodes classiques en classification de séquences : RPNI [Oncina and García, 1992], TRAXBAR [Lang, 1992] et RED-BLUE [Lang et al., 1998] de la famille EDSM (*Evidence Driven State Merging*). Tous sont capables d'apprendre à partir d'exemples et de contre-exemples. RPNI et TRAXBAR nécessitent de fixer une classe cible : pour chaque problème, nous les avons exécutés successivement chaque classe servant à son tour de cible, et retenu uniquement le meilleur résultat. Enfin, nous évaluons des méthodes propres à l'attribut-valeur puisque nous avons vu que les jeux de données pouvaient s'interpréter dans ce formalisme : C4.5, du *boosting de stumps* et nos propres algorithmes d'apprentissage (GLOBO, GLOBOOST et ADABOOST-MG) instanciés avec un moindre généralisé dédié à l'attribut-valeur (il s'agit d'un calcul de plus petit rectangle).

Le protocole décrit donne à chaque méthode 90 % des données pour apprendre, tandis que les 10 % restants sont gardés pour le test.

Le tableau 1 présente les résultats. Nous constatons que globalement nos méthodes se comportent aussi bien ou mieux que les algorithmes issus de l'inférence grammaticale. La supériorité sur les méthodes attributs-valeurs est moins claire mais cela s'explique comme nous l'avons indiqué par les préalignements utilisés qui leur facilitent la tâche. Concernant GLOBO, nos expériences démontrent qu'il est plus intéressant pour la prédiction d'avoir un grand nombre d'hypothèses.

Concernant plus spécifiquement les combinaisons d'automates, les résultats font apparaître une granularité trop faible, à la limite de l'apprentissage par cœur, et un manque de diversité : ce sont rapidement les mêmes automates qui sont produits et ils sont très spécifiques. Il n'est pas envisageable de multiplier le nombre d'hypothèses produites dans le cas des automates 0-réversibles ou k-TSS pour une autre raison : les algorithmes de généralisation des automates sont trop longs pour fournir autant d'hypothèses rapidement, sans doute à cause des opérations complexes qui sont mises en œuvre (fusion d'états, déterminisation, *etc.*).

Par contre, nous pouvons produire 1 000, 10 000 et 100 000 boules sur chaque problème considéré : les boules se révèlent très diverses et rapides à calculer (les opérations sur les mots comme le calcul de distance sont assez simples). Nous notons que la qualité des prédictions augmente avec le nombre de boules produites et finalement, nos combinaisons de boules sont plus que compétitives en termes de prédiction que nos combinaisons d'automates. Les exemples suivants permettent de constater que les concepts représentés par des boules et découverts par nos méthodes sont très différents de ce l'on découvre sous forme d'attributs-valeurs ou d'automates.

Exemple 8 *Sur la base* `tic-tac-toe`*, nous apprenons la boule de rayon 5 et de centre* `bbbb` *(qui ne représente aucune partie de morpion valide). Elle ne contient aucun négatif mais couvre 120 positifs, tous à distance 5 du centre :* `xxxoobbbb`*,* `xobxbbxbo`*,* `xbboxbobx`*,* `obxbbxobx`*,* `boxoxbxbb`*,* `bbxobxobx`*,* etc.

TABLE 1 – Précisions expérimentales, G_x (respectivement A_x et $GLOBO_x$) dénote GLOBOOST (respectivement ADABOOST et GLOBO) instancié par l'apprenant X

	tic-tac-toe	badges	promoters	first-name	splice
Majorité	65.34 %	71.43 %	50.00 %	81.62 %	50.26 %
C4.5	85.60 %	98.63 %	81.83 %	86.24 %	95.41 %
A_{STUMPS} ×1 000	98.33 %	100.00 %	81.33 %	84.48 %	96.08 %
$GLOBO_{MGRECT}$	98.89 %	97.22 %	77.40 %	86.93 %	93.68 %
A_{MGRECT} ×1 000	100.00 %	98.98 %	93.00 %	87.15 %	97.02 %
G_{MGRECT} ×1 000	99.86 %	96.33 %	93.97 %	86.92 %	96.19 %
RPNI	91.13 %	62.24 %	-	81.42 %	-
TRAXBAR	90.81 %	57.48 %	56.60 %	81.37 %	**58.33 %**
RED-BLUE	**93.89 %**	61.09 %	**63.02 %**	**82.83 %**	54.65 %
$GLOBO_{MGTSSI}$	81.43 %	72.24 %	60.00 %	83.83 %	73.42 %
$GLOBO_{MGZR}$	**95.84 %**	71.43 %	50.00 %	81.97 %	-
$GLOBO_{GBALL}$	89.32 %	**74.63 %**	**71.79 %**	**87.08 %**	**87.85 %**
G_{MGTSSI} ×1 000	91.47 %	**72.69 %**	**61.13 %**	**89.50 %**	**78.07 %**
G_{MGZR} ×1 000	**98.36 %**	71.43 %	50.00 %	83.07 %	-
G_{GBALL} ×1 000	92.62 %	80.41 %	87.63 %	87.10 %	93.76 %
G_{GBALL} ×10 000	94.69 %	**81.39** %	88.43 %	88.80 %	**95.63 %**
G_{GBALL} ×100 000	**94.90 %**	81.36 %	**89.08 %**	**89.06 %**	**95.63 %**

	NIST special database 3
SEDiL	95.86 %
G_{GBALL} ×1 000	93.81 %
G_{GBALL} ×10 000	95.93 %
G_{GBALL} ×100 000	**96.32 %**

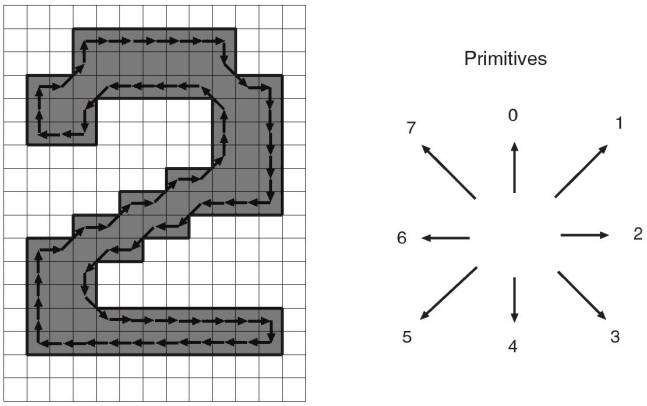


FIGURE 2 – Codage d'un chiffre « 2 », la chaîne équivalente vaut « 22222432444446 665656543222222246666666666600002121212100766666546600210 »

Autre observation visible sur cet exemple et sur le suivant : pour chaque série d'expérimentations, les exemples se placent sur les bords des boules apprises, le centre n'est jamais un exemple du concept.

Exemple 9 *Sur la base* `us-first-name`*, la boule de centre* `LRLRTSVKCA` *et de rayon 7 est apprise : elle couvre 346 prénoms féminins et pas un masculin. Le centre n'a pas de sens pour le problème considéré et les exemples couverts sont à la périphérie.*

Bien que cela puisse s'expliquer par notre méthode de construction des hypothèses et par les propriétés intrinsèques des boules de mots, cela reste néanmoins remarquable. En effet, lorsque les boules sont utilisées dans le cadre d'un apprentissage en situations bruitées [Tantini et al., 2006], le centre est généralement une donnée non bruitée, et le rayon est considéré comme un degré de tolérance au bruit. Ici, le centre de l'hypothèse finale est plutôt à considérer comme une chaîne moyenne ou médiane des exemples positifs. Par ailleurs, nous pouvons noter que dans $\mathbb{R}^n$ une boule avec les points de l'échantillon placés à la frontière serait indubitablement une boule moindre généralisée de ces points. Et pour terminer sur la diversité et la structure des boules, rappelons le résultat de VC-dimension infinie (théorème 6) : il éclaire la diversité des boules observées et sa preuve utilise elle aussi une boule creuse.

En conclusion de ces premières expériences, les combinaisons de boules de mots sont les meilleures candidates pour affronter un problème réel, en particulier dans les cas où il n'y a pas d'alignement évident qui permette aux méthodes attributs-valeurs de s'appliquer.

5.2 Reconnaissance de chiffres manuscrits

Nous considérons également la base de données *NIST special database 3* qui est constituée d'images *bitmap* 128×128 de chiffres et de lettres manuscrits. Nous nous concentrons sur un sous-ensemble de chiffres, écrits par 100 scribes différents et pouvant être tracés « en une ligne » (l'ensemble des pixels étant ainsi connexe). Chaque classe (chiffres de 0 à 9) a environ 1 000 instances, pour un corpus de 10 568 chiffres.

Comme nous travaillons sur des mots, chaque image est transformée en chaîne octale suivant un algorithme décrit par [Micó and Oncina, 1998] : à partir du pixel le plus en haut à gauche, nous suivons la frontière jusqu'à retomber sur le premier pixel. Durant son chemin, l'algorithme construit une chaîne incluant la direction du prochain pixel sur la frontière (la figure 2 donne un exemple de chiffre ainsi codé). Ce codage utilise un alphabet de taille 8 et produit des séquences dont la longueur moyenne vaut 76 et au maximum 200 caractères.

Pour ce corpus, nous comparons notre méthode à celle de [Oncina and Sebban, 2006] en utilisant le protocole de classification de la plateforme SEDiL [Boyer et al., 2008]. Pour

cela, nous utilisons une matrice d'édition construite par Marc Sebban, selon la méthode décrite dans [Oncina and Sebban, 2006]. Elle est apprise à l'aide d'un transducteur stochastique sur 8 000 paires de mots, une paire consistant en une chaîne et son plus proche voisin (soit environ 80 % des données disponibles utilisées en apprentissage par SEDiL). L'étiquetage final, réalisé par SEDiL, consiste alors à attribuer la classe du plus proche voisin, les distances d'édition étant calculées grâce à la matrice d'édition pondérée apprise. Notons que cette phase de classification par SEDiL nécessite un nombre quadratique de calculs de distance entre mots.

Pour nos algorithmes, nous avons conservé notre protocole habituel, à savoir une validation croisée 10 fois. Cependant, par souci de rapidité, les apprentissages se font cette fois sur 10 % des données, les tests sur les autres 90 %. SEDiL a été évalué sur les mêmes exemples de test, ce qui indubitablement induit un biais en sa faveur puisque bon nombre des mots présents dans le test ont été vus en apprentissage par SEDiL.

Le tableau 1 donne les résultats obtenus. À partir de 10 000 boules combinées, nous atteignons le niveau de performance de SEDiL et le dépassons avec des combinaisons de 100 000 boules, malgré un protocole qui nous est défavorable.

6 Conclusion

Notre recherche s'inscrit dans la lignée des travaux qui cherchent à bénéficier des méthodes d'ensemble en y embarquant des apprenants issus de l'inférence grammaticale [Janodet et al., 2004, García et al., 2010]. Une différence notable est que ces approches proposent des modifications *ad hoc* de la méthode d'ensemble ou des modifications *ad hoc* de l'apprenant de base.

A contrario, nous avons choisi VOLATA, un cadre générique ayant déjà fait ses preuves, qui nous permet de décliner plusieurs méthodes d'apprentissage de haut niveau et d'explorer différents opérateurs de généralisation dédiés aux séquences. Nous avons montré ainsi que les travaux théoriques sur l'apprentissage à la limite de langages par positifs seuls peuvent enrichir des techniques d'apprentissage supervisé. Nous avons également établi que lorsque cet apport théorique fait défaut, il est encore possible d'utiliser nos méthodes, avec la proposition d'un opérateur de généralisation adéquat.

Les résultats expérimentaux valident cette démarche : les combinaisons d'automates permettent effectivement de classer des séquences et les combinaisons de boules se révèlent meilleures que les autres méthodes, en particulier sur un problème réel de reconnaissance de caractères. Ces bons résultats pour les boules peuvent sembler étonnants au regard de la relative pauvreté d'une boule qui ne dénote qu'un langage fini mais nous avons relativisé cette apparente pauvreté par la VC-dimension infinie des boules de mots.

Finalement, nous avons été capables d'intégrer des hypothèses à moindres généralisés multiples dans le système VOLATA, dédié jusque là au moindre généralisé unique. Cela ouvre le champ des classes d'hypothèses pouvant intégrer ce système et encourage à l'explorer. Nous pourrions ainsi tester dans notre cadre l'heuristique ECGI proposée comme opérateur de généralisation pour une classe d'automates à moindres généralisés multiples [Prieto and Vidal, 1992]. Concernant les boules de mots, nos méthodes pourraient s'améliorer par l'apprentissage des coûts des opérations d'édition, apprentissage spécifique à chaque problème. Enfin, il nous reste à explorer les possibilités de VOLATA : variations autour de GC pour gérer des données bruitées et remplacement de GLOBOOST par ADABOOST-MG pour bénéficier d'un véritable algorithme de *boosting*.

Remerciements. Merci à Jean-Christophe Janodet pour le résultat sur la VC-dimension des boules de mots. Merci à Marc Sebban pour les échanges sur la reconnaissance de caractères et SEDiL. Merci aux relecteurs de CAp'2010 et de RIA pour leurs commentaires.

Références

[Angluin, 1982] Angluin, D. (1982). Inference of reversible languages. *Journal of the ACM*, 29(3) :741–765.

[Bourgne et al., 2010] Bourgne, G., Soldano, H., and Fallah-Seghrouchni, A. E. (2010). Learning better together. In Coelho, H., Studer, R., and Wooldridge, M., editors, *ECAI 2010*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 85–90. IOS Press.

[Boyer et al., 2008] Boyer, L., Esposito, Y., Habrard, A., Oncina, J., and Sebban, J. (2008). Sedil : Software for Edit Distance Learning. In Daelemans, W., Goethals, B., and Morik, K., editors, *Proceedings of the 19th European Conference on Machine Learning*, pages 672–677. Springer.

[de la Higuera and Casacuberta, 2000] de la Higuera, C. and Casacuberta, F. (2000). Topology of strings : median string is NP-complete. *Theoretical Computer Science*, 230 :39–48.

[De Raedt and Bruynooghe, 1993] De Raedt, L. and Bruynooghe, M. (1993). A theory of clausal discovery. In Bajcsy, R., editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1058–1063. Morgan Kaufmann.

[Feigenbaum, 1977] Feigenbaum, E. A. (1977). The art of artificial intelligence : Themes and case studies of knowledge engineering. In Reddy, R., editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 1014–1029. Morgan Kaufmann.

[Frank and Asuncion, 2010] Frank, A. and Asuncion, A. (2010). UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, `http://archive.ics.uci.edu/ml`.

[Ganascia, 1993] Ganascia, J.-G. (1993). Algebraic structure of some learning systems. In *ALT '93 : Proceedings of the 4th International Workshop on Algorithmic Learning Theory*, pages 398–409, London, UK. Springer-Verlag.

[García et al., 2010] García, P., de Parga, M. V., López, D., and Ruiz, J. (2010). Learning automata teams. In *Proceedings of the 10th international colloquium conference on Grammatical inference : theoretical results and applications*, ICGI'10, pages 52–65, Berlin, Heidelberg. Springer-Verlag.

[García and Vidal, 1990] García, P. and Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(9) :920–925.

[Janodet et al., 2004] Janodet, J.-C., Nock, R., Sebban, M., and Suchier, H.-M. (2004). Boosting grammatical inference with confidence oracles. In Greiner, editor, *International Conference on Machine Learning (ICML04)*, pages 425–432.

[Kuncheva and Whitaker, 2003] Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2) :181–207.

[Lang, 1992] Lang, K. J. (1992). Random dfa's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52. ACM Press.

[Lang et al., 1998] Lang, K. J., Pearlmutter, B. A., and Price, R. A. (1998). Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *ICGI '98 : Proceedings of the 4th International Colloquium on Grammatical Inference*, pages 1–12, London, UK. Springer-Verlag.

[Levenshtein, 1965] Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4) :845–848.

[Micó and Oncina, 1998] Micó, L. and Oncina, J. (1998). Comparison of fast nearest neighbour classifiers for handwritten character recognition. *Pattern Recognition Letter*, 19(3-4) :351–356.

[Oncina and García, 1992] Oncina, J. and García, P. (1992). Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108. World Scientific Publishing.

[Oncina and Sebban, 2006] Oncina, J. and Sebban, M. (2006). Learning stochastic edit distance : Application in handwritten character recognition. *Pattern Recognition*, 39(9) :1575–1587.

[Plotkin, 1970] Plotkin, G. (1970). A note on inductive generalization. In Meltzer, B. and Mitchie, D., editors, *Machine Intelligence*, volume 5, pages 153–165. Edinburgh University Press.

[Prieto and Vidal, 1992] Prieto, N. and Vidal, E. (1992). Learning language models through the ecgi method. *Speech Communication*, 11(2-3) :299–309.

[Tantini, 2009] Tantini, F. (2009). *Inférence grammaticale en situations bruitées*. PhD thesis, Université Jean Monnet de Saint-Étienne.

[Tantini et al., 2006] Tantini, F., de la Higuera, C., and Janodet, J.-C. (2006). Identification in the limit of systematic-noisy languages. In *Proceedings of the 9th International Conference in Grammatical Inference*, pages 19–31.

[Tantini et al., 2010] Tantini, F., Terlutte, A., and Torre, F. (2010). Sequences classification by least general generalisations. In Sempere, J. M., editor, *10th International Colloquium on Grammatical Inference (ICGI 2010)*, pages 189–202, Valencia (Spain). Springer-Verlag.

[Torre, 2000] Torre, F. (2000). *Intégration des biais de langage à l'algorithme générer-et-tester - Contributions à l'apprentissage disjonctif*. PhD thesis, Laboratoire de Recherche en Informatique Université Paris-Sud France.

[Torre, 2005] Torre, F. (2005). Globoost : Combinaisons de moindres généralisés. *Revue d'Intelligence Artificielle*, 19(4-5) :769–797.

[Torre and Rouveirol, 1997] Torre, F. and Rouveirol, C. (1997). Natural ideal operators in inductive logic programming. In van Someren, M. and Widmer, G., editors, *9th European Conference on Machine Learning (ECML'97)*, volume 1224 of *LNAI*, pages 274–289. Springer-Verlag.

[van der Laag and Nienhuys-Cheng, 1994] van der Laag, P. R. J. and Nienhuys-Cheng, S. (1994). Existence and nonexistence of complete refinement operators. In Bergadano, F. and de Raedt, L., editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag.

[Wagner and Fischer, 1974] Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM*, 21 :168–178.

[Webb and Agar, 1992] Webb, G. I. and Agar, J. W. M. (1992). Inducing diagnostic rules for glomerular disease with the DLG machine learning algorithm. *Artificial Intelligence in Medicine*, 4 :419–430.