Adaptation du boosting à l'inférence grammaticale *via* l'utilisation d'un oracle de confiance

Jean-Christophe Janodet* — **Richard Nock****
Marc Sebban* — **Henri-Maxime Suchier***

* *Équipe Universitaire de Recherche en Informatique de Saint-Étienne*
23 rue du Dr Paul Michelon, 42023 Saint-Étienne cédex 2
{janodet,Marc.Sebban,Henri.Maxime.Suchier}@univ-st-etienne.fr

** *Département Scientifique Inter-facultaire, Université Antilles-Guyane*
Campus de Schoelcher, B.P. 7209, Schoelcher 97275, Martinique
rnock@martinique.univ-ag.fr

RÉSUMÉ. *Cet article présente une adaptation du boosting à l'inférence grammaticale. Notre but est d'améliorer les performances d'un algorithme à base de fusion d'états, en présence de données bruitées. Notre algorithme de boosting utilise une nouvelle règle de mise à jour des poids qui tient compte d'une information supplémentaire fournie par un oracle. Cette information est une évaluation de la confiance en l'étiquette d'un exemple. Nous montrons que la règle de mise à jour des poids conserve les propriétés théoriques du boosting. Quant à l'oracle, il doit être simulé en pratique et nous proposons une construction adaptée à l'inférence grammaticale. Nous décrivons enfin une étude expérimentale sur l'algorithme à base de fusions d'états* RPNI**, dont les performances sont significativement améliorées.*

ABSTRACT. *In this paper we focus on the adaptation of boosting to grammatical inference. We aim at improving the performances of state merging algorithms in the presence of noisy data by using, in the update rule, additional information provided by an oracle. This strategy requires the construction of a new weighting scheme that takes into account the confidence in the labels of the examples. We prove that our new framework preserves the theoretical properties of boosting. Using the state merging algorithm* RPNI**, we describe an experimental study on various datasets, showing a dramatic improvement of performances.*

MOTS-CLÉS : *boosting, inférence grammaticale.*

KEYWORDS: *boosting, grammatical inference.*

RSTI - RIA – 19/2005. Apprentissage automatique, pages 713 à 740

1. Introduction

L'inférence grammaticale est un sous-domaine de l'apprentissage automatique dont le but est d'apprendre des modèles de langages (ensembles de mots). Parmi ceux-ci, les automates finis sont des machines à états qui prennent des mots en entrée et qui les acceptent ou les rejettent. Du point de vue de l'apprentissage automatique, les automates finis sont des classifieurs qui séparent l'ensemble de tous les mots en deux classes, la classe dite positive des mots acceptés par l'automate, et celle dite négative des mots rejetés par celui-ci. Une des raisons expliquant les efforts consentis pour apprendre de tels classifieurs tient au fait que de nombreux domaines d'application, comme la reconnaissance de la parole, la reconnaissance de formes ou le traitement des langues naturelles, tirent partie des propriétés structurelles et sémantiques des automates finis (de la Higuera, 2004). Nous nous intéressons dans cet article aux algorithmes produisant des automates finis par un mécanisme de fusion d'états. On peut répartir de tels algorithmes en deux familles principales. D'une part, les algorithmes d'inférence exacte, qui utilisent à la fois des exemples positifs et négatifs du concept à apprendre pour produire un automate fini déterministe (AFD). Citons par exemple RPNI (Oncina *et al.*, 1992), mais aussi EDSM (Lang *et al.*, 1998) ou ESMA (Coste, 1999) qui en sont dérivés. D'autre part, les algorithmes d'inférence probabiliste, qui utilisent des exemples positifs uniquement, pour produire des automates finis probabilistes (comme MDI (Thollard *et al.*, 2000), ou encore ALERGIA (Carrasco *et al.*, 1994)).

Cet article se concentre uniquement sur les algorithmes d'inférence grammaticale exacte. Nous utilisons en particulier RPNI, même si nos résultats peuvent s'étendre facilement à d'autres algorithmes, variantes ou extensions de RPNI. RPNI fonctionne à partir de deux ensembles de données, E_+ et E_-, qui contiennent respectivement des mots acceptés et rejetés par l'AFD que l'on doit apprendre. Par exemple, l'AFD de la figure 1 a été obtenu avec RPNI pour $E_+ = \{b, ab, abb, bab, \lambda\}$ et $E_- = \{aa, ba\}$, où λ désigne le mot vide.

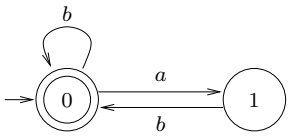

Figure 1. *Un* AFD *appris avec* RPNI

RPNI est considéré comme un algorithme d'apprentissage exact au sens où il retourne un AFD qui « colle » aux données : un exemple positif (de E_+) sera nécessairement reconnu par cet AFD, et un exemple négatif (de E_-) sera nécessairement rejeté. De plus, si les données d'apprentissage contiennent certains mots, dits *caractéristiques* de l'AFD qui a produit les données (AFD *cible*), alors on peut montrer que RPNI trouve cette cible (Oncina *et al.*, 1992). Ces propriétés ont un intérêt théorique indénia-

ble. Mais dès qu'on tente d'utiliser RPNI (et plus généralement les algorithmes d'inférence exacte) sur un problème réel, où les données sont intrinsèquement bruitées, ces propriétés constituent un véritable handicap. RPNI n'étant pas immunisé contre le surapprentissage (au contraire), les données bruitées ont un effet désastreux ; cela se traduit par une augmentation du nombre d'états des AFD produits, et par une chute du taux de succès en généralisation de ces AFD. C'est pourquoi la tolérance au bruit est devenue un problème crucial à résoudre, aussi bien en inférence grammaticale, que dans l'ensemble des domaines de l'apprentissage automatique d'ailleurs.

Une première approche pour limiter le risque de surapprentissage des algorithmes à base de fusions d'états a été présentée dans (Sebban *et al.*, 2003). Les auteurs y proposent une relaxation de la règle de fusion de RPNI. Ces travaux ont abouti à un nouvel algorithme, RPNI*, dont la règle de fusion, reposant sur la statistique inférentielle, tolère la présence de données bruitées. D'un point de vue expérimental, les améliorations apportées par rapport à RPNI sont très significatives. Mais cette approche reste limitée pour deux raisons. D'abord, les auteurs fournissent des résultats théoriques montrant la difficulté d'inférer des AFD en présence de forts taux de bruit (au-delà de 10 %). Ensuite, de par sa nature statistique, l'utilisation de RPNI* peut entrainer l'acceptation à tort d'une fusion. En effet, si RPNI n'accepte pas d'exemple négatif dans un état final, RPNI*, en relâchant la contrainte de fusion, permet à des exemples positifs et négatifs de cohabiter dans un même état. Si l'on espère que ces exemples négatifs sont des données bruitées (et donc faussement négatives), on peut craindre la présence d'exemples réellement négatifs dans un état final, ce qui peut de nouveau conduire à des automates de grande taille, avec une faible capacité de généralisation.

En fait, ces deux problèmes de RPNI* sont à rapprocher de deux questions récurrentes et plus générales en apprentissage automatique :

1) Est-il possible d'améliorer les performances en généralisation d'un algorithme d'apprentissage donné (dans notre cas le taux de succès d'un algorithme à base de fusions d'états) ?

2) Est-il possible de préserver ces performances en présence de données bruitées (dans notre cas, éviter les fusions d'états qui feraient cohabiter des « vrais » exemples positifs et négatifs dans le même état) ?

Indiscutablement, nous pensons que ces deux questions peuvent être traitées à l'aide du boosting, même si cela n'a jamais été fait en inférence grammaticale, à notre connaissance. Le principe du boosting et de son algorithme ADABOOST (Freund *et al.*, 1996; Freund *et al.*, 1997) consiste à apprendre T hypothèses (dites *faibles*) à l'aide d'un algorithme WL faible sur T distributions de probabilités $\mathbf{w}_t$ d'un échantillon d'apprentissage E, puis à combiner ces hypothèses faibles h_t en une seule hypothèse H_T « forte ». A chaque itération, la mise à jour de la distribution $\mathbf{w}_t$, point crucial d'ADABOOST, favorise (exponentiellement) les exemples mal appris par l'algorithme à l'étape précédente. Pour le lecteur non spécialiste du boosting, rentrons brièvement dans le détail d'ADABOOST (dont le pseudo-code est présenté dans l'algorithme 1). On dispose d'un ensemble d'apprentissage E constitué de couples $(x, y(x))$

où $y(x) \in \{-1, +1\}$ est l'étiquette de x, lui-même étant un vecteur de représentation à p dimensions. On définit une première distribution $\mathbf{w}_0$ uniforme sur l'échantillon d'apprentissage E. Puis, à chaque itération t, on utilise WL sur E et $\mathbf{w}_t$ pour obtenir une hypothèse faible h_t. On repondère ensuite les exemples d'apprentissage en construisant une distribution $\mathbf{w}_{t+1}$ à partir de $\mathbf{w}_t$: on augmente exponentiellement le poids des exemples mal classés par h_t (c'est-à-dire ceux pour lesquels $y(x)h_t(x) = -1$), et on diminue exponentiellement le poids des exemples bien classés par h_t (c'est-à-dire ceux pour lesquels $y(x)h_t(x) = +1$) ; cette repondération est fonction d'un coefficient c_t dépendant de l'erreur en apprentissage ϵ_t de h_t. Ce processus est répété T fois. L'hypothèse finale H_T est la combinaison des T hypothèses faibles ainsi construites, pondérées par les coefficients c_t.

Data : Un échantillon d'apprentissage E, un apprenant faible WL, le nombre T d'hypothèses à construire.
Result : Une combinaison H_T de classifieurs.
for all $x \in E$ **do** $w_0(x) \longleftarrow 1/|E|$;
for $t = 0$ **to** T **do**
  $h_t \longleftarrow \text{WL}(E, \mathbf{w}_t)$;
  $c_t \longleftarrow (1/2)\ln((1-\epsilon_t)/\epsilon_t)$ où $\epsilon_t = \sum_{\{e \in E : y(x) \neq h_t(x)\}} w_t(x)$;
  for all $x \in E$ **do**
    $w_{t+1}(x) \longleftarrow w_t(x)\exp(-c_t y(x) h_t(x))/Z_t$
    où $Z_t = \sum_{e \in E} w_t(x)\exp(-c_t y(x) h_t(x))$;
return H_T **such that** $H_T(x) = \text{signe}\left(\sum_{t=0}^{T} c_t h_t(x)\right)$;

Algorithme 1: *Pseudo-code d'*ADABOOST

Récemment, des travaux ont tenté de limiter les risques de surapprentissage du boosting en présence de données bruitées en évaluant l'intérêt d'augmenter le poids d'un exemple mal classé : ADABOOST ayant tendance à augmenter à tort les poids des exemples bruités, certaines approches tentent de contrôler la mise à jour en utilisant des fonctions de repondération plus douces que la fonction exponentielle originale (Friedman *et al.*, 1998; Domingo *et al.*, 2000; Freund, 2001). Mais l'utilisation de ces nouvelles fonctions peut remettre en question les propriétés théoriques de convergence du boosting. On pourrait aussi penser que des exemples bruités, dont la classe est douteuse, devraient être supprimés de l'échantillon d'apprentissage. Mais une tendance actuelle, initiée par (Blum *et al.*, 1998), montre que les exemples sans étiquette (ou d'étiquette erronée) peuvent aussi apporter une aide significative, pourvu que l'on sache utiliser l'information qu'ils contiennent. En inférence grammaticale, si la classe des mots permet de donner une étiquette aux états, les mots eux-mêmes permettent de déterminer la structure de l'AFD (au sens des états et des transitions). Il semble donc essentiel d'intégrer ces mots, à l'étiquette douteuse, dans les processus d'apprentissage et de boosting.

Dans cet article, nous avons choisi de conserver les exemples bruités et l'emploi de la fonction exponentielle du boosting, mais nous supposons avoir accès à un oracle

de confiance. Cet oracle, que nous serons amenés parfois à simuler en cas d'absence d'expertise suffisante du domaine traité, fournit une estimation sur la confiance dans la classe d'un exemple. Une valeur positive pour un exemple signifie que nous pouvons être certains de sa classe. En revanche, une valeur négative signifie que l'on doit émettre des doutes sur son appartenance. Il est crucial de remarquer qu'une confiance négative ne signifie donc pas que l'exemple appartient à la classe opposée, mais plutôt que l'on ne connaît pas sa classe réelle. De plus, il est nécessaire de supposer que l'oracle fournit des valeurs *réelles* dans $[-1, +1]$ plutôt que des valeurs *entières* dans $\{-1, +1\}$. Ce choix est lié au fait que cet oracle devant parfois être simulé en pratique, il est préférable de tenir compte des imperfections de cette simulation dès l'étude théorique.

L'utilisation de l'oracle de confiance pour traiter les données bruitées nécessite incontournablement une modification de la règle standard de mise à jour des poids. En effet, si ADABOOST est normalement utilisé pour augmenter le poids de tous les exemples mal appris, nous voulons ici seulement augmenter les poids des exemples qui le « méritent », c'est-à-dire ceux non bruités. Si cette stratégie semble utile pour améliorer n'importe quel algorithme d'apprentissage, nous pensons qu'elle est particulièrement adaptée pour repousser les limites de RPNI*. D'où l'utilisation spécifique, dans cet article, de notre nouveau cadre de boosting à l'inférence grammaticale.

Cet article est organisé comme suit. Après avoir détaillé le fonctionnement de RPNI* (section 2), nous proposons en section 3 un nouveau cadre théorique de boosting adapté à l'utilisation d'un oracle. Nous modifions la règle standard de mise à jour d'ADABOOST, et nous prouvons que cette nouvelle stratégie de repondération conserve les propriétés théoriques du boosting. Puis nous proposons dans la section 4 une stratégie pour simuler de manière pertinente un oracle satisfaisant les propriétés voulues. Notre méthode est basée sur l'utilisation d'un graphe des k plus proches voisins construit à partir de l'ensemble d'apprentissage. Dans la section 5, nous montrons sur différentes bases de données, que notre algorithme améliore significativement les performances de RPNI*. Nous étudions la part de cette amélioration due à l'oracle, et celle directement induite par notre schéma de boosting.

2. L'algorithme à base de fusions d'états RPNI*

Le but de cette section, après quelques rappels de notions élémentaires en théorie des langages, est de décrire RPNI* tel qu'il a été défini dans (Sebban *et al.*, 2003). Comme son fonctionnement s'appuie sur celui de RPNI (Oncina *et al.*, 1992), nous rappelons le principe de ce dernier sur un exemple d'exécution. Nous analyserons ensuite l'effet du bruit, ce qui permettra de présenter le mode de fonctionnement de RPNI*.

Un *alphabet* Σ est un ensemble fini non vide de *lettres* et on appelle *mot* sur Σ toute séquence finie $w = a_1 a_2 \dots a_n$ de lettres. On note λ le mot vide (sans lettre). Un *automate fini déterministe* (AFD) est un quadruplet $A = \langle Q, \delta, s_0, F \rangle$ tel que Q

soit un ensemble fini d'états, $\delta : Q \times \Sigma \rightarrow Q$ soit une fonction de transition, $s_0 \in Q$ soit un état initial et $F \subseteq Q$ soit un ensemble d'états finaux. δ est étendue à l'ensemble de tous les mots en posant $\delta(q, \lambda) = q$ et $\delta(q, xw) = \delta(\delta(q, x), w)$. On dit qu'un mot w est *reconnu* par A si $\delta(s_0, w) \in F$ et qu'il est *rejeté* sinon. Par abus de langage, on dira qu'un état s *contient* un mot w ssi $\delta(s_0, w) = s$. Un mot sera donc accepté par A s'il est contenu par un état final. Ainsi, l'AFD de la figure 1 est défini par $Q = \{0, 1\}$, $s_0 = 0$, $F = \{0\}$, $\delta(0, a) = 1$ et $\delta(0, b) = \delta(1, b) = 0$. abb est reconnu car $\delta(0, abb) = \delta(1, bb) = \delta(0, b) = 0 \in F$. Par contre, ba et aa sont rejetés car $\delta(0, ba) = 1 \notin F$ et $\delta(0, aa)$ n'est pas défini.

```
Data   : Un échantillon d'apprentissage ⟨E+, E−⟩
Result : Un AFD A
A ⟵ PTA(E+) ; N ⟵ Nb d'états de A ; π ⟵ {{q} : q ∈ Q};
for i = 1 to N − 1 do
    if i = min(π(i)) then
        j ⟵ 0 ; continue ⟵ vrai;
        while (j < i) and continue do
            if j = min(π(j)) then
                π′ ⟵ calcule_fusion(π, i, j, δ);
                if compatible(A/π′, E−) then
                    π ⟵ π′ ; continuer ⟵ faux
            j ⟵ j + 1
return(A/π)
```

Algorithme 2: *Pseudo-code de* RPNI

Le principe de fonctionnement de RPNI (Oncina *et al.*, 1992) est relativement simple. En entrée, RPNI prend un ensemble E_+ de mots reconnus (ou exemples) par un AFD cible inconnu et un ensemble E_- de mots que cet AFD rejette (contre-exemples). Prenons par exemple $E_+ = \{\lambda, b, ab, abb, bab\}$ et $E_- = \{aa, ba\}$. En sortie, RPNI retourne un AFD qui généralise les données de E_+ tout en rejetant les données de E_-. Sous certaines conditions théoriques, RPNI trouve l'AFD cible qui a produit les données[1]. Pour aboutir à cet AFD final, RPNI entreprend tout d'abord la construction du PTA (*prefix tree acceptor*) des mots de E_+. Ce PTA est le plus grand AFD reconnaissant uniquement les mots de E_+. Ses états sont numérotés selon l'ordre hiérarchique sur les préfixes de E_+ (Oncina *et al.*, 1992). Dans notre exemple, le PTA des mots de E_+ est l'AFD du haut dans la figure 2. Une fois le PTA construit, RPNI parcourt ses états dans l'ordre. Le traitement du $i$ème état consiste à essayer de le fusionner avec les états 0, 1, …, $i - 1$, dans l'ordre. Fusionner deux états consiste à les regrouper en un seul dont le numéro est le minimum entre les deux états fusionnés. Cet état est final si au moins un des deux états l'était avant fusion. Les transitions sortantes des deux états sont elles-mêmes fusionnées deux à deux lorsqu'elles sont étiquetées par la

1. RPNI identifie polynomialement par données fixées la classe des langages rationnels représentée par des AFD (Oncina *et al.*, 1992).

même lettre, et dans ce cas, les deux états qu'elles pointent doivent alors être fusionnés de manière récursive. Dans la figure 2, RPNI réalise la fusion des états 1 et 0 du PTA. Ceci génère une boucle étiquetée par a sur l'état 0. Comme 1 et 0 ont tous deux des transitions sortantes d'étiquette b, les états 3 et 2 qu'elles pointent sont fusionnés, ce qui conduit à l'AFD du milieu dans la figure 2. Une fusion *réussit* si aucun exemple de E_- n'est accepté par l'AFD résultant de cette fusion ; elle *échoue* sinon. Dans notre exemple, la fusion de 1 et 0 échoue puisque $aa \in E_-$ est accepté. Donc RPNI abandonne cette fusion et tente la fusion des états 2 et 0, ce qui conduit à l'AFD du bas dans la figure 2. Celui-ci n'accepte aucun exemple de E_- donc la fusion réussit. RPNI considère donc ce nouvel AFD et tente, avec succès, la fusion des états 3 et 0. Celle-ci conduit à l'AFD de la figure 1, résultat final de RPNI sur les données.

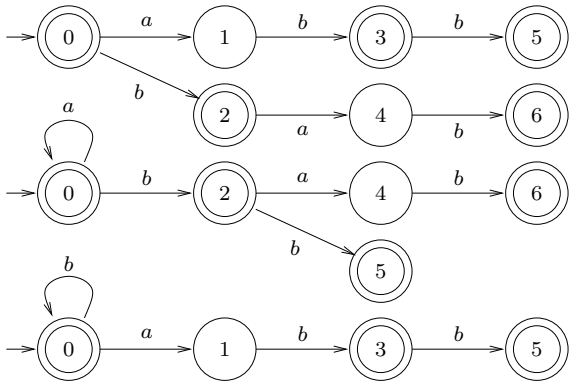

Figure 2. *L'*AFD *du haut est le* PTA *de* $E_+ = \{\lambda, b, ab, abb, bab\}$ *; celui du milieu résulte de la fusion des états 1 et 0 dans le* PTA*, et celui du bas, de la fusion des états 2 et 0*

Plus formellement à présent, analysons le pseudo-code de RPNI dans l'algorithme 2. A chaque étape, RPNI maintient une partition déterministe π des états du PTA A de E_+. L'AFD courant, A/π, reconnaît tous les mots de E_+ et rejette tous les mots de E_-. Par une partition déterministe, nous entendons une partition qui est compatible avec la fonction de transition de A : $\forall p_1, p_2 \in Q, \forall x \in \Sigma, \forall q_1, q_2 \in Q$, si $\pi(p_1) = \pi(p_2)$, $\delta(p_1, x) = q_1$ et $\delta(p_2, x) = q_2$, alors $\pi(q_1) = \pi(q_2)$. $\pi(q)$ désigne l'ensemble des états appartenant au même bloc que l'état q. Fusionner deux états i et j revient à calculer la plus petite partition déterministe π' telle que (1) $\forall s \in Q, \pi'(s) \supseteq \pi(s)$ et (2) $\pi'(i) = \pi'(j)$; elle est obtenue en calculant la fermeture de $\pi \setminus \{\pi(i), \pi(j)\} \cup \{\pi(i) \cup \pi(j)\}$ par la fonction de transition δ du PTA. Cette fusion réussit si l'AFD A/π' n'accepte aucun mot de E_-. Notons que, par construction, A/π' accepte nécessairement tous les mots de E_+, ce qui préserve l'invariant de la boucle principale. Quant aux conditions techniques $i = \texttt{min}(\pi(i))$ et $j = \texttt{min}(\pi(j))$, elles permettent d'éviter de tenter des fusions qui ont déjà été testées.

Supposons maintenant que l'on insère le mot bruité $bbbb$ dans l'ensemble E_- de l'exemple précédent. Si l'automate cible était celui de la figure 1, ce mot devrait être positif. Or la figure 3 (AFD de gauche) présente l'automate profondément modifié que RPNI apprend sur ce nouvel échantillon bruité. Cet exemple jouet montre que RPNI n'est pas immunisé contre le surapprentissage : la présence de données bruitées induit la construction d'AFD avec un plus grand nombre d'états et de mauvaises performances en généralisation. Pour résoudre ce problème, une approche probabiliste a été proposée dans (Sebban *et al.*, 2003), traitant le bruit sur les étiquettes, c'est-à-dire un bruit transformant des mots de E_+ en mots de E_-, et réciproquement (d'autres types de bruit existent mais ne sont pas abordés dans le papier).

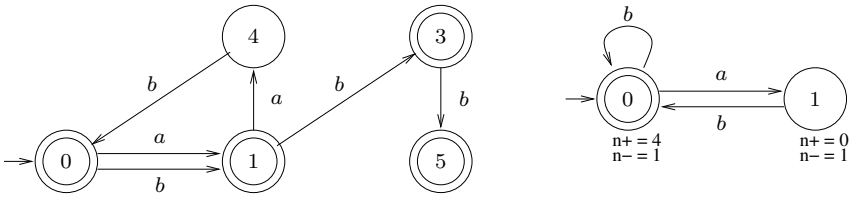

Figure 3. *A gauche,* AFD *inféré par* RPNI *lorsque* $E_+ = \{\lambda, b, ab, abb, bab\}$ *et* $E_- = \{aa, ba, bbbb\}$ *; si l'*AFD *cible est celui de la figure 1, alors* $bbbb$ *devrait être dans* E_+ *; à droite,* AFD *inféré par* RPNI* *;* n^+ *et* n^- *représentent respectivement le nombre d'exemples positifs et négatifs contenus dans chaque état*

Dans (Sebban *et al.*, 2003), les auteurs sont partis du constat suivant : avec RPNI, deux états sont fusionnables si aucun exemple négatif n'est accepté par l'AFD résultant de leur fusion. De manière plus formelle, soit $B = A/\pi = \langle Q, \delta, s_0, F\rangle$ l'AFD résultant de la fusion. Considérons un état s de B et définissons les quantités n_1 et n_2 qui sont respectivement les nombres de mots de E_+ et de E_- contenus par s, *i.e.*, $n_1 = |\{w \in E_+ : \delta(s_0, w) = s\}|$ et $n_2 = |\{w \in E_- : \delta(s_0, w) = s\}|$. Dans RPNI, s est final dès que $n_1 > 0$. De plus, pour qu'une fusion réussisse, il faut nécessairement que $(n_1 > 0 \Longrightarrow n_2 = 0)$. Mais en présence de bruit, il se peut très bien que s, avant le test de compatibilité, contienne plusieurs mots de E_+ et de E_-, *i.e.* que $n_1 > 0$ et $n_2 > 0$, sans pour autant qu'une erreur d'apprentissage soit en train d'être commise. En effet, prenons l'exemple d'un mot w qui appartiendrait à E_+ du fait du bruit. L'état s qui le contient après fusion acquiert le statut d'état final. Donc tout mot de E_- qui est contenu par s joue le rôle de contre-exemple à la fusion. Cette fusion est donc rejetée par RPNI alors qu'elle serait acceptée en l'absence de bruit.

De cette analyse, qui n'est pas exhaustive dans la mesure où d'autres phénomènes peuvent se produire (Sebban *et al.*, 2003), les auteurs ont tiré deux conclusions. D'une part, il faut relâcher la règle d'attribution du statut d'état final à un état, et utiliser désormais la règle de majorité suivante : on dira qu'un état est *positif* (ou final) s'il contient strictement plus de mots positifs (de E_+) que de mots négatifs (de E_-) : $s \in F$ ssi $n_1 > n_2$; on dira qu'il est *négatif* sinon. D'autre part, il faut relâcher la contrainte d'acceptation d'une fusion en permettant à un certain nombre de mots

de E_+ et de E_- d'être mal classés par l'AFD résultant de la fusion : on dira qu'un exemple négatif (resp. positif) est *mal classé* s'il est contenu par un état positif (resp. négatif) ; de plus, on dira qu'une fusion est *statistiquement acceptable* si la proportion p_2 d'exemples mal classés dans tout l'AFD après fusion n'est pas significativement plus grande que la proportion p_1 d'exemples mal classés avant fusion.

Data : Un échantillon d'apprentissage $\langle E_+, E_- \rangle$, un paramètre $\alpha \in [0, 0.5]$
Result : Un AFD
$A \longleftarrow \text{PTA}(E_+)$; $B \longleftarrow A$;
$N \longleftarrow$ Nb d'états de A ; $\pi \longleftarrow \{\{q\} : q \in Q\}$;
for $i = 1$ **to** $N - 1$ **do**
  if $i = \texttt{min}(\pi(i))$ **then**
    $j \longleftarrow 0$; `continue` $\longleftarrow$ `vrai`;
    while $(j < i)$ **and** `continue` **do**
      if $j = \texttt{min}(\pi(j))$ **then**
        $\pi' \longleftarrow$ `calcule_fusion`(π, i, j, δ);
        $B' \longleftarrow$ `attribue_etat_finaux`$(A/\pi', E_+, E_-)$;
        if `statistiquement_compatible`$(B, B', E_+, E_-, \alpha)$ **then**
          $\pi \longleftarrow \pi'$; $B \longleftarrow B'$; `continuer` $\longleftarrow$ `faux`
      $j \longleftarrow j + 1$
return(B)

Algorithme 3: *Pseudo-code de* RPNI*

Cette manière de faire évite ainsi les phénomènes de surapprentissage qui sont dus à la présence de bruit. On accepte donc de réduire, par fusion d'états, la taille des AFD appris si elle n'induit pas d'augmentation significative de l'erreur. Comme les fluctuations d'échantillonnage peuvent avoir une influence lors d'une décision d'acceptation ou de rejet d'une fusion, une simple comparaison entre p_1 et p_2 n'est pas pertinente. Une règle statistique plus judicieuse consiste à utiliser un test de comparaison de proportions avec des intervalles de confiance, dépendant d'un risque de première espèce α. Ce travail a abouti à l'algorithme RPNI* dont le pseudo-code est présenté dans l'algorithme 3. RPNI* fonctionne comme RPNI, à ceci près qu'il attribue le statut d'état final aux états contenant une majorité d'exemples de E_+ (avec la fonction `attribue_etat_finaux`), et qu'il utilise un test statistique de comparaison de proportions pour décider de la réussite ou de l'échec d'une fusion (avec la fonction `statistiquement_compatible`). Remarquons que du point de vue de RPNI*, le risque α est un paramètre de généralisation qu'on peut faire varier entre 0 et 0,5. Une bonne utilisation de ce paramètre consiste donc à tester RPNI* pour différentes valeurs de α, puis à garder le meilleur AFD produit, en termes de taux de succès en généralisation et de nombre d'états. L'AFD de droite de la figure 3 représente le résultat obtenu par RPNI* sur notre problème bruité précédent. On peut ainsi noter que RPNI* est capable d'inférer l'automate cible.

3. Un nouveau schéma de boosting utilisant un oracle de confiance

Dans (Sebban *et al.*, 2003), les auteurs ont montré par leur étude expérimentale que RPNI* améliorait considérablement les performances de RPNI. Cependant, ces bons résultats reposent sur l'hypothèse selon laquelle les exemples mal classés sont réellement bruités. En pratique, il s'avère que cela est souvent le cas, d'où l'amélioration des performances. Mais, de par sa nature statistique, RPNI* peut toujours faire une mauvaise fusion, en considérant qu'un exemple mal classé est bruité alors qu'il ne l'est pas. Cette erreur est mesurable en pratique par le risque de première espèce du test de comparaison de proportions utilisé dans RPNI*. Même si on peut théoriquement réduire ce risque, il ne peut pas être nul, surtout à taille d'échantillon fixé, ce qui peut avoir à nouveau des effets néfastes sur les performances des AFD produits.

Dans la suite de cet article, nous allons tenter d'optimiser les performances de RPNI* à l'aide du boosting. Cette technique d'optimisation n'a jamais été utilisée en inférence grammaticale exacte, et pour cause : on ne peut booster qu'un apprenant faible, soit un algorithme qui commet des erreurs d'apprentissage. Puisque RPNI, tout comme les autres algorithmes d'inférence exacte d'ailleurs, apprend « par cœur » les données, appliquer ADABOOST tel quel n'aurait donc aucun sens. Le développement de RPNI* a permis de supprimer cette contrainte forte. RPNI* devient donc un apprenant faible potentiel. Malgré cela, il ne serait toujours pas judicieux d'appliquer directement ADABOOST sur RPNI*. En effet, il est bien connu que la règle originelle de mise à jour des poids d'ADABOOST, consistant à augmenter le poids des exemples mal classés, n'est pas adaptée aux données bruitées, pourtant les seules justifiant l'utilisation de RPNI*. Plusieurs solutions, nous l'avons vu, ont déjà été proposées pour tenter de traiter le problème du bruit dans le boosting (Friedman *et al.*, 1998; Domingo *et al.*, 2000; Freund, 2001). Souvent, c'est par l'utilisation de fonctions plus douces que la fonction exponentielle que les données bruitées sont gérées. Dans cette section, nous proposons une nouvelle approche originale. Nous supposons pouvoir faire appel à un oracle afin de disposer d'une information supplémentaire sur la confiance en l'étiquette des exemples. Plus précisément, nous supposons que l'oracle est capable de distinguer les exemples non bruités de ceux dont l'étiquette est douteuse ; cela ne signifie pas nécessairement que ces derniers soient bruités, mais qu'il y a au moins des présomptions pour qu'ils le soient. Ce faisant, nous sommes alors capables de séparer les exemples mal classés par une hypothèse au cours d'une itération de boosting en deux parties. D'une part, ceux qui doivent bénéficier d'une augmentation de poids parce que leur étiquette est correcte aux dires de l'oracle, et d'autre part ceux dont il est préférable de baisser le poids, parce que l'oracle émet un doute sur leur étiquette. En d'autres termes, nous préférons baisser à tort le poids d'un exemple mal classé mais non bruité, plutôt que d'augmenter à tord le poids d'un exemple mal classé et effectivement bruité.

La prise en compte de l'information de l'oracle nécessite la mise en place d'une nouvelle stratégie de repondération et d'un nouvel algorithme de boosting que nous allons développer dans la section suivante.

3.1. *Un algorithme de boosting avec oracle*

Soit E un ensemble d'exemples de cardinal $|E| = m$. Chaque exemple x possède une étiquette $y(x)$ pouvant être soit positive ($y(x) = +1$), soit négative ($y(x) = -1$). Un *oracle de confiance* est une fonction q qui associe à tout exemple $x \in E$ une valeur réelle non nulle $q(x)$ dans $[-1, +1]\backslash\{0\}$. L'utilisation de valeurs réelles est justifiée par le fait que l'oracle sera souvent simulé en pratique, que cette simulation ne pourra pas être parfaite, ce dont il faut tenir compte dès à présent. Une confiance positive pour un exemple x signifie que x appartient avec certitude à la classe $y(x)$ qui lui est assignée, donc que x n'est probablement pas bruité, tandis qu'une confiance négative signifie que l'on ne peut rien supposer sur l'étiquette de x. Pour simplifier notre développement, nous supposerons que $q(x)$ n'est pas nul, ce qui signifie que l'oracle ne s'abstient pas, qu'il émet nécessairement un avis sur tous les exemples d'apprentissage ; cette hypothèse peut être levée moyennant un développement théorique plus important, proche de celui de (Schapire *et al.*, 1999, p.8). Les valeurs de $q(.)$ seront supposées constantes au cours du processus d'apprentissage.

Data : Un échantillon d'apprentissage E, un apprenant faible WL, un oracle q, le nombre T de d'hypothèses à construire.
Result : Une combinaison H_T de classifieurs.
for all $x \in E$ **do** $w_0(x) \longleftarrow 1/|E|$;
for $t = 0$ **to** T **do**
  $h_t \longleftarrow \text{WL}(E, \mathbf{w}_t)$;
  $c_t \longleftarrow$ `calcul_coeff_boosting`();
  for all $x \in E$ **do**
    $a_t(x) \longleftarrow$ (**if** $q(x) < 0$ **then** $-q(x)$ **else** $q(x)y(x)h_t(x)$) ;
  for all $x \in E$ **do**
    $w_{t+1}(x) \longleftarrow w_t(x)\exp(-c_t a_t(x))/Z_t$
    où $Z_t = \sum_{e \in E} w_t(x)\exp(-c_t a_t(x))$;
return H_T **such that** $H_T(x) = \text{signe}\left(\sum_{t=0}^{T} c_t h_t(x)\right)$;

Algorithme 4: *Pseudo-code du boosting avec oracle*

Notre algorithme de boosting (voir pseudo-code dans l'algorithme 4) suit le même schéma qu'ADABOOST : on définit une première distribution $\mathbf{w}_0$ uniforme sur l'échantillon d'apprentissage E ; ensuite, à chaque itération t, on utilise WL sur E et $\mathbf{w}_t$ pour obtenir une hypothèse faible h_t et on repondère les exemples d'apprentissage en construisant la distribution $\mathbf{w}_{t+1}$; enfin, au bout de T itérations, on combine les T hypothèses faibles produites en une hypothèse globale H_T « forte ».

Cependant, le schéma de repondération des exemples change sensiblement, dans la mesure où il fait intervenir explicitement les réponses de l'oracle. Par cette mise à jour, nous désirons non seulement augmenter le poids de tout exemple mal classé pour lequel l'oracle se dit confiant, mais également baisser le poids de tout exemple « douteux », qu'il soit bien ou mal classé par l'‘hypothèse courante. La règle de mise

à jour que nous proposons permet de réaliser cela. En effet, pour un exemple x à qui l'oracle attribue une confiance positive ($q(x) > 0$) et qui n'est donc probablement pas bruité, on a :

$$w_{t+1}(x) \quad = \quad \frac{w_t(x)\exp(-c_t q(x) y(x) h_t(x))}{Z_t}.$$

Pour un exemple x de confiance négative ($q(x) < 0$), donc un exemple suspect, on a :

$$w_{t+1}(x) \quad = \quad \frac{w_t(x)\exp(c_t q(x))}{Z_t}.$$

Si nous arrivons à prouver que $c_t > 0$, ce qui sera fait dans le Lemme 2, nous pourrons alors assurer que les objectifs visés seront bien atteints. En effet, dans le cas de confiances positives, les poids des exemples mal classés augmenteront, ceux des bien classés baisseront. Dans le cas de confiances négatives, que l'exemple soit correctement classé ou non, son poids baissera systématiquement, à chaque itération.

Une autre modification majeure intervient dans notre nouvel algorithme de boosting. Elle concerne ce fameux coefficient c_t, différent de celui d'ADABOOST et qui nécessite d'être calculé optimalement pour préserver les propriétés théoriques du boosting. Nous développerons dans ce qui suit deux approches, aboutissant toutes les deux à une même contrainte $(\mathcal{C})$ qui caractérise l'optimalité du choix de c_t. La première, développée dans la section 3.2, consiste à montrer que notre règle de mise à jour des poids est la solution optimale d'un problème de minimisation d'une fonction convexe (divergence de Breigman), quand on choisit correctement c_t. La seconde (section 3.3), plus classique, montre qu'on peut minorer l'erreur en apprentissage de H_T en choisissant cette même valeur optimale de c_t. La contrainte $(\mathcal{C})$ est en fait une équation dont c_t est l'unique solution. Dans le cas d'ADABOOST, cette équation peut être résolue littéralement (en fonction des ϵ_t). Ce n'est pas le cas ici, car notre règle de mise à jour des poids utilise explicitement l'oracle, c'est-à-dire une fonction à valeurs réelles $q(.)$. Néanmoins, nous pourrons obtenir une valeur approchée $\hat{c}_t$ de la valeur théorique de c_t avec une précision quelconque. Pour ce faire, nous commençons par expliciter une hypothèse d'apprentissage faible, propre au boosting avec oracle (section 3.4), ce qui permettra ensuite d'approcher c_t par dichotomie (section 3.5).

Dans les deux derniers paragraphes, nous nous intéressons à l'erreur en apprentissage commise par l'hypothèse globale H_T. Nous commençons par montrer que sous l'hypothèse d'apprentissage faible, et lorsque le coefficient c_t est optimal, cette erreur converge exponentiellement vers 0 avec le nombre T d'itérations (section 3.6). Pour finir, nous donnons une condition sur la précision de l'approximation du c_t théorique par $\hat{c}_t$ qui assure qu'avec $\hat{c}_t$, cette erreur converge toujours vers 0 (section 3.7).

3.2. *Sur la règle de mise à jour des poids*

Soit $\mathcal{P}_m$ l'espace des vecteurs de probabilité de dimension m et $\mathbf{u} \in \mathcal{P}_m$ le vecteur uniforme. Nous supposons l'existence d'un vecteur $\mathbf{w}_t \in \mathcal{P}_m$ qui contient le poids de chaque exemple $x \in E$ à l'étape t, avec $\mathbf{w_0} = \mathbf{u}$ (la distribution initiale étant donc uniforme, comme dans ADABOOST).

Nous savons, depuis (Kivinen *et al.*, 1999), que le boosting standard tel que défini dans (Schapire *et al.*, 1999) est un cas particulier de l'optimisation sous contrainte d'une divergence de Breigman. En replaçant ce problème dans notre cadre, nous établissons l'équation que doit satisfaire le coefficient c_t optimal.

La stratégie du boosting standard est de calculer successivement $\mathbf{w}_{t+1}$ à partir de $\mathbf{w}_t$ en minimisant la *divergence d'information*

$$\mathbf{1}.\mathbf{i}(\mathbf{w}_{t+1}, \mathbf{w}_t) \quad = \quad \sum_{x \in E} i(\mathbf{w}_{t+1}, \mathbf{w}_t)(x),$$

où $\mathbf{i}(.,.)$ est le vecteur de composante

$$i(\mathbf{w}_{t+1}, \mathbf{w}_t)(x) \quad = \quad w_{t+1}(x) \ln \left(\frac{w_{t+1}(x)}{w_t(x)} \right) - w_{t+1}(x) + w_t(x),$$

pour tout $x \in E$. Remarquons que cette divergence d'information est une fonction convexe de $\mathbf{w}_{t+1}$.

Nous souhaitons optimiser $\mathbf{1}.\mathbf{i}(\mathbf{w}_{t+1}, \mathbf{w}_t)$ sous deux contraintes. La première est immédiate ; elle exprime le fait que $\mathbf{w}_{t+1}$ est une distribution, ce qui ne change évidemment pas dans notre nouveau schéma :

$$\mathbf{1}.\mathbf{w}_{t+1} - 1 \quad = \quad 0. \qquad [1]$$

La seconde contrainte intègre une fonction de perte pour chaque exemple de E. Dans le boosting classique, elle exprime le fait que la dernière hypothèse apprise est, en quelque sorte, la plus mauvaise hypothèse sur la nouvelle distribution, ses performances étant en moyenne équivalentes à celles de la décision aléatoire :

$$\mathbf{w}_{t+1}.(\mathbf{y}\mathbf{h}_t) = 0 \quad \Leftrightarrow \quad \sum_{x \in E} w_{t+1}(x) y(x) h_t(x) = 0.$$

Cette contrainte, combinée avec l'*hypothèse d'apprentissage faible* (Kearns *et al.*, 1994), assure que la nouvelle hypothèse h_{t+1} construite sur la distribution $\mathbf{w}_{t+1}$ est significativement différente de h_t, ce qui impose à l'algorithme d'apprendre quelque chose de nouveau sur E, à chaque étape.

Dans notre cas, cette contrainte est différente. On commence par partitionner E en deux sous-ensembles, l'ensemble $E_N = \{x \in E : q(x) < 0\}$ des exemples dont

l'étiquette est douteuse, et l'ensemble $E_{\overline{N}} = E \backslash E_N$ des exemples probablement non bruités, et on pose :

$$\begin{aligned} \mathbf{w}_{t+1}.\mathbf{a}_t &= 0, \text{ avec} \quad [2] \\ \forall x \in E_{\overline{N}}, a_t(x) &= q(x)(yh_t)(x) \text{ et} \\ \forall x \in E_N, a_t(x) &= -q(x). \end{aligned}$$

Pour faire une comparaison avec la seconde contrainte du boosting classique, notons que l'égalité [2] entraîne :

$$\sum_{x \in E_{\overline{N}}} w_{t+1}(x)q(x)y(x)h_t(x) = \sum_{x \in E_N} w_{t+1}(x)q(x).$$

Comme $\forall x \in E_N$, $q(x) < 0$, le membre droit de cette équation est strictement négatif, ce qui rend la dernière hypothèse construite h_t particulièrement mauvaise sur $E_{\overline{N}}$, avec une distribution dépendant à la fois de $\mathbf{w}_{t+1}$ et de $\mathbf{q}$. On voit ici clairement la différence entre notre schéma de repondération et celui du boosting classique qui assure une performance moyenne (comparable au choix aléatoire), uniquement avec la distribution $\mathbf{w}_{t+1}$ sur l'ensemble E tout entier.

La minimisation de la divergence d'information sous les contraintes [1] et [2] est obtenue comme solution ($\forall x \in E$) de :

$$\left[\partial_{\mathbf{w}_{t+1}} \mathbf{i}(\mathbf{w}_{t+1}, \mathbf{w}_t) + b_t \partial_{\mathbf{w}_{t+1}}(\mathbf{1}.\mathbf{w}_{t+1} - 1) + c_t \partial_{\mathbf{w}_{t+1}}(\mathbf{w}_{t+1}.\mathbf{a}_t)\right](x) = 0, \quad [3]$$

où b_t et c_t sont des multiplicateurs de Lagrange. La résolution de [3] amène, $\forall x \in E$:

$$\left(\ln \frac{\mathbf{w}_{t+1}}{\mathbf{w}_t} + b_t \mathbf{1} + c_t \mathbf{a}_t\right)(x) = 0 \quad \Leftrightarrow \quad w_{t+1}(x) = \frac{w_t(x) \exp(-c_t a_t(x))}{\exp(b_t)}.$$

La contrainte [1] permet de déduire :

$$b_t = \ln \sum_{x \in E} w_t(x) \exp(-c_t a_t(x)), \quad [4]$$

Le terme dans le $\ln(.)$ correspond au coefficient de normalisation pour $\mathbf{w}_{t+1}$:

$$Z_t = \sum_{x \in E} w_t(x) \exp(-c_t a_t(x)). \quad [5]$$

La dernière inconnue c_t est obtenue par la contrainte [2] comme solution de :

$$\sum_{x \in E} w_t(x) a_t(x) \exp(-c_t a_t(x)) = 0. \quad [6]$$

3.3. *Résultat sur l'erreur en apprentissage*

La valeur de c_t est donc obtenue comme solution de l'équation [6]. On peut remarquer que cette équation est strictement équivalente à :

$$\left(\frac{\partial Z_t}{\partial c_t}\right) \quad = \quad 0.$$

Comme Z_t est une fonction convexe de c_t, résoudre [6] revient donc à chercher le coefficient c_t qui minimise Z_t. Ce problème de minimisation est intimement lié avec celui de la minimisation de l'erreur en apprentissage commise par H_T.

Commençons par définir cette erreur : puisque l'étiquette des exemples de E_N n'est pas fiable, nous définissons l'erreur empirique de notre algorithme de boosting sur $E_{\overline{N}}$ uniquement. Aussi, on pose $\forall T > 0$,

$$\varepsilon(E_{\overline{N}}, H_T) \quad = \quad \frac{1}{|E_{\overline{N}}|} \sum_{x \in E_{\overline{N}}} [\![y(x) \neq H_T(x)]\!], \qquad [7]$$

où $[\![\pi]\!]$ désigne la valeur de vérité d'un prédicat π (dans $\{0, 1\}$).

Nous obtenons le résultat suivant :

Lemme 1

$$\varepsilon(E_{\overline{N}}, H_T) \quad \leq \quad \frac{m}{|E_{\overline{N}}|} \left(\prod_{t=0}^{T} Z_t \right).$$

Preuve : comme $\forall x \in E_{\overline{N}}$, $q(x) > 0$, nous avons pour tous ces exemples :

$$[\![H_T(x) \neq y(x)]\!] \quad \leq \quad \exp(-q(x)y(x) \sum_{t=0}^{T} c_t h_t(x)).$$

D'autre part, en déroulant la règle de mise à jour, on obtient, $\forall x \in E_{\overline{N}}$:

$$w_{T+1}(x) \quad = \quad \frac{w_0(x) \exp(-q(x)y(x) \sum_{t=0}^{T} c_t h_t(x))}{\left(\prod_{t=0}^{T} Z_t \right)}.$$

Par conséquent $[\![H_T(x) \neq y(x)]\!] \leq m w_{T+1}(x) (\prod_{0 \leq t \leq T} Z_t)$. En sommant pour tout $x \in E_{\overline{N}}$, nous obtenons $\varepsilon(E_{\overline{N}}, H_T) \leq (m/|E_{\overline{N}}|)(\prod_{0 \leq t \leq T} Z_t)(\sum_{x \in E_{\overline{N}}} w_{T+1}(x))$. Enfin, en utilisant le fait que $\sum_{x \in E_{\overline{N}}} w_{T+1}(x) \leq 1$, nous obtenons le Lemme. □

Comme le terme $(m/|E_{\overline{N}}|)$ est constant pour E et $E_{\overline{N}}$, le Lemme 1 signifie que pour minimiser l'erreur sur $E_{\overline{N}}$, il faut se concentrer sur la minimisation de chaque Z_t. Le constat est désormais clair : résoudre [6] nous permet d'obtenir non seulement la solution de [3], mais également la minimisation de l'erreur sur $E_{\overline{N}}$ au regard du Lemme 1.

3.4. *Une hypothèse d'apprentissage faible adaptée*

Pour trouver une solution de [6], il nous faut maintenant formuler une hypothèse :

• $\forall t \geq 0$, il existe une constante $\gamma_t > 0$ telle que :

$$\frac{\sum_{x \in E_{\overline{N}}} (w_t q y h_t)(x)}{\sum_{x \in E_{\overline{N}}} (w_t q)(x)} \quad = \quad \gamma_t. \qquad [8]$$

En suivant la terminologie classique du boosting, [8] doit être vue comme une hypothèse d'apprentissage faible (*Weak Learning Assumption*, WLA). Notons que si h_t correspond au choix aléatoire, alors $\gamma_t = 0$. En d'autres termes, [8] traduit le fait que h_t doit être légèrement plus performante que le choix aléatoire pour pouvoir être considérée comme une hypothèse faible.

Notre WLA est différente de celle du boosting classique (Kearns *et al.*, 1994) pour deux raisons. Tout d'abord, elle est locale : elle se concentre sur un sous-ensemble de E. Ensuite, elle s'appuie sur une distribution qui n'est pas exactement $\mathbf{w}_t$ mais une distribution $\mathbf{w}_{q,t} \in \mathcal{P}_m$ paramétrée par q qu'on définit ainsi, $\forall x \in E$:

$$w_{q,t}(x) \quad = \quad \frac{|q(x)| w_t(x)}{Q_t}, \text{ avec } Q_t = \sum_{x \in E} |q(x)| w_t(x).$$

Notons que $\mathbf{w}_{q,t} \in \mathcal{P}_m$, car tous les exemples ont une confiance $q(.)$ différente de 0. Nous étendons cette définition en posant, pour tout $E' \subseteq E$,

$$W_{q,t}(E') \quad = \quad \sum_{x \in E'} w_{q,t}(x).$$

Afin de compléter la discussion autour de la contrainte [2] que nous avons commencée dans la section 3.2, nous définissons

$$E^+_{\overline{N},t} \quad = \quad \{x \in E_{\overline{N}} : (y h_t)(x) = +1\}, \text{ et}$$
$$E^-_{\overline{N},t} \quad = \quad \{x \in E_{\overline{N}} : (y h_t)(x) = -1\}.$$

La WLA s'écrit alors :

$$W_{q,t}(E^+_{\overline{N},t}) - W_{q,t}(E^-_{\overline{N},t}) \quad = \quad \gamma_t W_{q,t}(E_{\overline{N}}). \qquad [9]$$

Nous ne pouvons donc pas choisir $h_{t+1} = h_t$ sous la WLA, car sinon [2] impliquerait que $W_{q,t+1}(E^+_{\overline{N},t}) - W_{q,t+1}(E^-_{\overline{N},t}) < 0$, tandis que [9] impliquerait $W_{q,t+1}(E^+_{\overline{N},t}) - W_{q,t+1}(E^-_{\overline{N},t}) > 0$. Comme dans le cadre classique d'ADABOOST, notre WLA force donc l'algorithme d'apprentissage à retourner une hypothèse h_{t+1} différente de h_t sur $E_{\overline{N}}$, donc à apprendre quelque chose de nouveau.

3.5. *Approximation des coefficients* c_t

Sous notre WLA, nous obtenons le résultat suivant :

Lemme 2 *La solution de [6] est unique, strictement positive, et elle vérifie :*

$$\frac{1}{2\overline{q}} \ln\left(\frac{1 - W_{q,t}(E^-_{\overline{N},t})}{W_{q,t}(E^-_{\overline{N},t})}\right) \leq c_t \leq \frac{1}{2\underline{q}} \ln\left(\frac{1 - W_{q,t}(E^-_{\overline{N},t})}{W_{q,t}(E^-_{\overline{N},t})}\right),$$

où $\underline{q} = \min_{x \in E} |q(x)|$ *et* $\overline{q} = \max_{x \in E} |q(x)|$.

En d'autres termes, $c_t = r \ln((1 - W_{q,t}(E^-_{\overline{N},t}))/W_{q,t}(E^-_{N,t}))$ est solution de l'équation [6] pour un certain $r \in [1/2\overline{q}, 1/2\underline{q}]$. Cette borne est plus précise que celle donnée dans (Schapire *et al.*, 1999) (où $c_t \in \mathbb{R}$).

De plus, elle permet une approximation très rapide de c_t par dichotomie. Supposons qu'on veuille approximer c_t par un $\hat{c}_t$ tel que $|\hat{c}_t - c_t|/|c_t| < \epsilon$ pour un $0 < \epsilon \leq 1$. L'approximation dichotomique se fera alors en au plus $\mathcal{O}(\ln(\overline{q}/\underline{q}) + \ln(1/\epsilon))$ étapes. Cette efficacité est d'autant plus appréciable que ce calcul devra être fait à chaque itération de boosting, soit T fois.

Preuve : posons

$$g(c) = \sum_{x \in E} (w_t a_t)(x) \exp(-c a_t(x)). \qquad [10]$$

$g'(c) = -\sum_{x \in E} (w_t a_t^2)(x) \exp(-c a_t(x)) < 0$. Donc $g(c)$ est strictement décroissante sur $\mathbb{R}$. Comme $\lim_{c \to +\infty} g(c) = -\infty$ (à la condition qu'il existe $x \in E$ tel que $a_t(x) < 0$) et $\lim_{c \to -\infty} g(c) = +\infty$ (à la condition qu'il existe $x \in E$ tel que $a_t(x) > 0$), l'équation [6] admet une solution unique c_t. De plus,

$$\begin{aligned} \frac{g(0)}{Q_t} &= W_{q,t}(E^+_{\overline{N},t}) - W_{q,t}(E^-_{\overline{N},t}) + W_{q,t}(E_N) \\ &= \gamma_t W_{q,t}(E_{\overline{N}}) + W_{q,t}(E_N) \\ &= \gamma_t + (1 - \gamma_t) W_{q,t}(E_N). \end{aligned} \qquad [11]$$

Sous la WLA, on a donc $g(0) > 0$, et comme g est strictement décroissante, on en déduit que $c_t > 0$. Enfin,

$$\frac{g(c_t)}{Q_t} = 0 = \sum_{x \in E^+_{\overline{N},t} \cup E_N} w_{q,t}(x) \exp\left(-c_t |q(x)|\right) - \sum_{x \in E^-_{\overline{N},t}} w_{q,t}(x) \exp\left(c_t |q(x)|\right).$$

Comme $\underline{q} \leq |q(x)| \leq \overline{q}$ pour tout $x \in E$, nous obtenons :

$$W_{q,t}(E^+_{\overline{N},t} \cup E_N) e^{-c_t \overline{q}} - W_{q,t}(E^-_{\overline{N},t}) e^{c_t \overline{q}} \leq 0 \quad \text{et}$$
$$W_{q,t}(E^+_{\overline{N},t} \cup E_N) e^{-c_t \underline{q}} - W_{q,t}(E^-_{\overline{N},t}) e^{c_t \underline{q}} \geq 0.$$

Comme $W_{q,t}(E^+_{\overline{N},t} \cup E_N) = 1 - W_{q,t}(E^-_{\overline{N},t})$, on en déduit le résultat. □

3.6. *Convergence théorique de l'erreur en apprentissage*

Sous notre WLA, nous pouvons maintenant établir le lemme suivant :

Lemme 3 $\forall t \geq 0$, *en posant* $\sigma_t = \gamma_t \frac{\underline{q}}{\overline{q}}$, *on a :*

$$Z_t \leq \sqrt{1 - \sigma_t^2} < \exp\left(-\frac{\sigma_t^2}{2}\right).$$

Comme nous l'avons évoqué au début de cette section, nous supposons que la fonction $q(.)$, donc $\underline{q}$ et $\overline{q}$, sont des constantes au cours des itérations du boosting. Aussi, sous la WLA, la valeur maximale de Z_t est toujours < 1. Or comme $\varepsilon(E_{\overline{N}}, H_T) \leq \frac{m}{|E_{\overline{N}}|} \left(\prod_{t=0}^{T} Z_t\right)$, ceci nous garantit une convergence exponentielle de l'erreur en apprentissage de H_T vers 0.

Preuve : on a l'inégalité suivante, $\forall x \in [-1, 1], \forall \eta \in \mathbb{R}$:

$$\exp(-\eta x) \leq \frac{1+x}{2} \exp(-\eta) + \frac{1-x}{2} \exp(\eta), \qquad [12]$$

En effet, la fonction $x \mapsto \exp(-\eta x)$ est convexe et le membre droit de l'inégalité est l'équation de la droite passant par les points de coordonnées $(-1, \exp(\eta))$ et $(1, \exp(-\eta))$. En posant $\eta = c\overline{q}$ et $x = a_t(x)/\overline{q}$, nous obtenons $\forall x \in E$:

$$\exp(-c a_t(x)) \leq \frac{\overline{q} + a_t(x)}{2\overline{q}} \exp(-c\overline{q}) + \frac{\overline{q} - a_t(x)}{2\overline{q}} \exp(c\overline{q}). \qquad [13]$$

En appelant $\ell(x, c)$ le membre droit de l'inéquation [13], nous obtenons donc :

$$Z_t \leq \inf_{c \in \mathbb{R}} \sum_{x \in E} w_t(x) \ell(x, c). \quad [14]$$

Le c minimisant le membre droit de l'inéquation [14] est

$$c = \frac{1}{2\overline{q}} \ln \left(\frac{\overline{q} + \sum_{x \in E} (w_t a_t)(x)}{\overline{q} - \sum_{x \in E} (w_t a_t)(x)} \right) = \frac{1}{2\overline{q}} \ln \left(\frac{\overline{q} + g(0)}{\overline{q} - g(0)} \right),$$

où $g(.)$ est la fonction définie dans l'équation [10]. En remplaçant c par cette valeur dans l'inéquation [14], on obtient :

$$Z_t \leq \sqrt{1 - \left(\frac{g(0)}{\overline{q}} \right)^2}, \quad [15]$$

Enfin, en utilisant le fait que $g(0) \geq \gamma_t Q_t$ (d'après l'équation [11]) et que $Q_t \geq \underline{q}$, nous obtenons le résultat du lemme. □

3.7. *Convergence, en pratique, de l'erreur en apprentissage*

La borne du Lemme 3 est purement théorique, dans la mesure où c_t ne peut pas être calculé de manière exacte, mais seulement approché par une valeur $\hat{c}_t$, calculée par dichotomie. Aussi, la valeur exacte de Z_t ne peut pas non plus être calculée, mais seulement approchée par une valeur $\hat{Z}_t = \sum_{x \in E} w_t(x) \exp(-\hat{c}_t a_t(x))$. Or une approximation trop grossière de $\hat{c}_t$ ne permettra certainement pas d'assurer que $\hat{Z}_t < 1$, donc la convergence de notre schéma de boosting en pratique. Notre dernier résultat montre que $\hat{Z}_t < 1$ dès que $|\hat{c}_t - c_t| < \exp\left(-\frac{\sigma_t^2}{2}\right)$. Nous obtenons plus précisément le résultat suivant :

Lemme 4 $\forall t \geq 0$,

$$\textit{si } |\hat{c}_t - c_t| \leq \frac{(\ln 2)\sigma_t^2}{4\overline{q}_t}, \textit{ alors } \hat{Z}_t \leq \sqrt{1 - \frac{\sigma_t^2}{2}} < \exp\left(-\frac{\sigma_t^2}{4}\right).$$

Preuve : soit $\zeta(y) = \sum_{x \in E} w_t(x) \exp(-y a_t(x))$. En développant ζ en séries entières, on obtient $\forall y \in \mathbb{R}$,

$$\zeta(y) = \sum_{k=0}^{+\infty} \frac{\partial^k \zeta}{\partial y^k}(c_t) \frac{(y - c_t)^k}{k!} \leq \sum_{k=0}^{+\infty} \left| \frac{\partial^k \zeta}{\partial y^k}(c_t) \right| \frac{|y - c_t|^k}{k!} \quad .$$

Comme $\left|\frac{\partial^k \zeta}{\partial y^k}(c_t)\right| \leq \overline{q}_t^k Z_t$ pour tout $k \geq 0$, on en déduit

$$\zeta(y) \leq Z_t \left(\sum_{k=0}^{+\infty} \frac{\overline{q}_t^k |y - c_t|^k}{k!} \right) = Z_t \exp\left(\overline{q}_t |y - c_t|\right) \ ,$$

donc $\hat{Z}_t \leq Z_t \exp\left(\overline{q}_t |\hat{c}_t - c_t|\right)$. Clairement, $\forall \epsilon > 0$, si $|\hat{c}_t - c_t| \leq \ln(1+\epsilon)/\overline{q}_t$, alors $\hat{Z}_t \leq (1+\epsilon) Z_t$. Donc en posant $\epsilon = \frac{\sigma_t^2}{4}$ et en utilisant le Lemme 3, on obtient

$$\hat{Z}_t \quad \leq \quad \sqrt{1 - \frac{\sigma_t^2}{2}} \qquad [16]$$

Pour finir, on peut remarquer que $\ln(1+\epsilon) \geq (\ln 2)\epsilon$ pour tout $\epsilon \in [0,1]$, du fait de la concavité de la fonction $x \mapsto \ln(1+x)$. Donc l'inéquation [16] est satisfaite dès que $|\hat{c}_t - c_t| \leq \frac{(\ln 2)\sigma_t^2}{4\overline{q}_t}$. □

4. D'un graphe de voisinage vers un oracle

Le cadre théorique que nous venons de proposer nécessite de pouvoir faire appel à un oracle. En pratique, nous pouvons assimiler celui-ci à un expert du domaine étudié capable de certifier ou non la légitimité de l'étiquette d'un exemple. Son existence est tout à fait crédible dans les domaines où le niveau d'expertise est suffisant. Ceci n'est malheureusement pas toujours le cas, et afin de pouvoir appliquer notre schéma de boosting sur un large spectre de problèmes réels, nous allons devoir parfois simuler le comportement d'un oracle. Dans cette section, nous décrivons une approche empirique permettant d'effectuer cette simulation qui devra conserver les comportements théoriques décrits précédemment. L'objectif principal est de s'assurer qu'une confiance positive donnée par l'oracle correspond à un exemple dont l'étiquette est très probablement correcte. Nous proposons ici une stratégie qui pourrait s'apparenter en quelque sorte aux méthodes dites de *co-training* (Blum *et al.*, 1998). Ces méthodes sont utilisées lorsque l'on dispose d'une certaine quantité d'exemples, dont seule une petite proportion est étiquetée. On suppose par ailleurs qu'il existe une partition naturelle des attributs des exemples en deux sous-ensembles d'attributs, ce qui revient à considérer que l'on dispose de deux représentations différentes des données. Un algorithme d'apprentissage auxiliaire apprend un modèle à partir d'une des deux représentations des exemples étiquetés. Ce modèle est alors utilisé pour étiqueter les exemples qui ne le sont pas encore. Un algorithme d'apprentissage utilise ensuite le deuxième sous-ensemble d'attributs, et la totalité des exemples, désormais tous étiquetés, pour construire un modèle final. Dans notre cas, l'oracle joue le rôle de l'algorithme d'apprentissage auxiliaire, qui apprend à partir d'une nouvelle représentation des données, comme nous le verrons ci-dessous. Mais plutôt que d'étiqueter des exemples qui ne le sont pas, il fournit une information supplémentaire sur la confiance en l'étiquette

des exemples. Nous proposons ici une stratégie basée sur un graphe de voisinage, qui permet de distinguer géométriquement les exemples douteux des exemples sûrs.

4.1. *Une approche géométrique de la confiance*

Nous pensons que les graphes des k plus proches voisins (*k*PPV) (Cover *et al.*, 1967) sont particulièrement adaptés pour traiter un tel problème. Rappelons que la règle de classification des *k*PPV attribue la classe d'un exemple x en calculant la classe majoritaire parmi les k exemples de E qui sont les plus proches de x. Cette méthode, au-delà du fait que son erreur limite est théoriquement bornée par deux fois l'erreur bayésienne, est très résistante au bruit. En effet, un exemple bruité isolé au milieu d'exemples de la classe opposée a un impact très limité sur la règle de classification. La propriété duale que l'on peut déduire de la remarque précédente, est qu'un exemple bruité peut être plus facilement détecté en analysant son voisinage. Définissons la fonction de marge $m(x)$ d'un exemple x de classe $y(x)$ calculée à l'aide d'un graphe des *k*PPV :

$$m(x) \quad = \quad \frac{N_{y(x)} - N_{\overline{y}(x)}}{N_{y(x)} + N_{\overline{y}(x)}},$$

où $N_{y(x)}$ (resp. $N_{\overline{y}(x)}$) désigne le nombre d'exemples parmi les k plus proches voisins de x étant de la même classe $y(x)$ (resp. étant de la classe opposée $\overline{y}(x)$). Un exemple n'ayant que des voisins de la classe opposée à la sienne (resp. de la même classe) recevra une marge de -1 (resp. $+1$). Même si $m(x)$ semble représenter d'une certaine manière la valeur de confiance $q(x)$ utilisée dans la section précédente, poser $q(x) = m(x)$ ne serait pas judicieux. En effet, la règle de classification d'un *k*PPV permet « seulement » d'assurer que l'erreur limite sera bornée par deux fois l'erreur bayésienne, et ce n'est pas ce qui nous intéresse ici. Nous cherchons à fournir une mesure pertinente de la confiance des exemples, en nous assurant que les valeurs positives de $q(x)$ correspondent aux x ayant une étiquette correcte, dans $(100 - \epsilon)$ % des cas (avec ϵ très petit). Si $m(x)$ est légèrement plus grande que 0, cette condition est loin d'être respectée (bien que la marge soit positive). Pour contourner cet obstacle, nous fixons un paramètre $\beta \in [0, 1[$, et nous calculons la confiance comme suit :

– $q(x) = (m(x) - \beta)/(1 - \beta)$ $\forall x$ satisfaisant $\beta < m(x) \leq 1$.
x aura donc une confiance positive si et seulement si $m(x) > \beta$.

– $q(x) = (m(x) - \beta)/(1 + \beta)$ $\forall x$ satisfaisant $-1 \leq m(x) < \beta$,
et donc x recevra une marge négative si $m(x) < \beta$.

En d'autres termes, les confiances élevées seront assignées uniquement aux exemples ayant une large majorité d'exemples de leur classe dans leur voisinage. En revanche, les exemples peu pertinents, ayant une marge faiblement positive, se verront attribuer une confiance négative. Ce choix est motivé, dans le cadre de l'inférence grammaticale, par le fait qu'il est préférable de réduire l'impact des données pertinentes, plutôt que d'inférer un AFD à partir de données bruitées. Notons donc que le choix de

la valeur de β devra tenir compte du compromis à trouver entre le désir de réduire au maximum le risque de booster des exemples bruités, et celui de conserver suffisamment d'exemples de confiance positive sur lesquels seulement la convergence du processus s'opère.

4.2. *Fonction de distance entre mots*

Rappelons que notre objectif est ici d'optimiser les performances de l'algorithme d'inférence grammaticale RPNI* qui manipule des mots. Avant de calculer les valeurs de confiance, on doit donc pouvoir utiliser une distance entre mots pour construire le graphe de voisinage. La distance la plus utilisée dans cette situation est probablement la distance d'édition. Cependant, nous avons observé dans des expérimentations préliminaires que nous pouvions obtenir de meilleurs résultats en utilisant des distances « orientées données » construites après une transformation de l'espace de représentation. Nous avons décidé d'utiliser des bigrammes comme modèles de langage, afin de décrire les mots sous forme de vecteurs numériques. Les bigrammes sont des modèles de représentation de connaissances qui ont prouvé leur utilité dans le domaine de la reconnaissance de données textuelles. Ils estiment la probabilité d'un mot relativement à un échantillon donné, et permettent donc d'appliquer des méthodes géométriques, ou d'autres méthodes mathématiques, qui sont plus appropriées pour des vecteurs que pour des mots. Nous présentons ci-après le principe de fonctionnement des bigrammes. La probabilité d'une lettre dans un mot dépend seulement de la lettre précédente dans ce mot. Etant donné un mot de n lettres, $w = x_1x_2x_3 \ldots x_n$, sa probabilité est :

$$p(w) \quad = \quad \prod_{i=0}^{n} p(x_{i+1}/x_i).$$

Les probabilités élémentaires $p(x_i/x_{i-1})$ sont estimées à partir des mots de l'échantillon d'apprentissage. Enfin, pour donner un sens aux probabilités initiales et finales $p(x_1/x_0)$ et $p(x_{n+1}/x_n)$, deux caractères particuliers, <BOW> (pour Beginning Of the Word) et <EOW> (pour End Of the Word), sont ajoutés à l'alphabet, et l'on pose $x_0 =$<BOW> et $x_{n+1} =$<EOW>.

Afin de permettre une visualisation de nos données, et par là-même évaluer graphiquement la qualité de notre oracle simulé, nous avons choisi de construire une représentation géométrique des mots dans un espace euclidien à deux dimensions. Nous désignons par $P_+(w)$ et $P_-(w)$ les probabilités de w relativement aux sous-classes positives et négatives de E. Nous estimons ces probabilités par deux bigrammes construits par comptage sur des données positives (pour $P_+(w)$) et négatives (pour $P_-(w)$) d'un échantillon d'exemples. Dans le but de permettre la comparaison entre deux mots de longueurs différentes, nous normalisons chacune de ces probabilités en utilisant la moyenne géométrique : $P'_+(w) = \sqrt[n+1]{P_+(w)}$ et $P'_-(w) = \sqrt[n+1]{P_-(w)}$. Etant données les valeurs $P'_+(w)$ et $P'_-(w)$, nous pouvons maintenant :

1) projeter w en un point de coordonnées $P'_+(w)$ et $P'_-(w)$ dans un espace euclidien à deux dimensions. La figure 4 présente un exemple de cette projection pour une base de prénoms français dans laquelle les exemples positifs sont les prénoms féminins, et les exemples négatifs sont les prénoms masculins. Il est à noter ici que ce problème de prénoms français est difficile à apprendre par les algorithmes d'inférence grammaticale exacte. Les performances de base de RPNI ou RPNI* sont en effet très faibles. Or on constate que la projection des exemples dans $\mathbb{R}^2$ permet manifestement de séparer correctement les deux classes. Ceci laisse présager de bonnes performances en tant qu'oracle pour un graphe de voisinage construit dans cet espace. Par ailleurs, comme nous l'avons déjà mentionné, le fait d'utiliser un espace à deux dimensions nous permet d'estimer visuellement la qualité de la projection. Il est cependant envisageable de projeter les exemples dans un espace de dimension plus élevée. Il faut pour cela utiliser comme attributs des informations numériques permettant de discriminer encore plus les exemples, la question restant ouverte sur la nature de ces informations ;

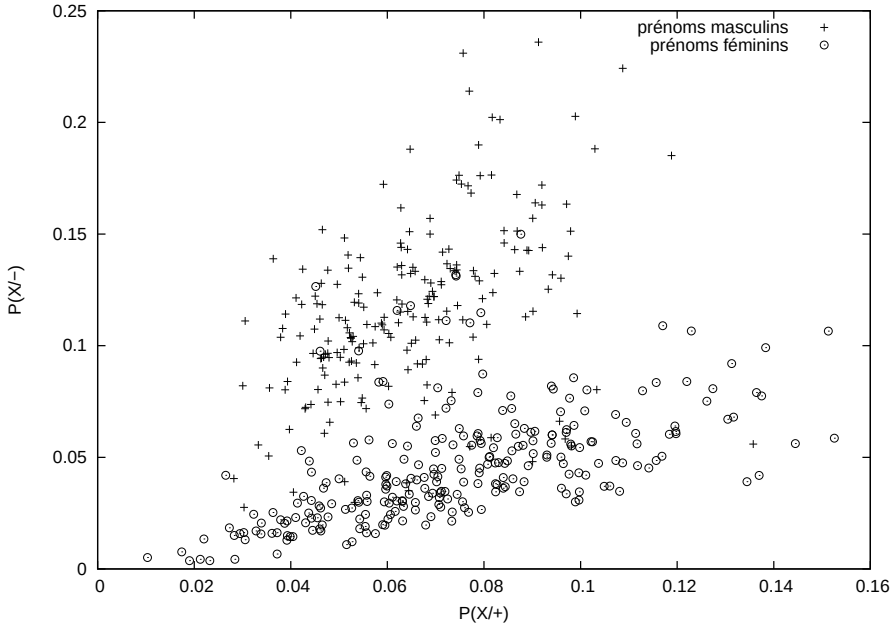

Figure 4. *Un exemple de projection d'une base d'exemples dans un espace euclidien à deux dimensions à l'aide de bigrammes*

2) construire un graphe de voisinage en utilisant la distance euclidienne usuelle dans l'espace de projection. La figure 5 montre une partie du graphe de voisinage construit à partir de la projection de la base de prénoms français présentée précédemment. Ce graphe présente effectivement des regroupements entre prénoms du même genre, et des éloignements entre prénoms de genre opposé. Rappelons toutefois que ce sont les capacités du graphe à fournir des informations sur la confiance en l'étiquette des exemples que nous souhaitons exploiter. Si ses performances en généralisation, en

tant que classifieur, sont une bonne indication de ces capacités, elles ne seront pas au centre de notre étude. Le débat n'est donc pas de savoir si un graphe des *k*PPV construit sur les mots est plus performant en tant que classifieur qu'un algorithme d'inférence grammaticale. Le but est plutôt d'exploiter au mieux ce graphe comme oracle pour optimiser RPNI*.

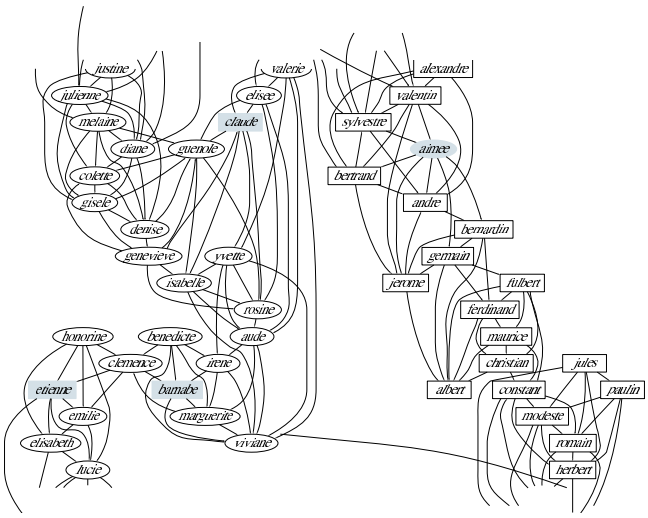

Figure 5. *Une partie du graphe de voisinage construit sur la projection de la base de prénoms français*

5. Expérimentations et résultats

Nos expérimentations ont été guidées par la nécessité de valider nos résultats théoriques, ainsi que d'estimer les performances d'un graphe de voisinage en tant qu'oracle. Pour effectuer cette tâche, nous avons utilisé trois sortes de bases de données. Les premières sont des benchmarks connus du répertoire UCI[2] : AGARICUS et TICTACTOE. Les secondes sont des bases de données réelles. L'une, appelée USF, provient des Etats-Unis, et recense les mille prénoms les plus fréquemment donnés aux USA en 2002. L'autre, appelée WF, contient tous les prénoms commençant par la lettre « a ». Pour ces deux bases de données, le concept à apprendre est le sexe correspondant au prénom. La dernière sorte de bases est constituée de bases artificielles. La taille de l'alphabet utilisé pour les générer varie de quatre à huit lettres selon la base. Deux méthodes de construction sont utilisées. Il s'agit pour chacune d'elle de générer aléatoirement des exemples dont la longueur est comprise dans un intervalle défini, et d'étiqueter comme positifs ceux comportant des *motifs* (*i.e.* des séquences de lettres)

2. http ://www.ics.uci.edu/~mlearn/MLRepository.html

fixés. La taille de ces motifs varie de 4 à 6 caractères d'une base à l'autre. La première des deux techniques utilise ensuite d'autres motifs pour étiqueter comme négatifs les exemples qui les contiennent, tandis que la deuxième technique étiquette comme négatifs les exemples ne contenant pas les motifs utilisés pour les exemples positifs. Nous avons généré sept bases de tailles variables ; les six premières contiennent 3 000 mots environ, tandis que la dernière, destinée à évaluer l'intérêt de notre méthode sur un plus grand volume de données, contient 10 000 exemples.

Pour des raisons de temps de calculs importants, liés à la nature même du boosting, nous n'avons pas effectué ici de procédure de validation croisée. Pour chacune des onzes bases, les exemples ont été répartis aléatoirement en cinq ensembles de taille égale. Quatre de ces ensembles ont été regroupés en un échantillon d'apprentissage, tandis que le dernier a servi d'échantillon de validation.

Finalement, nous avons artificiellement introduit du bruit dans l'ensemble des onze échantillons d'apprentissage, en retournant l'étiquette d'un certain pourcentage de mots. L'estimation des performances en généralisation se fait sur l'échantillon de validation non bruité.

		5 %					10 %				
BASE	TAILLE	SHIFT	RPNI	RPNI*	BOOST	PERF	SHIFT	RPNI	RPNI*	BOOST	PERF
AGARICUS	5 644	5,0 %	7,2 %	7,2 %	4,3 %	0,0 %	6,2 %	14,0 %	10,0 %	6,9 %	0,0 %
TICTACTOE	809	38,9 %	39,5 %	34,5 %	28,3 %	4,9 %	42,6 %	41,9 %	40,7 %	34,5 %	10,5 %
USF	1 871	32,3 %	33,6 %	31,2 %	26,1 %	26,4 %	30,0 %	35,7 %	33,0 %	31,2 %	31,7 %
WF	1 887	30,9 %	29,3 %	25,3 %	21,4 %	20,6 %	34,8 %	29,8 %	31,2 %	22,7 %	26,1 %
BASE1	2 540	42,3 %	46,4 %	41,5 %	41,1 %	35,6 %	43,9 %	52,1 %	44,8 %	43,7 %	36,2 %
BASE2	1 505	21,6 %	48,9 %	21,9 %	19,9 %	14,3 %	13,6 %	39,2 %	13,6 %	30,5 %	12,3 %
BASE3	1 532	27,0 %	43,9 %	27,0 %	26,7 %	13,0 %	39,0 %	39,0 %	39,0 %	12,7 %	12,7 %
BASE4	2 969	1,0 %	39,7 %	2,0 %	10,0 %	0,0 %	23,9 %	45,7 %	29,2 %	23,4 %	0,0 %
BASE5	2 179	36,5 %	36,7 %	36,7 %	22,2 %	19,2 %	18,6 %	45,9 %	18,6 %	25,0 %	16,2 %
BASE6	2 004	7,7 %	42,9 %	7,0 %	13,0 %	2,0 %	42,9 %	45,9 %	42,9 %	25,6 %	1,2 %
BASE7	10 000	46,6 %	46,1 %	39,7 %	37,2 %	39,7 %	40,3 %	49,4 %	42,7 %	43,7 %	39,4 %
MOYENNE	2 994,5	26,3 %	37,7 %	24,9 %	22,7 %	16,0 %	30,5 %	39,9 %	31,4 %	27,3 %	16,9 %
		15 %					20 %				
AGARICUS	5 644	9,9 %	14,0 %	10,0 %	6,1 %	0,0 %	13,3 %	23,2 %	18,2 %	11,9 %	0,0 %
TICTACTOE	809	40,7 %	44,0 %	40,7 %	40,7 %	9,8 %	41,9 %	40,1 %	40,1 %	38,8 %	11,7 %
USF	1 871	32,3 %	34,9 %	32,0 %	27,7 %	27,7 %	34,4 %	48,8 %	32,2 %	30,6 %	28,8 %
WF	1 887	33,3 %	32,5 %	36,2 %	24,6 %	24,6 %	34,6 %	35,4 %	32,0 %	31,7 %	22,7 %
BASE1	2 540	40,6 %	47,0 %	42,9 %	42,3 %	38,5 %	42,9 %	51,2 %	45,7 %	45,2 %	35,2 %
BASE2	1 505	39,7 %	39,8 %	39,8 %	37,5 %	10,6 %	44,9 %	46,8 %	46,8 %	45,1 %	12,6 %
BASE3	1 532	41,0 %	47,0 %	39,4 %	35,5 %	14,9 %	49,5 %	49,5 %	49,5 %	41,6 %	20,5 %
BASE4	2 969	17,2 %	44,2 %	16,0 %	26,9 %	0,0 %	25,4 %	43,4 %	35,0 %	30,4 %	0,0 %
BASE5	2 179	41,7 %	44,5 %	43,8 %	31,4 %	15,1 %	42,4 %	41,0 %	39,4 %	34,6 %	9,4 %
BASE6	2 004	43,6 %	50,6 %	42,9 %	33,1 %	1,7 %	44,1 %	46,6 %	41,9 %	39,2 %	1,0 %
BASE7	10 000	43,9 %	48,0 %	40,7 %	43,2 %	40,7 %	45,4 %	48,0 %	44,3 %	41,8 %	40,5 %
MOYENNE	2 994,5	34,9 %	40,6 %	34,9 %	31,7 %	16,7 %	38,0 %	43,1 %	38,6 %	35,5 %	16,6 %

Tableau 1. *Taux d'erreurs obtenus sur 11 bases avec 5, 10, 15 et 20 % de bruit*

Dans le tableau 1, nous présentons les résultats que nous obtenons sur ces bases avec RPNI, RPNI* et notre approche boostée (BOOST après 200 itérations). Cette étude expérimentale a été effectuée pour 5, 10, 15 et 20 % de bruit, et en utilisant un graphe des kPPV construit dans un espace euclidien à deux dimensions, tel que décrit en section 4.2. Le paramètre k a été déterminé en testant plusieurs valeurs comprises dans $[0, 15]$. Le paramètre β (introduit en section 4.1), quant à lui, n'a pas fait l'objet d'expérimentations spécifiques durant cette étude. Il a été fixé à $0,5$ durant tout le protocole expérimental, de manière à garantir des valeurs de confiance faibles pour les exemples suspicieux, même si par ce choix nous courons le risque de perdre des infor-

mations apportées par l'étiquette de certains exemples de marge faiblement positive. En plus des performances de RPNI* et du boosting (colonne BOOST), deux colonnes supplémentaires ont été ajoutées au tableau. Nous avons voulu d'abord par ces expérimentations analyser l'écart entre les colonnes BOOST et RPNI*. Plus précisément, nous désirons connaître, dans cet écart, la part due à l'oracle lui-même, et celle issue directement de notre processus de boosting. Pour atteindre cet objectif, nous avons décidé de retourner l'étiquette des exemples d'apprentissage auxquels le graphe de voisinage attribue une marge négative. Nous avons ensuite exécuté RPNI* sur ce jeu de données (colonne SHIFT). Dans l'autre colonne ajoutée (PERF), nous présentons les résultats obtenus avec un oracle parfait, *i.e.* un oracle connaissant exactement les exemples non bruités. L'objectif de cette colonne est d'établir le taux d'erreur optimal (mais inaccessible en pratique sans accès à ce niveau d'expertise) de notre méthode pour chacune des bases.

Les remarques suivantes peuvent être faites. Tout d'abord, l'utilisation d'un graphe de voisinage pour simuler un oracle est tout à fait justifiée aux vues des résultats obtenus. Alors que RPNI* améliore déjà significativement RPNI (32,5 % contre 40,3 % en moyenne sur tous les niveaux de bruit), notre modèle boosté est encore plus performant : l'utilisation d'un oracle permet de réduire significativement le taux d'erreur de RPNI* (29,3 % contre 32,5 % en moyenne), comme un test de Student apparié permet de le vérifier.

La colonne SHIFT permet de faire un constat intéressant relatif à l'apport de l'oracle seul. Nos résultats montrent que celui-ci n'est pas toujours très efficace pour améliorer, à lui tout seul, les performances de RPNI*, et que seul un couplage avec notre processus de boosting est presque toujours gagnant. Il s'avère même parfois que les résultats se trouvent détériorés. Une explication de ce phénomène tient à la méthode utilisée pour réétiqueter les exemples : en effet, nous avons fait le choix de retourner l'étiquette de tous les exemples de marge négative, même si parmi eux, certains ont une marge légèrement négative. Pour ces exemples, l'oracle émet un faible doute, ce qui n'implique pas de manière catégorique que ces exemples soient des exemples bruités. Retourner la classe de ces exemples induit donc un risque élevé de bruiter un exemple qui ne l'était pas, et donc d'ajouter du bruit dans l'échantillon d'apprentissage (ce qui n'est pas le cas pour les exemples de marge fortement négative, pour lesquels la suspicion de bruit est forte), et de ce fait de dégrader les performances de RPNI* lorsque celui-ci apprend sur cet échantillon. En conséquence, nous déduisons de cette colonne SHIFT que la qualité des résultats observés dans la colonne BOOST est essentiellement liée au schéma même de repondération du boosting, qui utilise l'oracle, plutôt qu'aux seules performances de cet oracle. Enfin, il est à noter que l'ensemble des comportements observés précédemment restent relativement constants lorsque le niveau de bruit augmente. Pour montrer la convergence exponentielle du taux d'erreur en généralisation au cours des itérations, la figure 6 présente le comportement de l'algorithme sur quatre bases caractéristiques (pour 5 % de bruit). Elle met en évidence un effet rapide du boosting sur les performances (souvent avant la vingtième itération).

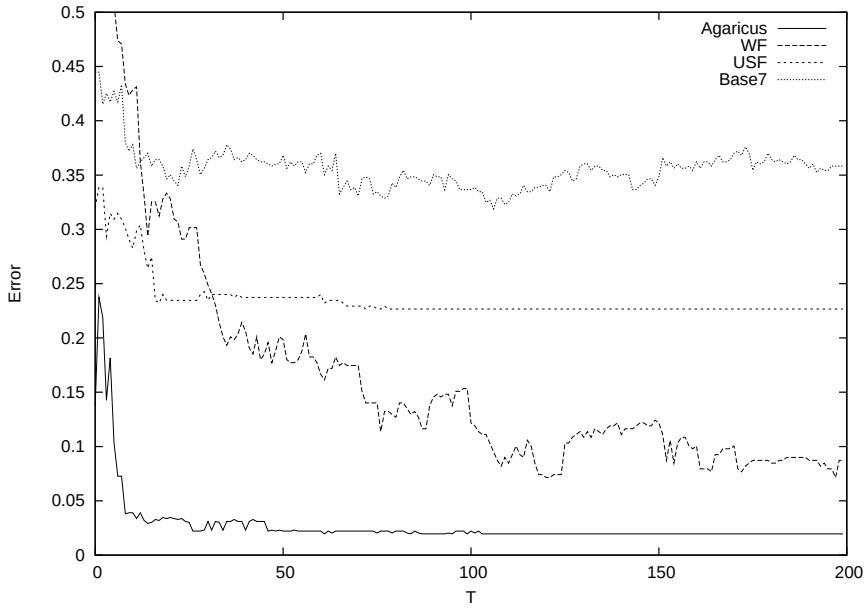

Figure 6. *L'erreur en généralisation en fonction des itérations*

6. Conclusion

Nous avons introduit dans cet article une nouvelle stratégie de boosting permettant d'améliorer significativement les performances d'un algorithme d'inférence grammaticale en présence de données bruitées. Notre approche est originale, dans la mesure où elle repose sur l'utilisation d'un oracle capable d'estimer une valeur de confiance sur l'étiquette des exemples d'apprentissage. Nous pensons qu'elle est intéressante pour optimiser *tout* algorithme d'apprentissage, bien que ceci nécessite une étude expérimentale plus poussée. Dans le cadre de l'inférence grammaticale, nous avons proposé une heuristique, basée sur un graphe des kPPV, pour simuler efficacement cet oracle. Mais de même, l'utilisation d'autres oracles, basés sur des modèles de Markov cachés par exemple, mériterait de plus amples investigations.

7. Bibliographie

Blum A., Mitchell T., « Combining Labeled and Unlabeled Data with co-training », *Proc. of the 11 th International Conference on Computational Learning Theory*, p. 92-100, 1998.

Carrasco R., Oncina J., « Learning stochastic regular grammars by means of a state merging method », *Proc. 2nd International Colloquium on Grammatical Inference - ICGI '94*, vol. 862, Springer-Verlag, p. 139-150, 1994.

Coste F., State Merging Inference of Finite State Classifiers, Technical report, Publication interne n°1250, 1999.

Cover T., Hart P., « Nearest neighbor pattern classification », *IEEE Trans. Inform. Theory*, vol. IT13, p. 21-27, 1967.

de la Higuera C., « A bibliographic survey on grammatical inference », *Pattern Recognition*, 2004. Accepted.

Domingo C., Watanabe O., « Madaboost : a modification of Adaboost », *in* , A. Press (ed.), *Third Annual Conference on Computational Learning Theory*, p. 180-189, 2000.

Freund Y., « An adaptive version of the boost by majority algorithm », *Machine Learning*, vol. 43, n° 3, p. 293-318, 2001.

Freund Y., Schapire R. E., « Experiments with a New Boosting Algorithms », *Proc. of the 13 th International Conference on Machine Learning*, p. 148-156, 1996.

Freund Y., Schapire R. E., « A Decision-Theoretic generalization of on-line learning and an application to Boosting », *Journal of Computer and System Sciences*, vol. 55, p. 119-139, 1997.

Friedman J., Hastie T., Tibshirani R., Additive logistic regression : a statistical view of boosting, Technical report, 1998.

Kearns M. J., Vazirani U. V., *An Introduction to Computational Learning Theory*, M.I.T. Press, 1994.

Kivinen J., Warmuth M., « Boosting as entropy projection », *Proc. of the 12 th International Conference on Computational Learning Theory*, p. 134-144, 1999.

Lang K., Pearlmutter B., Price R., « Results of the Abbadingo One DFA Learning Competition », *4th Int. Coll. on Grammatical Inference*, p. 1-12, 1998.

Oncina J., García P., *Inferring Regular Languages in Polynomial Update Time*, vol. 1 of *Machine Perception and Artificial Intelligence*, World Scientific, p. 49-61, 1992.

Schapire R. E., Singer Y., « Improved Boosting Algorithms using Confidence-rated Predictions », *Machine Learning*, vol. 37, p. 297-336, 1999.

Sebban M., Janodet J., « On state merging in grammatical : a statistical approach for dealing with noisy data », *Proc. of the 20 th International Conference on Machine Learning*, p. 688-695, 2003. See also CAp'03.

Thollard F., Dupont P., de la Higuera C., « Probabilistic DFA Inference using Kullback-Leibler Divergence and Minimality », *in* , P. Langley (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning*, San Francisco, June, 2000.