

GLOBOOST: Boosting de Moindres Généralisés

Fabien Torre

GRAppA - Université Charles de Gaulle - Lille 3

Résumé : Nous explorons dans cet article l'utilisation de moindres généralisés corrects comme apprenant faible dans des techniques de *boosting* (Freund & Schapire, 1996).

Nous établissons dans un premier temps que cette idée est pertinente : les expérimentations sur des problèmes classiques de l'UCI (Blake & Merz, 1998) montrent qu'ADABOOST instancié avec un apprenant faible à base de moindres généralisés obtient des taux d'erreur plus faibles que C4.5, GLOBO (Torre, 1999) (système utilisant les moindres généralisés mais sans *boosting*) et ADABOOST muni d'un apprenant faible plus classique.

Puis, devant la constatation que notre nouvel apprenant faible peut être coûteux en temps de calcul, nous proposons un mode de génération des hypothèses qui peut être distribué sur différentes machines et une définition du poids des hypothèses *a posteriori*. Cela aboutit au nouvel algorithme GLOBOOST et, à nouveau, les expérimentations nous encouragent dans cette voie : GLOBOOST obtient des performances comparables à celles d'ADABOOST.

Nous discutons finalement des résultats obtenus, de leur signification vis-à-vis du *boosting* et des perspectives ouvertes.

Introduction

Le *boosting* est une technique qui a des origines théoriques ancrées dans le modèle PAC (Valiant, 1984). Dans ce cadre, un résultat fort a été montré : tout algorithme d'apprentissage produisant des hypothèses faisant mieux qu'une procédure de choix aléatoire peut être *boostée* jusqu'à atteindre une erreur aussi faible que voulue et ce avec une probabilité aussi grande que désirée et en temps polynomial.

Deux preuves de ce résultat ont été proposées (Schapire, 1990; Freund, 1995) ; elles ont pour point commun de définir chacune un algorithme réalisant le *boosting* d'un apprenant faible, toujours dans le cadre PAC. L'idée générale de ces algorithmes est d'utiliser plusieurs fois l'apprenant faible sur des distributions différentes, puis de combiner les différentes hypothèses obtenues au sein d'un vote.

L'algorithme utilisé dans (Freund, 1995) a de plus la particularité de se rapprocher d'un cadre plus réaliste que le cadre PAC (par exemple, en constituant son échantillon au tout début de l'algorithme). Cette démarche a finalement conduit à l'algorithme ADABOOST (Freund & Schapire, 1995; Freund & Schapire, 1997). Son principe est d'associer un poids à chaque exemple, puis de solliciter l'apprenant faible sur ces exemples et ces poids pour obtenir une hypothèse qui classe correctement les exemples de poids forts ; ensuite, les poids des exemples sont modifiés

en fonction de cette hypothèse (les poids des bien classés sont diminués et ceux des mal classés augmentés) et le processus itéré.

ADABOOST a permis de tester les idées du *boosting* sur des données réelles (Freund & Schapire, 1996), tout comme BOOSTEXTER la version dédiée à la classification de textes (Schapire & Singer, 2000). Ces expérimentations, et bien d'autres ensuite, démontrent qu'ADABOOST est capable d'atteindre des erreurs en généralisation très faibles.

Dans cet article, nous nous intéressons à un constructeur d'hypothèses qui, à notre connaissance, n'a jamais été utilisé comme apprenant faible dans un algorithme de *boosting* : le calcul de *moindres généralisés corrects*. Ces moindres généralisés ont montré leur capacité à capturer des sous-concepts, en particulier sur des problèmes hautement disjonctifs (Torre, 1999). Outre l'aspect prédictif, les moindres généralisés permettent la production de théories compréhensibles pour un expert humain.

Leur défaut majeur est le temps de calcul qu'ils nécessitent et la question se pose alors de pouvoir distribuer le calcul effectué par ADABOOST sur plusieurs machines. Or, comme nous l'avons signalé, ADABOOST produit une hypothèse en fonction des précédentes ce qui, *a priori*, empêche sa parallélisation.

Ce sont ces questions que nous traitons dans cet article dont le plan est le suivant. Nous revenons à la section 1 sur la technique du *boosting*, les questions qu'elle soulève et les éléments de réponse disponibles. À la section 2, nous rappelons ce qu'est un *moindre généralisé correct* (Torre, 1999) et discutons de son utilisation comme apprenant faible d'ADABOOST. La section 3 présente les résultats expérimentaux obtenus en suivant cette voie ainsi que les performances sur les mêmes données de systèmes plus classiques. Nous évoquons ensuite, section 4, la possibilité de remplacer la génération des hypothèses d'ADABOOST par une méthode de production plus facilement parallélisable et une pondération des hypothèses qui soit détachée de cette génération. À la section 5, nous montrons à nouveau des résultats expérimentaux. Enfin, à la section 6 nous dressons un bilan de ce travail et une liste des pistes ouvertes.

1 Quelques éléments de *boosting*

Il ne s'agit pas ici de donner une vision exhaustive des travaux sur le *boosting*, mais simplement de montrer que certaines questions relatives aux conditions et aux modes d'application de cette technique restent ouvertes.

Dans le cadre PAC (Valiant, 1984), une classe de concepts est dite *apprenable* s'il existe un algorithme capable, pour tout concept de cette classe et toute distribution des exemples, de fournir une hypothèse dont l'erreur par rapport au concept cible peut être aussi proche de 0 que voulu et ce avec une probabilité aussi proche de 1 que désiré. Précisons qu'en plus l'algorithme doit fournir l'hypothèse en temps polynomial dans les inverses de l'erreur et de la confiance choisies.

Cette notion d'apprenabilité est dite *forte*, par opposition à la version dite *faible* qui apparaît dans (Kearns & Valiant, 1989) : une classe de concepts est dite *faiblement apprenable* s'il existe un algorithme capable, pour tout concept de cette classe et toute distribution des exemples, de fournir une hypothèse dont l'erreur est strictement inférieure à $\frac{1}{2}$ avec une confiance non nulle. Autrement dit, il est simplement exigé que l'apprenant faible fournisse une hypothèse qui fasse mieux qu'un classifieur aléatoire.

Vient ensuite la question du lien entre ces deux notions d'apprenabilité et la réponse est surprenante : elles sont équivalentes. Une première preuve de ce résultat est donnée dans (Schapire, 1990) (voir (Kearns & Vazirani, 1994) pour une présentation limpide de cette preuve) puis une seconde dans (Freund, 1995). Chacune de ces deux preuves repose sur un algorithme auquel on fournit un apprenant faible et qui fait appel un nombre polynomial de fois à cet apprenant faible pour finalement produire une hypothèse forte. Dans les deux cas, cette hypothèse forte est un vote qui combine les hypothèses faibles produites par l'apprenant faible.

À la lumière de ces preuves, l'équivalence des deux notions d'apprenabilité revient à dire que, dans le cadre PAC, on est capable de *booster* un apprenant faisant à peine mieux que la hasard pour atteindre une erreur aussi petite que voulue.

Un algorithme pratique nommé ADABOOST et inspiré de la preuve de (Freund, 1995) a finalement été proposé (Freund & Schapire, 1995; Freund & Schapire, 1997). ADABOOST est présenté à l'algorithme 1 dans sa version à deux classes et avec un apprenant faible pouvant fournir des hypothèses qui s'abstiennent. Cette version est reprise de (Schapire & Singer, 1999) et définit pour chaque hypothèse trois quantités : W_+ la somme des poids des exemples bien classés, W_- la somme des poids des exemples mal classés et W_0 la somme des poids des exemples sur lesquels l'hypothèse s'abstient.

Algorithm 1 ADABOOST

Entrées : n exemples x_i et leurs classes $y_i \in \{-1, +1\}$; T un nombre d'itérations à effectuer ;

A un apprenant faible acceptant en entrée un échantillon d'exemples étiquetés et pondérés.

Sortie : H le classifieur final.

for $i = 1$ to n **do**

$w_i = 1/n$ {initialisation des poids des exemples}

end for

for $t = 1$ to T **do**

$h_t = A(\{(x_i, y_i, w_i)\})$

for $b \in \{-, 0, +\}$: $W_b = \sum_{i: \text{sign}(y_i h_t(x_i))=b} w_i$

$\alpha_t = \frac{1}{2} \log \left(\frac{W_+ + \frac{1}{2}W_0}{W_- + \frac{1}{2}W_0} \right)$

$Z_t = \sum_i [w_i \cdot \exp(-\alpha_t y_i h_t(x_i))]$

for $i = 1$ to n **do**

$w_i = \frac{w_i \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

end for

end for

return $H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$

Dans la réalité, il est difficile de réunir les conditions théoriques énoncées dans le cadre PAC pour que le *boosting* soit effectivement garanti :

- on n'est jamais certain de disposer d'assez d'exemples et dans la majorité des cas, on ne peut pas comme dans le cadre PAC en réclamer plus à un oracle ;
- on ignore si le nombre d'étapes de *boosting* choisi est suffisant ;
- on est rarement sûr de disposer d'un apprenant faible, c'est-à-dire d'un algorithme fournissant pour toute distribution une hypothèse faisant mieux que l'aléatoire.

De plus, dans la pratique, on pourrait s'attendre à une dégradation de l'erreur en généralisation au cours du *boosting*, un trop grand nombre d'étapes ayant pour effet de sur-spécialiser le système de vote.

Malgré toutes ces réserves, on constate de très bonnes performances d'ADABOOST sur des données réelles. Les conditions imposées par le cadre PAC et les théorèmes de *boosting* n'étant clairement pas remplies, il faut trouver une autre explication à ces bons résultats.

Dans (Breiman, 1996b), l'erreur d'un classifieur est décomposée en biais et variance puis il est avancé que, comme toute méthode de type *arcing* (*adaptively resample and combine*) et comme le *bagging* (Breiman, 1996a), ADABOOST diminuerait la composante due à la variance. Cet argument est battu en brèche dans (Freund & Schapire, 1998) où sont présentées des expérimentations où l'on voit ADABOOST faire chuter à la fois les deux composantes de l'erreur mais toujours sans explication du phénomène.

Une observation sur les algorithmes de *boosting* est souvent rapportée : l'erreur en généralisation décroît encore après que l'erreur en apprentissage soit stable ou même nulle. L'explication est que même si tous les exemples d'apprentissage sont déjà bien classés, la poursuite du *boosting* tend à maximiser les *marges* (Schapire *et al.*, 1997). À la suite de cette explication, certains ont cherché à maximiser explicitement la marge (Rätsch & Warmuth, 2002; Rätsch & Warmuth, 2003; Grove & Schuurmans, 1998) et ont dû constater que leurs systèmes étaient moins performants qu'ADABOOST. Sur cette thématique, citons encore (Harries, 1999) qui parvient à construire un classifieur donnant une marge de 1 à chacun des exemples d'apprentissage mais qui a des performances toujours moins bonnes qu'ADABOOST. Cette dernière expérience permet de rejeter définitivement l'explication des performances d'ADABOOST par les marges.

Nous pouvons également nous interroger sur la signification des poids α_t qu'ADABOOST associe aux hypothèses produites. À la vue de l'algorithme, il s'agit d'une valeur déterminée sur une distribution artificielle, elle-même fonction des échecs et réussites des hypothèses précédentes. Cependant, nous avons noté lors d'expériences sur des données très simples que l'erreur en généralisation diminuait encore alors que l'apprenant faible avait déjà fourni toutes les hypothèses possibles. Autrement dit, lorsqu'une hypothèse apparaît plusieurs fois, elle vote finalement avec un poids, cumul de tous ses α_t , qui a peut-être un caractère plus absolu. Si c'est bien le cas, on peut espérer approcher ces valeurs par un processus non adaptatif.

Enfin, une dernière question est celle de l'apprenant faible : comment bien le choisir ? À nouveau les éléments de réponse sont limités. Les auteurs d'ADABOOST indiquent que la définition simple et claire d'un apprenant faible est perdue avec le passage du cadre PAC aux problèmes réels et que la seule caractérisation d'un bon apprenant dont nous disposons est : *an algorithm that gives small errors when run under Adaboost* (Freund & Schapire, 1998). En particulier, faut-il autoriser ou même encourager l'apprenant faible à se tromper sur une partie des exemples d'apprentissage (les exemples de poids faibles) ? La réponse n'est pas évidente, surtout si l'on prend en compte qu'ADABOOST traite de manière privilégiée des données non bruitées (son

comportement naturel sur des données bruitées est de s'acharner sur les exemples difficiles à classer, donc sur le bruit).

Dans cet article, nous proposons justement d'utiliser un apprenant faible qui produit des hypothèses correctes : celles-ci peuvent donc s'abstenir sur une partie des exemples mais en aucun cas se tromper sur un exemple. Il s'agit de *moindres généralisés maximalement corrects*.

2 Booster des moindres généralisés avec ADABOOST

2.1 Moindres généralisés corrects

Comme son nom l'indique, le moindre généralisé de deux exemples est l'hypothèse la plus spécifique possible qui couvre ces deux exemples. En attribut-valeur, cette hypothèse est unique et peut être simplement définie comme suit :

- si les deux exemples partagent la même valeur d'un attribut catégoriel, celle-ci est conservée dans le moindre généralisé ; si ces valeurs sont distinctes, le moindre généralisé fait apparaître une valeur indéterminée pour cet attribut ;
- pour un attribut continu, on construit le plus petit intervalle incluant les deux valeurs présentes dans les exemples à généraliser.

Cette procédure apparaît clairement sur l'exemple suivant :

	âge	fumeur	sexe	classe
e ₁	25	non	homme	positif
e ₂	35	non	femme	positif
g ₁	[25, 35]	non	?	positif

Naturellement, cette opération peut se généraliser pour s'appliquer à deux hypothèses ou, cas qui nous intéresse le plus, à une hypothèse et un exemple :

	âge	fumeur	sexe	classe
g ₁	[25, 35]	non	?	positif
e ₃	30	oui	homme	positif
g ₂	[25, 35]	?	?	positif
e ₄	40	non	femme	positif
g ₃	[25, 40]	?	?	positif

Du point de vue de l'apprentissage supervisé, il est intéressant de généraliser ensemble des exemples d'une même classe, en prenant garde à ne pas couvrir d'exemples d'autres classes. Ce calcul, qui fournit la caractérisation d'un sous-concept au sein d'une classe, est celui d'un *moindre généralisé maximalement correct* et est décrit formellement par l'algorithme 2 : étant donné un exemple-graine, on essaye de généraliser, un à un, les exemples de la même classe avec comme critère d'acceptation la non couverture d'exemples d'autres classes (*correction*). La généralisation produite est *maximalement correcte* dans le sens où plus aucun exemple d'apprentissage ne peut lui être ajoutée sans perdre la correction. Notons également que cette hypothèse est fonction de l'ordre dans lequel les exemples sont testés pour l'ajout.

Algorithm 2 Correct_LGG_1

Entrées : s un exemple graine, $P = \{p_1, \dots, p_n\}$ un ensemble *ordonné* de n exemples de la même classe que la graine, N un ensemble de contre exemples. L'algorithme utilise l'opération élémentaire de MoindreGénéralisé entre une hypothèse et un exemple, ainsi qu'un test de subsomption noté \succeq permettant de décider si une hypothèse couvre ou non un exemple (ces deux opérations sont à défi nir en fonction du langage de description choisi).

Sortie : g une généralisation de P , maximale par rapport à N .

```

 $g = s$ 
for  $i = 1$  to  $n$  do
   $g' = \text{MoindreGénéralisé}(g, p_i)$  {Généralisation entre deux hypothèses.}
  if ( $\forall n \in N : \text{NOT}(g' \succeq n)$ ) then
     $g = g'$  {Si  $g'$  est correcte, elle devient la généralisation courante.}
  end if
end for
return  $g$ 

```

L'exemple suivant montre trois exécutions possibles de l'algorithme 2 (les exemples qui provoquent une perte de correction pour la généralisation courante sont repérés par un soulignement) :

graine	autres positifs	→	paquet maximale correct
p_1	$p_5 p_8 \underline{p_2} p_{14} \dots$	→	$\{p_1, p_5, p_8, p_{14}, \dots\}$
p_2	$\underline{p_{14}} p_3 \underline{p_1} p_{12} \dots$	→	$\{p_2, p_3, p_{12}, \dots\}$
p_3	$p_7 \underline{p_4} \underline{p_{18}} \underline{p_{12}} \dots$	→	$\{p_3, p_7, \dots\}$

On y voit l'importance de l'ordre de présentation des exemples et également que la contrainte de correction amène des généralisations qui prisent isolément ne couvrent pas tous les exemples positifs. Autrement dit, une telle généralisation ne peut conclure que sur une seule classe, la classe des exemples sur lesquels elle a été construite. Pour tous les autres exemples, ceux de la même classe qui ne sont pas couverts et ceux d'autres classes, la généralisation ne fournit pas de classe, elle s'abstient.

Cette brique de base a déjà été utilisée dans l'algorithme GLOBO (Torre, 1999). Dans ce cas, chaque exemple joue successivement le rôle de graine et pour chacun, la liste des exemples de même classe est mélangée aléatoirement. Puis GLOBO opère une couverture minimale pour ne garder que le nombre minimum d'hypothèses permettant de couvrir tous les exemples. Le but de la couverture minimale dans GLOBO est de produire finalement un ensemble réduit de règles qui soit appréhendable par un expert humain. Cette approche s'est révélée pertinente puisque GLOBO a remporté le PTE Challenge (Srinivasan *et al.*, 1997; Srinivasan *et al.*, 1999) dans la catégorie des systèmes produisant une théorie compréhensible.

Dans cet article, notre volonté est d'abandonner la compréhensibilité pour gagner en capacité prédictive et cela en utilisant le calcul de moindre généralisé correct comme apprenant faible dans des techniques de *boosting*.

2.2 Instanciation de l'algorithme ADABOOST

Nous proposons ici une nouvelle version du calcul de moindres généralisés corrects destinée à devenir un apprenant faible pour ADABOOST. Pour cela, il faut prendre en compte les poids des exemples, autrement dit favoriser la couverture des exemples de poids élevé. Pour cela, l'idée de notre apprenant faible est de réutiliser l'algorithme 2 en triant au préalable les exemples candidats à la généralisation par poids décroissants. Précisons immédiatement que, quels que soient les poids, nous ne relâchons pas notre critère de correction : la généralisation produite ne couvrira aucun contre-exemple, même de poids faible. Cette nouvelle version est présentée à l'algorithme 3.

Algorithm 3 Correct_LGG_2 (Weak Learner)

Entrées : n exemples (x_i, y_i, w_i) avec leurs étiquettes et poids.

Sortie : g une généralisation maximale correcte d'exemples de poids élevés.

$target$ = classe choisie au hasard

$seed$ = l'exemple de la classe $target$ ayant le poids le plus fort

$P = \{x_i | y_i = target, x_i \neq seed\}$

$N = \{x_i | y_i \neq target\}$

trier P par poids décroissants

return Correct_LGG_1($seed, P, N$) {Appel à l'algorithme 2}

Cet apprenant peut être fourni en l'état à ADABOOST (algorithme 1) dont l'utilisation ne nécessite pas d'autre information. Remarquons simplement que la formule de α_t (poids des hypothèses) utilisée dans ADABOOST se simplifie en prenant en compte que les moindres généralisés ainsi calculés ne font pas d'erreur de classification (ils classifient bien ou s'abstiennent). Ainsi, $W_- = 0$, par suite $W_0 = 1 - W_+$ et finalement

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 + W_+}{1 - W_+} \right)$$

La section suivante présente les résultats de ADABOOST ainsi instancié.

3 Expérimentations avec ADABOOST

Notre version d'ADABOOST est maintenant comparée aux algorithmes suivants : le classique C4.5 ; GLOBO, système à base de moindres généralisés mais sans *boosting* ; et ADABOOST (algorithme 1) avec en guise d'apprenant faible un constructeur de règles réduites à un unique test, un test pouvant prendre l'une des formes suivantes :

$att = valeur \rightarrow classe$

$att \geq valeur \rightarrow classe$

$att \leq valeur \rightarrow classe$

Ces règles peuvent être vues comme une simplification des *decision stumps*, arbres de décision limités à un seul niveau, utilisés comme hypothèses faibles par exemple dans (Freund & Schapire, 1996). Dans ce cas, le travail de l'apprenant faible consiste simplement à choisir parmi

TAB. 1 – Taux d’erreur de C4.5, GLOBO et ADABOOST sur huit problèmes de l’UCI. L’apprenant faible d’ADABOOST est soit le constructeur de tests élémentaires, soit celui de moindres généralisés corrects ; le nombre d’étapes de boosting vaut 100.

Problèmes	C4.5	GLOBO	Adaboost	
			test	généralisé
crx	14.78 %	16.83 %	14.35 %	17.33 %
glass	28.97 %	5.51 %	63.08 %	4.86 %
ionosphere	7.98 %	7.78 %	16.24 %	6.75 %
iris	5.33 %	7.40 %	34.00 %	6.00 %
pima	29.30 %	27.16 %	23.70 %	30.26 %
sonar	28.85 %	30.14 %	16.83 %	25.34 %
tic-tac-toe	14.41 %	1.20 %	25.57 %	0.21 %
wine	8.99 %	9.94 %	42.70 %	5.11 %
Moyennes	17.33 %	13.24 %	29.56 %	11.98 %

toutes les règles de cette forme, celle qui se comporte le mieux vis-à-vis de la distribution courante.

Ces systèmes sont évalués sur huit problèmes classiques de l’UCI (Blake & Merz, 1998), représentant un éventail assez large des problèmes que l’on peut rencontrer : attributs tous continus, tous discrets, ou mélange des deux types, avec deux classes à prédire ou plus, avec ou sans valeurs manquantes, hautement disjonctif ou pas.

Pour chacun de ces problèmes, on utilise une validation croisée 10 fois, le découpage étant le même pour toutes les expérimentations¹. Finalement, pour chaque problème et pour chaque système, on mesure le taux d’erreur moyen sur les 10 blocs de test. Précisons que les systèmes stochastiques (GLOBO et ADABOOST avec moindres généralisés) sont exécutés dix fois sur chaque bloc et c’est la moyenne sur ces dix exécutions qui est utilisée pour mesurer l’erreur du système.

Les résultats pour 100 étapes de *boosting* sont présentés à la table 1 où l’on a en plus fait apparaître pour chaque système la moyenne des taux d’erreur sur l’ensemble des problèmes. Il apparaît que de tous les systèmes en compétition, c’est ADABOOST avec moindres généralisés corrects qui obtient globalement les meilleurs résultats :

- le système C4.5 est battu pour cinq problèmes sur huit ;
- GLOBO est battu pour six de ces problèmes et l’on mesure ainsi ce que nous avons gagné en abandonnant la compréhension des théories produites par GLOBO (en moyenne, on passe de 13.24 % d’erreurs à 11.98 %) ;
- ADABOOST avec tests élémentaires est battu pour cinq problèmes et s’effondre complètement sur ceux-ci, au point d’altérer fortement la performance moyenne sur l’ensemble des problèmes (on observe 29.56 % pour les tests élémentaires contre 11.98 % pour les moindres généralisés).

¹Les fichiers de la validation croisée ainsi que des résultats complémentaires se trouvent à cette adresse : <http://www.grappa.univ-lille3.fr/~torre/Recherche/Experiments/>.

TAB. 2 – Taux d’erreur d’ADABOOST avec les deux apprenants faibles pour 100 et 1000 étapes de *boosting*.

Problèmes	test élémentaire		moindre généralisé	
	100	1000	100	1000
crx	14.35 %	13.77 %	17.33 %	14.06 %
glass	63.08 %	56.54 %	4.86 %	4.22 %
ionosphere	16.24 %	16.24 %	6.75 %	5.13 %
iris	34.00 %	34.00 %	6.00 %	5.33 %
pima	23.70 %	23.05 %	30.26 %	25.09 %
sonar	16.83 %	19.23 %	25.34 %	23.72 %
tic-tac-toe	25.57 %	9.50 %	0.21 %	0.00 %
wine	42.70 %	39.89 %	5.11 %	5.43 %
Moyennes	29.56 %	26.53 %	11.98 %	10.37 %

La table 2 présente les résultats d’expérimentations identiques où l’on a fait passer le nombre d’étapes de *boosting* de 100 à 1000. On note que l’erreur d’ADABOOST avec moindres généralisés diminue notablement sur quasiment tous les problèmes lorsque l’on passe de 100 étapes de *boosting* à 1000 (l’erreur globale passe de 11.98 % à 10.37 %). Ce comportement est aussi observé dans le cas des tests élémentaires mais dans une moindre mesure puisque l’erreur augmente de 2.4 points sur le problème *sonar*.

Nous voyons ainsi confirmée notre analyse des résultats de la table 1 : avec 1000 étapes de *boosting*, C4.5 et GLOBO sont maintenant battus sur tous les problèmes par ADABOOST avec moindres généralisés.

Ces résultats nous encouragent à augmenter encore le nombre d’étapes de *boosting* pour continuer à faire baisser l’erreur. Malheureusement, le calcul de moindres généralisés corrects est coûteux : pour chacun, il faut essayer d’ajouter les exemples de la classe cible un à un et chacun de ces ajouts doit être validé sur l’ensemble des contre-exemples.

Ainsi, le plus long des apprentissages avec 1000 étapes de *boosting* et l’utilisation de moindres généralisés prend une heure et vingt minutes sur une machine i686 à 2.5 Ghz pour une base qui contient moins de 1000 exemples. Cette remarque est sans doute vraie également pour bon nombre d’apprenants faibles. Ainsi, notre apprenant produisant des tests simples nécessite de calculer à l’avance toutes les règles possibles ce qui prend également plus d’une heure pour le problème évoqué (cela dit, une fois cette tâche effectuée, les étapes de *boosting* peuvent être multipliées sans surcoût important).

Dans notre cas, il est donc impératif, pour que la méthode soit viable, de rendre l’apprentissage plus rapide. Nous avons choisi pour cela de paralléliser la génération des hypothèses et donc de différer l’affectation de poids à ces hypothèses. Cette étude est l’objet de la section suivante.

4 Booster des moindres généralisés sans ADABOOST

Nous l'avons vu, la production d'un moindre généralisé correct est de complexité quadratique dans le nombre d'exemples disponibles et cela n'est pas acceptable pour un apprenant faible destiné à être appelé de nombreuses fois dans un algorithme de type ADABOOST.

Nous aurions pu, pour réduire le temps de calcul, échantillonner et distribuer les données à différentes unités de calcul. Nous avons préféré distribuer, non pas les données, mais le calcul des hypothèses, lequel se fera toujours sur l'ensemble complet des données disponibles. Cela nécessite de savoir :

- produire les hypothèses indépendamment les unes des autres ;
- affecter un poids à chacune de ces hypothèses *a posteriori*.

Nous commençons par le deuxième point en attribuant de nouveaux poids aux hypothèses disponibles, quel que soit leur mode de production.

4.1 Affectation de poids aux hypothèses produites

Le premier de ces poids, dans la suite repéré par le nom *adalike*, cherche à imiter le mode de calcul des poids d'ADABOOST. Comme ADABOOST, nous commençons par définir le poids d'un exemple x_i , cela à partir de T_i le nombre de fois où cet exemple est couvert par les hypothèses produites :

$$w_i = \exp(-T_i)$$

Cette formule fait référence à ADABOOST (algorithme 1) qui multiplie le poids d'un exemple par $\exp(-\alpha_t)$ pour chaque hypothèse h_t couvrant l'exemple. Une fois ces poids attribués aux exemples, on les normalise en divisant chacun par $Z = \sum_i w_i$. Puis nous définissons le poids d'une hypothèse h . W_+ est défini comme la somme des poids des exemples couverts par h et n est le nombre de fois où cette hypothèse est apparue pendant la production. Finalement, le poids *adalike* de h est calculé comme suit :

$$\alpha(h) = \frac{n}{2} \log \left(\frac{1 + W_+}{1 - W_+} \right)$$

La valeur de n intervient pour simuler le cumul des poids effectué par ADABOOST lorsqu'une même hypothèse apparaît plusieurs fois.

Nous considérons également les poids suivants, plus simples :

- *covering* : chaque hypothèse distincte vote avec un poids qui correspond au nombre d'exemples qu'elle couvre dans l'ensemble d'apprentissage (dans notre cas, les hypothèses sont correctes par construction et les exemples couverts par une hypothèse sont donc tous de la même classe) ;
- *frequency* : chaque hypothèse distincte vote avec un poids qui correspond à son nombre d'apparitions pendant la phase de production ;
- *uniform* : toutes les hypothèses distinctes votent avec un poids égal.

4.2 Production d'hypothèses sans ADABOOST

L'étape suivante est de trouver une alternative à ADABOOST pour produire des hypothèses. Nous proposons pour cela GLOBOOST (algorithme 4) : celui-ci produit des moindres généralisés corrects de façon purement stochastique et, par conséquent, sans aucun lien de dépendance entre eux. Cela nous permettra à terme de partager ce travail de génération entre plusieurs machines.

Algorithm 4 GLOBOOST

Entrées : n exemples (x_i, y_i) avec étiquettes et poids ; T un nombre d'itérations.

Sortie : H le classifieur final.

```

for  $t = 1$  to  $T$  do
     $target$  = classe choisie au hasard
     $seed$  = un exemple de la classe  $target$  choisi au hasard
     $P = \{x_i | y_i = target, x_i \neq seed\}$ 
     $N = \{x_i | y_i \neq target\}$ 
    mélanger  $P$  aléatoirement
     $h_t = \text{Correct\_LGG\_1}(seed, P, N)$  {Appel à l'algorithme 2, page 6}
end for
for all  $t$  tel que  $1 \leq t \leq T$  do
     $\alpha_t = \text{poids}(h_t)$  {fonction à instancier}
end for
return  $H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t \cdot h_t(x) \right)$ 

```

5 Évaluation expérimentale de GLOBOOST

Dans cette section, nous évaluons expérimentalement les idées de la section précédente : génération des hypothèses par GLOBOOST en remplacement d'ADABOOST et affectation de poids *a posteriori*. Le protocole et les données sont les mêmes qu'à la section 3.

5.1 Évaluation expérimentale des poids proposés

Nous conservons tout d'abord ADABOOST comme générateur d'hypothèses. Après la phase de *boosting*, nous obtenons un ensemble d'hypothèses avec leurs poids attribués par ADABOOST et nous leur affectons les poids supplémentaires définis précédemment. On obtient ainsi des systèmes de vote distincts même si tous sont fondés sur les mêmes votants.

Notre intuition, forgée par les différents travaux qui cherchent à redéfinir les poids pour maximiser la marge (Schapire *et al.*, 1997; Rätsch & Warmuth, 2002; Rätsch & Warmuth, 2003; Grove & Schuurmans, 1998; Harries, 1999), est que ADABOOST a un comportement quasi optimal dans le réglage des poids des hypothèses. Il s'agit donc ici de quantifier ce que nous perdons en abandonnant les poids des hypothèses fixés par ADABOOST.

Les résultats pour 100 et 1000 étapes de *boosting* sont présentés à la table 3. La première

TAB. 3 – Taux d'erreur selon les poids attribués aux moindres généralisés corrects produits par ADABOOST.

Problèmes	adaboost	adalike	covering	frequency	uniform
<i>100 étapes de boosting</i>					
crx	17.33 %	16.62 %	17.26 %	20.19 %	20.17 %
glass	4.86 %	4.86 %	4.95 %	4.86 %	4.95 %
ionosphere	6.75 %	15.53 %	6.75 %	7.26 %	7.29 %
iris	6.00 %	5.07 %	6.07 %	6.07 %	6.07 %
pima	30.26 %	30.21 %	30.42 %	33.24 %	33.24 %
sonar	25.34 %	41.97 %	25.00 %	23.75 %	23.75 %
tic-tac-toe	0.21 %	0.21 %	0.21 %	0.69 %	1.11 %
wine	5.11 %	15.28 %	5.06 %	5.11 %	5.06 %
Moyennes	11.98 %	16.22 %	11.96 %	12.65 %	12.71 %
<i>1000 étapes de boosting</i>					
crx	14.06 %	26.38 %	14.35 %	21.00 %	20.72 %
glass	4.21 %	8.41 %	6.07 %	4.21 %	6.07 %
ionosphere	5.13 %	52.99 %	9.31 %	7.50 %	7.60 %
iris	5.33 %	6.67 %	6.00 %	6.00 %	6.89 %
pima	25.09 %	24.83 %	25.87 %	29.77 %	29.77 %
sonar	23.72 %	50.80 %	22.76 %	21.79 %	21.79 %
tic-tac-toe	0.00 %	0.00 %	0.00 %	0.52 %	2.71 %
wine	5.43 %	45.13 %	4.49 %	5.06 %	4.49 %
Moyennes	10.37 %	26.90 %	11.11 %	11.98 %	12.50 %

TAB. 4 – Résultats de GLOBOOST et des poids *adalike*, *covering*, *frequency*, *uniform*.

Problèmes	adalike	covering	frequency	uniform
<i>100 étapes</i>				
crx	15.28 %	14.45 %	14.61 %	14.59 %
glass	4.72 %	4.72 %	4.72 %	4.72 %
ionosphere	8.06 %	8.03 %	8.15 %	8.21 %
iris	6.40 %	6.40 %	6.40 %	6.33 %
pima	27.17 %	26.91 %	27.15 %	27.15 %
sonar	34.33 %	25.29 %	24.57 %	24.57 %
tic-tac-toe	0.59 %	0.59 %	0.71 %	2.18 %
wine	14.04 %	4.04 %	4.10 %	4.10 %
Moyennes	13.82 %	11.30 %	11.30 %	11.48 %
<i>1000 étapes</i>				
crx	25.48 %	14.06 %	13.97 %	14.09 %
glass	8.79 %	6.57 %	4.67 %	6.54 %
ionosphere	16.01 %	7.69 %	7.29 %	7.46 %
iris	12.00 %	5.87 %	6.00 %	5.87 %
pima	29.04 %	24.32 %	24.35 %	24.35 %
sonar	48.27 %	23.56 %	23.46 %	23.46 %
tic-tac-toe	0.23 %	0.52 %	0.17 %	7.33 %
wine	46.85 %	3.93 %	4.49 %	3.82 %
Moyennes	23.33 %	10.82 %	10.55 %	11.62 %

constatation est que ADABOOST reste le meilleur pour fixer les poids des hypothèses qu'il produit, en particulier lorsque le nombre d'étapes de *boosting* augmente.

Notre imitation *adalike*, pour un nombre d'étapes de *boosting* de 100, obtient les taux d'erreur les plus faibles sur cinq des huit problèmes considérés mais se comporte très mal sur trois autres problèmes. Malheureusement, cela se répercute fortement sur la moyenne globale et empire avec le passage de 100 à 1000 étapes de *boosting*.

Les poids *covering*, *frequency* et, dans une moindre mesure *uniform*, présentent des résultats plus intéressants : *covering* semble être équivalent au poids d'ADABOOST pour 100 étapes de *boosting*. Il est important de noter que, même si ces poids sont finalement moins bons que les poids *adaboost*, ils maintiennent de meilleures performances que celles des systèmes C4.5 et GLOBO évalués à la section 3, table 1.

En conclusion de ces expérimentations, nous disposons de poids qui peuvent raisonnablement se substituer aux poids traditionnels d'ADABOOST.

5.2 Évaluation expérimentale de GLOBOOST

Toujours avec le protocole et les données de la section 3, nous évaluons l'algorithme GLOBOOST muni des différents poids dont nous disposons (*adalike*, *covering*, *frequency* et *uniform*). Les résultats sont présentés à la table 4.

À nouveau, notre poids *adalike* fait apparaître des valeurs aberrantes et supporte mal l'augmen-

TAB. 5 – Comparaison de GLOBOOST, C4.5, GLOBO et ADABOOST avec moindres généralisés. GLOBOOST et ADABOOST utilisent ici 1000 étapes de génération.

Problèmes	C4.5	GLOBO	ADABOOST	GLOBOOST
crx	14.78 %	16.83 %	14.06 %	13.97 %
glass	28.97 %	5.51 %	4.21 %	4.67 %
ionosphere	7.98 %	7.78 %	5.13 %	7.29 %
iris	5.33 %	7.40 %	5.33 %	6.00 %
pima	29.30 %	27.16 %	25.09 %	24.35 %
sonar	28.85 %	30.14 %	23.72 %	23.46 %
tic-tac-toe	14.41 %	1.20 %	0.00 %	0.17 %
wine	8.99 %	9.94 %	5.43 %	4.49 %
Moyennes	17.33 %	13.24 %	10.37 %	10.55 %

tation du nombre d'étapes de *boosting*. Par contre, les poids *covering* et *frequency* confirment leurs bons résultats au point de concurrencer ADABOOST. De plus, cette tendance se confirme lorsque l'on augmente le nombre d'étapes de 100 à 1000. Ce résultat est surprenant pour nous : ces poids faisaient baisser les performances d'ADABOOST lorsque celui-ci était utilisé comme générateur et on s'attendait à ce que les taux d'erreur augmentent un peu plus avec un générateur purement aléatoire. Au contraire, ces poids se révèlent mieux adaptés à ce mode de production des hypothèses.

Enfin, notons que le poids *uniform*, même s'il est moins bon que *covering* et *frequency* et même s'il ne s'améliore pas avec le nombre d'étapes, obtient tout de même des résultats honorables en regard de sa simplicité.

Précisons que, dans le cas de la génération purement aléatoire de GLOBOOST, la fréquence et la couverture sont fortement liées : une hypothèse ayant un fort support sortira souvent, et inversement, une hypothèse fréquente couvre nécessairement beaucoup d'exemples. Ceci explique les résultats proches des poids basés sur ces quantités.

6 Bilan et perspectives

Nous avons montré que les résultats sur le *boosting* laissent une large place à l'investigation, en particulier sur les apprenants faibles qui peuvent être utilisés. Nous avons proposé d'utiliser dans ce rôle la production de *moindres généralisés corrects*. Devant la complexité de ce calcul et l'usage intensif que doit en faire ADABOOST, nous avons proposé une génération des hypothèses parallélisable et avons démontré que des poids simples pouvaient ensuite être affectés aux hypothèses. Toutes ces idées ont été validées expérimentalement.

Aujourd'hui, le système GLOBOOST implémente la production aléatoire vue à l'algorithme 4 avec le poids *frequency*. Ce poids offre de bonnes performances et n'occasionne pas de calculs importants après la génération. La table 5 dresse le bilan de nos expérimentations pour cet algorithme. En résumé GLOBOOST présente une erreur faible, comparable à celle d'ADABOOST, avec l'avantage que le calcul peut être réparti sur plusieurs machines.

La question se pose maintenant de savoir si ce résultat est propre aux moindres généralisés corrects. La seule particularité de notre apprenant faible est de fournir des hypothèses qui ne se trompent pas ; on a vu que cela simplifiait le calcul des poids dans ADABOOST. Peut-être que cela rend les poids et les performances d'ADABOOST plus faciles à approcher. Il faudra également déterminer si autoriser un moindre généralisé à être incorrect permet à ADABOOST d'atteindre de meilleures performances. Précisons que cela va dans le sens de notre souci d'efficacité : un apprenant faible incorrect n'a plus besoin de considérer tous les exemples mais seulement ceux de poids forts. Enfin, il faudra déterminer si, dans ce cadre, nous sommes encore capables de proposer des poids pertinents.

Nous avons vu que ADABOOST et GLOBOOST, boostant des moindres généralisés corrects, s'amélioreraient avec l'augmentation du nombre d'hypothèses produites. Il faudra donc poursuivre dans cette voie, cela étant plus facile en pratique pour la version parallèle de GLOBOOST.

La recherche de nouveaux poids reste elle aussi ouverte. En particulier, il faut déterminer pourquoi notre poids *adali* se comporte mal sur certains problèmes.

Une perspective qui nous semble importante est celle de la compréhensibilité du classifieur produit. Il s'agit maintenant de savoir si, à partir des hypothèses produites par ADABOOST ou GLOBOOST, on peut revenir à des théories compréhensibles comme celles de GLOBO. La solution la plus simple est d'appliquer la couverture minimale de GLOBO sur les hypothèses produites mais il est peu probable que les performances de la théorie obtenue restent proches de celles du système de votants. Il serait donc préférable de chercher un ensemble d'hypothèses d'assez petite taille pour être appréhendable par un humain mais en conservant au maximum le pouvoir prédictif du classifieur complet. Précisons que, tout proche de cette problématique, le problème de fournir un nombre minimal de votants a été suggéré dans (Quinlan, 1999) et une solution proposée dans (Long, 2002), solution qui implique de calculer d'abord toutes les hypothèses faibles possibles (dans ce cas, des *decision stumps*).

Enfin, un dernier objectif est de poursuivre la parallélisation en distribuant également les données : la génération des hypothèses serait distribuée comme nous l'avons décrit, mais en plus elle se ferait sur des échantillons de l'ensemble de données initial. Cela conduira à des hypothèses incorrectes mais nous avons vu que cela pouvait être un avantage. À noter qu'une fois cette distribution mise en œuvre, le poids *frequency* sera un candidat privilégié puisqu'il ne nécessite pas de calcul sur l'ensemble de données, comme ce serait le cas par exemple pour le poids *covering*.

Références

- BLAKE C. & MERZ C. (1998). UCI repository of machine learning databases [<http://www.ics.uci.edu/~mllearn/MLRepository.html>].
- BREIMAN L. (1996a). Bagging predictors. *Machine Learning*, **24**(2), 123–140.
- BREIMAN L. (1996b). Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California.
- FREUND Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, **121**(2), 256–285.
- FREUND Y. & SCHAPIRE R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, p. 23–37.

- FREUND Y. & SCHAPIRE R. E. (1996). Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, p. 148–156.
- FREUND Y. & SCHAPIRE R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**(1), 119–139.
- FREUND Y. & SCHAPIRE R. E. (1998). Discussion of the paper Arcing Classifiers by Leo Breiman. *The Annals of Statistics*, **26**, 824–832.
- GROVE A. J. & SCHUURMANS D. (1998). Boosting in the limit : Maximizing the margin of learned ensembles. In *AAAI/IAAI*, p. 692–699.
- HARRIES M. (1999). Boosting a strong learner : evidence against the minimum margin. In *Proc. 16th International Conf. on Machine Learning*, p. 171–180 : Morgan Kaufmann, San Francisco, CA.
- KEARNS M. & VALIANT L. G. (1989). Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, p. 433–444.
- KEARNS M. J. & VAZIRANI U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press.
- LONG P. M. (2002). Minimum majority classification and boosting. In *Proceedings of the The Eighteenth National Conference on Artificial Intelligence*.
- QUINLAN J. R. (1999). Some elements of machine learning. Invited talk (ILP / ICML).
- RÄTSCH G. & WARMUTH M. (2002). Maximizing the margin with boosting. In J. KIVINEN & R. H. SLOAN, Eds., *Proceedings of the Annual Conference on Computational Learning Theory (COLT)*, volume 2375 of *Lecture Notes in Computer Science*, p. 334–350 : Springer.
- RÄTSCH G. & WARMUTH M. K. (2003). Efficient margin maximizing with boosting. submitted to *Journal of Machine Learning Research (JMLR)*.
- SCHAPIRE R. E. (1990). The strength of weak learnability. *Machine Learning*, **5**, 197–227.
- SCHAPIRE R. E., FREUND Y., BARTLETT P. & LEE W. S. (1997). Boosting the margin : a new explanation for the effectiveness of voting methods. In *Proc. 14th International Conference on Machine Learning (ICML)*, p. 322–330 : Morgan Kaufmann.
- SCHAPIRE R. E. & SINGER Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, **37**(3), 297–336.
- SCHAPIRE R. E. & SINGER Y. (2000). Boostexter : A boosting-based system for text categorization. *Machine Learning*, **39**(2/3), 135–168.
- SRINIVASAN A., KING R. & BRISTOL D. (1999). An assessment of submissions made to the predictive toxicology evaluation challenge. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, p. 270–276 : Morgan Kaufmann.
- SRINIVASAN A., KING R. D., MUGGLETON S. H. & STERNBERG M. J. E. (1997). The predictive toxicology evaluation challenge. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, p. 4–9 : Morgan Kaufmann.
- TORRE F. (1999). GloBo : un algorithme stochastique pour l'apprentissage supervisé et non-supervisé. In M. SEBAG, Ed., *Actes de la Première Conférence d'Apprentissage*, p. 161–168.
- VALIANT L. G. (1984). A theory of the learnable. *Communications of the ACM*, p. 1134–1142.