Comparaison du stump et du Perceptron dans les méta-algorithmes d'apprentissage

Eric Buist
Département d'informatique et de recherche opérationnelle
Université de Montréal
CP 6128 succ. Centre-Ville, Montréal
buisteri@IRO.UMontreal.CA

Résumé

Il existe plusieurs algorithmes d'apprentissage pour la classification qui combinent un ensemble de classifieurs faibles afin d'obtenir de meilleures prédictions. Les performances de tels algorithmes dépendent de la technique utilisée pour effectuer les combinaisons ainsi que du classifieur faible mis en œuvre. Souvent, le decision stump est utilisé, car il est simple à calculer et à tester. Nous allons tenter d'étudier l'impact du remplacement de ce decision stump par un algorithme plus puissant, le Perceptron. Nous allons étudier le comportement sur l'ADABOOST ainsi que sur l'arbre de décision et nous allons comparer avec l'arbre de décision alternant.

1 Introduction

Plusieurs algorithmes d'apprentissage de classifieurs combinent plusieurs classifieurs faibles afin d'améliorer la prédiction. L'ADABOOST combine plusieurs classifieurs faibles par un vote pondéré, chaque itération ajoutant un nouveau classifieur faible à la somme pondérée formant la prédiction. L'arbre de décision utilise un algorithme d'apprentissage afin de déterminer la meilleure condition de coupe pour séparer les feuilles. À chaque itération, une feuille est choisie et scindée en deux feuilles distinctes. L'arbre de décision alternant constitue un algorithme combinant ADABOOST avec l'arbre de décision classique afin d'améliorer sa stabilité.

Les performances de ces algorithmes dépendent en grande partie du classifieur faible utilisé pour apprendre les différentes hypothèses ou conditions de base. Souvent, le decision stump est mis en œuvre, car il est simple à calculer, non itératif et très efficace à tester. Pour le test, il ne suffit en effet que de déterminer si un attribut de l'exemple testé est plus grand qu'un seuil prédéterminé lors de l'entraînement. Le decision stump n'autorise comme solution que des hyperplans orthogonaux aux axes. Le Perceptron, quant à lui, permet de séparer les données avec n'importe quel hyperplan. Toutefois, son apprentissage est itératif et il est assez instable lorsque les données apprises ne sont pas linéairement séparables. Puisque la solution du Perceptron est plus générale que le stump, il est possible qu'il puisse améliorer la prédiction obtenue avec des méta-algorithmes d'apprentissage. En l'utilisant comme classifieur de base ou comme décision dans un arbre de décision, il est

peut-être possible d'améliorer la performance, la stabilité et la résistance au bruit du méta-algorithme. Nous allons tenter de vérifier cette hypothèse en comparant les algorithmes sur des ensembles de données issus de domaines réels. La section 2 introduit les différents algorithmes que nous allons tester. La section 3 expose les classifieurs faibles utilisés dans les méta-algorithmes. La section 4 traite des expérimentations numériques effectuées et discute les résultats obtenus.

2 Méta-algorithmes utilisés

Soit $\mathbf{x}_i \in \mathbb{R}^d$, où $i = 1, \ldots, n$, un exemple d'apprentissage et soit $y_i \in \{-1, 1\}$ la classe associée à cet exemple. Les méta-algorithmes utilisés dans ce projet sont itératifs. Après une phase d'initialisation, il est possible de leur appliquer un certain nombre d'itérations. Parfois, ce nombre d'itérations est limité par la capacité d'apprentissage sur les données, mais il n'est jamais connu ni borné d'avance.

2.1 AdaBoost

L'algorithme AdaBoost [1] repose sur le vote pondéré de plusieurs classifieurs faibles. Une pondération initiale de $w_i^{(1)} = 1/n$, où n est le nombre d'exemples, est associée à chaque exemple d'apprentissage et, à chaque itération, l'algorithme tente de trouver une hypothèse de base $h^{(t)}$ minimisant l'erreur de classification pondérée

$$\varepsilon^{(t)} = \sum_{i:h^{(t)}(\mathbf{x}_i) \neq y_i} w_i^{(t)}.$$

Une pondération est ensuite associée à cette hypothèse de base.

$$\alpha^{(t)} = \frac{1}{2} \log \left(\frac{1 - \varepsilon^{(t)}}{\varepsilon^{(t)}} \right).$$

Les poids sont ensuite mis à jour pour la prochaine itération.

$$w_i^{(t+1)} = \begin{cases} \dfrac{w_i^{(t)}}{2(1-\varepsilon^{(t)})} & \text{si } h^{(t)}(\mathbf{x}_i) = y_i, \\ \dfrac{w_i^{(t)}}{2\varepsilon^{(t)}} & \text{si } h^{(t)}(\mathbf{x}_i) \neq y_i. \end{cases}$$

Ainsi, les points bien classifiés reçoivent de plus faibles pondérations que des points mal classifiés.

Pour classifier un exemple, l'algorithme calcule la prédiction pour ce dernier.

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\mathbf{x}).$$

Si cette prédiction est positive, la classe est $+1$. Si tel n'est pas le cas, la classe est -1. Si $f(\mathbf{x}) = 0$, la classe de $\mathbf{x}$ est indéfinie.

Cet algorithme peut s'arrêter pour plusieurs raisons. Il est possible qu'un classifieur faible retourne $\varepsilon^{(t)} = 0$ pour une certaine pondération des exemples. Dans ce cas, un classifieur parfait a été trouvé et sera utilisé en lieu et place du vote pondéré de plusieurs classifieurs. La valeur $\alpha^{(t)}$ pour un tel classifieur étant ∞, les classifieurs précédents ont une importance nulle par rapport à ce dernier. Il est aussi possible que $\varepsilon^{(t)} \geq 0.5$ pour un classifieur faible donné. Dans ce cas, les propriétés de convergence de l'AdaBoost ne tiennent plus et l'algorithme ne tient pas compte de ce dernier classifieur. Lorsqu'une de ces deux conditions

est rencontrée, l'algorithme ne peut plus apprendre davantage sur l'ensemble de données, car les poids ne peuvent pas être mis à jour. Il existe également un cas où l'algorithme oscille entre plusieurs distributions de poids sans jamais parvenir à diminuer son erreur d'entraînement, mais ce dernier n'est pas traité par notre implantation.

2.2 Arbre de décision

La construction d'un arbre de décision consiste à associer une condition à chaque nœud interne d'un arbre et à associer une étiquette à chaque feuille. Lors de la classification d'un exemple, l'algorithme parcourt un chemin à l'intérieur de l'arbre, testant l'exemple dans chaque nœud interne afin de savoir vers quelle branche l'envoyer. Lorsqu'une feuille est atteinte, la classe de l'exemple est définie comme la classe de cette feuille. Il existe un grand nombre de méthodes pour la construction d'arbres de décision et nous avons choisi d'utiliser CART [2, chapitre 5]. Initialement, l'algorithme ne contient qu'une feuille constituant la racine de l'arbre. L'erreur de classification initiale constitue le nombre $\min(n_+, n_-)$, où n_+ constitue le nombre d'exemples de classe $+1$ et n_- le nombre d'exemples classés -1. Nous avons bien entendu $n = n_+ + n_-$.

À chaque itération, chaque feuille de l'arbre est testée afin de trouver laquelle sera scindée. L'algorithme d'apprentissage faible est appelé et tente de trouver un classifieur minimisant l'erreur de classification non normalisée sur les exemples dans la feuille. Cette erreur est comparée à l'erreur associée à la feuille et la feuille qui réduit le plus l'erreur est choisie pour être scindée. Lors de la scission, le classifieur est utilisé pour séparer les exemples dans deux feuilles distinctes et la feuille originale devient un nœud interne. Ainsi, le nombre de feuilles dans l'arbre correspond toujours au nombre d'itérations plus un. L'algorithme s'arrête lorsque toutes les feuilles de l'arbre de décision sont pures ou lorsqu'aucune scission ne permet plus de diminuer l'erreur de classification. Il n'est alors plus possible de scinder les feuilles afin de minimiser l'erreur de classification.

2.3 Arbre de décision alternant

L'arbre de décision alternant [3] consiste en un hybride unissant l'arbre de décision et l'ADABOOST. Dans notre implantation, chaque nœud est défini comme une prédiction et un ensemble de décisions. Une décision consiste en un test déterminant un nœud de destination. Comme avec l'arbre de décision, l'algorithme est initialisé avec un seul nœud, la racine, qui ne contient aucune décision. Les poids de tous les exemples sont initialisés à $1/n$ comme dans ADABOOST et la prédiction associée à la racine est donnée par

$$a = 0.5 \log\left(\frac{w_+}{w_-}\right)$$

où w_+ constitue la somme pondérées des exemples de classe $+1$ et w_- l'analogue pour les exemples de classe -1. Lors de chaque itération, contrairement à l'arbre de décision, tous les nœuds sont testés. Pour chaque nœud, l'algorithme essaie de trouver une décision en minimisant l'erreur

$$Z_t = 2(\sqrt{w_+^{(l)} w_-^{(l)}} + \sqrt{w_+^{(r)} w_-^{(r)}}) + w_0$$

où $w_+^{(l)}$ $[w_-^{(l)}]$ constitue la somme pondérée des exemples de classe $+1$ $[-1]$ placés à gauche par la décision et $w_+^{(r)}$ et $w_-^{(r)}$ les sommes pondérées pour les éléments classés à droite. La valeur w_0 représente la somme des éléments qui ne font pas partie du nœud testé. Cette dernière valeur ne change pas d'une décision à l'autre pour un même nœud (et est nulle dans la racine), mais elle permet de comparer les erreurs entre elles, d'un nœud à l'autre.

Lorsque la feuille ou le nœud interne minimisant Z_t est trouvé, une nouvelle décision lui

est ajoutée. Le sous-arbre gauche de cette décision reçoit alors la prédiction

$$p_l = 0.5 \log \left(\frac{w_+^{(l)}}{w_-^{(l)}} \right)$$

et le sous-arbre droit reçoit la prédiction

$$p_r = 0.5 \log \left(\frac{w_+^{(r)}}{w_-^{(r)}} \right).$$

Cet algorithme consiste en fait à utiliser l'ADABOOST avec abstention pour chaque nœud de l'arbre. Comme ce dernier algorithme, l'arbre de décision alternant souffre des problèmes numériques associés. La prédiction donnée peut être infinie si $w_- = 0$. Pour contourner ce problème, des constantes $\delta = 0.1$ sont ajoutées aux poids. Ainsi, la prédicton devient $(w_+ + \delta)/(w_- + \delta)$. Dans le calcul des erreurs, la constante δ est ajoutée à w_+ et w_- afin d'éviter qu'un poids nul n'absorbe toute la valeur d'erreur. Ce second ajustement a été inspiré de l'implantation des arbres alternants de la librairie Weka [4].

Pour réaliser la classification, l'algorithme doit sommer les prédictions rencontrées sur tous les chemins dans l'arbre. Soit $f(N, \mathbf{x})$ la valeur de la prédiction pour le nœud N et l'exemple $\mathbf{x}$. Si $p(N)$ est la prédiction du nœud N, nous avons

$$f(N, \mathbf{x}) = p(N) + \sum_{d \in D(N)} f(C(d, N, \mathbf{x}), \mathbf{x})$$

où $D(N)$ constitue l'ensemble des décisions dans N et $C(d, N, \mathbf{x})$ constitue le nœud auquel la décision d envoie $\mathbf{x}$. Comme dans ADABOOST, une prédiction positive est associée à une classe $+1$ et une prédiction négative, à une classe -1.

Cet algorithme peut s'arrêter s'il n'est plus possible, pour quelque nœud que ce soit, d'obtenir une décision capable de minimiser l'erreur pondérée Z_t. Ceci est rare, car même si les feuilles sont pures, les nœuds internes ne le sont jamais.

3 Classifieurs faibles utilisés

Les trois algorithmes introduits précédemment nécessitent un algorithme sous-jacent appelé classifieur faible. Bien qu'il y ait une distinction entre le classifieur faible utilisé dans ADABOOST et une décision faible dans l'arbre de décision, pour l'implantation, nous avons uni les deux afin de permettre la réutilisation du même code. Une décision permet de tester un exemple $\mathbf{x}$ et lui associe une moitié de $\mathbb{R}^d$. Le classifieur associe une étiquette à chacune des moitiés définies par la décision.

3.1 Decision stump

Le decision stump constitue le classifieur de base le plus utilisé avec ADABOOST et dans l'arbre de décision. Son évaluation exige le traitement exhaustif de tous les stumps. Pour n points en d dimensions, il existe nd stumps non triviaux et un stump trivial. Le stump trivial consiste en une décision constante qui n'a aucun intérêt pour un arbre de décision. En ADABOOST, elle est utilisée pour introduire un biais. Afin d'améliorer la performance de l'évaluation, les exemples sont triés dans chaque dimension et l'erreur du stump et mise à jour de façon incrémentale lors du parcours dans chaque dimension.

L'évaluation du stump s'avère très simple, car il suffit de comparer l'attribut désigné par ce dernier avec un seuil lui aussi calculé lors de l'entraînement. Le stump définit un hyperplan orthogonal à un axe et puisque le nombre de stumps est fini, il est possible de les tester tous et ainsi de trouver celui qui minimise n'importe quelle fonction d'erreur. L'apprentissage n'étant pas itératif, le stump retournera toujours le classifieur d'erreur d'entraînement minimale.

3.2 Perceptron

Le Perceptron consiste à classer les points en utilisant un hyperplan $\mathbf{a} \cdot \mathbf{x} + a_0$ où $\mathbf{a} \in \mathbb{R}^d$ et $a_0 \in \mathbb{R}$. Le classifieur est initialisé avec $\mathbf{a} = 0$ et $a_0 = 0$. Une itération de Perceptron en ligne parcourt tous les exemples d'apprentissage et effectue n pas. Un pas consiste à déterminer la classe associée à l'exemple et, en cas d'erreur de classification, corriger le classifieur. La classe est déterminée par $f(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x} + a_0$. Si $f(\mathbf{x}) < 0$, le point est classé -1. Si $f(\mathbf{x}) > 0$, il est classé $+1$. Si le point est mal classifié, la valeur de y_i est ajoutée à a_0 et $\mathbf{x}_i y_i$ est ajouté à $\mathbf{a}$.

Malheureusement, cet algorithme est itératif et ne retourne un classifieur d'erreur d'entraînement minimale que si les données sont linéairement séparables. Il a donc fallu ajouter des heuristiques afin de faire arrêter l'algorithme et lors des arrêts, l'erreurs n'était pas toujours minimale. Puisque l'erreur oscille, il faut, au cours du processus, conserver les valeurs $\mathbf{a}$ et a_0 optimales. Si, au cours d'une itération, une erreur inférieure à l'erreur minimale précédente n'a pas été trouvé, l'algorithme s'arrête à moins que ce ne soit la première itération. L'erreur minimale initiale, avant toute itération, est définie comme l'erreur du classifieur constant. Ainsi, aucun Perceptron avec une erreur supérieure à l'erreur du classifieur constant ne sera retourné.

Il est plus difficile d'adapter le Perceptron pour l'ADABOOST, car il ne peut pas minimiser n'importe quelle fonction d'erreur. La fonction d'erreur pondérée n'étant pas lisse, elle ne peut pas être utilisée pour effectuer de la descente de gradient. Pour l'ADABOOST standard, le problème a été résolu en considérant les poids comme une loi de probabilité et en tirant n exemples avec remplacement de cette dernière. Les n exemples tirés sont ensuite donnés au Perceptron qui peut trouver l'hyperplan. Une seule itération de ce processus de ré-échantillonnage étant effectuée, il pourrait introduire de la variance dans la solution trouvée. Dans le cas de l'arbre de décision alternant, nous n'avons pas pu trouver une façon d'adapter le Perceptron afin de minimiser la fonction Z_t.

4 Expérimentations

Les expérimentations visent d'abord à vérifier l'hypothèse nulle selon laquelle un algorithme retourne la même erreur de test qu'un autre. Pour ce faire, nous allons utiliser le test 5x2cv [5]. Le test consiste à appliquer 5 itérations de la validation croisée en 2 blocs. Lors d'une itération, les exemples sont permutés aléatoirement puis une moitié de ces exemples est appelée S_1 et l'autre, S_2. Les deux algorithmes à comparer sont entraînés sur S_1 et testés sur S_2, puis vice versa. Ceci donne les quatre mesures d'erreur $P_A^{(1)}$, $P_B^{(1)}$, $P_A^{(2)}$ et $P_B^{(2)}$. Les deux premières correspondent à l'entraînement sur S_1 tandis que les deux autres correspondent à l'entraînement sur S_2. L'erreur de classification normalisée est utilisée pour calculer ces valeurs. Pour $k = 1,2$, nous avons ensuite $P^{(k)} = P_A^{(k)} - P_B^{(k)}$. Les valeurs $\bar{p} = (P^{(1)} + P^{(2)})/2$ et $s^2 = (P^{(1)} - \bar{p})^2 + (P^{(2)} - \bar{p})^2$ sont calculées pour chaque itération. Si l'erreur de test est statistiquement identique pour les deux algorithmes comparés, la valeur

$$\tilde{t} = \frac{P_1^{(1)}}{\sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}},$$

avec $P_1^{(1)}$ la valeur de $P^{(1)}$ dans la première itération et s_i^2 la valeur de s^2 lors de l'itération i, suit une distribution Student-t avec 5 degrés de liberté. Pour le test, nous évaluerons également la p-valeur de $\tilde{t}$ qui correspond à $P(t \geq \tilde{t})$. Si cette p-valeur est trop petite ($<$ 0.05), l'hypothèse nulle sera rejetée et nous pourrons considérer les algorithmes différents.

Comme il y a plus de deux algorithmes à comparer, le test 5x2cv est plus compliqué à appli-

quer. Nous allons comparer la version stump avec la version Perceptron de l'ADABOOST et de l'arbre de décision puis comparer deux à deux les versions stumps et Perceptron entre elles. Les ensembles de données pour les tests proviennent du UCI Machine Learning Repository [6]. Les premiers tests ont été effectués sur l'ensemble Contraceptive Method Choice (CMC) qui comprend environ 1400 exemples. Comme le problème était multi-classe (aucune méthode, court terme, long terme), nous l'avons ramené en binaire en fusionnant les classes « court terme » et « long terme. » Cet ensemble ne comprend que des attributs discrets. Le second ensemble, que nous appellerons ECHO, consiste en un ensemble d'echocardiogrammes permettant de prédire un décès par crise cardiaque d'un patient. Les exemples avec attributs manquants ont été supprimés et il en est resté $61/132$. Le troisième ensemble, ION, contient des relevés de radar dans l'ionosphère. Une instance de classe $+1$ représente un relevé positif ayant détecté une structure dans l'ionosphère. Dans le cas d'un exemple classé -1, les signaux radar ont passé à travers l'ionosphère. Étant donné que les données proviennent d'un instrument de mesure, elles sont plus bruitées que les données du domaine CMC. Notre implantation Java des algorithmes testés ainsi que du test 5x2cv est accessible à l'adresse `http://www.iro.umontreal.ca/~buisteri/ift6390.zip`.

Le tableau 1 montre les valeurs de $\tilde{t}$ et les p-valeurs pour différentes paires d'algorithmes. Nous observons que les p-valeurs sont plus petites lors de la comparaison de la version stump avec la version Perceptron d'un même algorithme. Dans le cas de l'arbre de décision sur le domaine CMC, l'hypothèse nulle est même rejetée. Pour la comparaison de l'ADABOOST avec l'arbre de décision et l'arbre de décision avec l'arbre de décision alternant, le test ne permet pas de tirer des conclusions. Dans le domaine ION, l'utilisation du Perceptron a un impact significatif dans l'ADABOOST plutôt que dans l'arbre de décision. Avec ce même domaine, l'ADABOOST diffère de façon significative de l'arbre de décision et de l'arbre de décision alternant, mais le test ne permet pas de conclure pour comparer les deux arbres.

Tableau 1 – Valeurs de $\tilde{t}$ avec p-valeurs

	CMC		ECHO		ION	
	$\tilde{t}$	p-val	$\tilde{t}$	p-val	$\tilde{t}$	p-val
ADABOOST avec stumps (A) et avec Perceptron (B)	0.6895	0.2606	-4.0825	0.9952	1.7104	0.0739
Arbre de décision avec stumps (A) et avec Perceptron (B)	2.0254	0.0493	-1.8605	0.9391	-0.811	0.7729
ADABOOST (A) et arbre de décision (B) avec stumps	-3.6696	0.9928	0	0.5	2.6124	0.0238
ADABOOST (A) et arbre de décision alternant (B) avec stumps	-0.2979	0.6111	0	0.5	1.6941	0.0755
ADABOOST (A) et arbre de décision (B) avec Perceptron	0.1837	0.4307	0.8048	0.2287	-2.767	0.9802
Arbre de décision (A) et arbre de décision alternant (B) avec stumps	-0.0861	0.5326	0	0.5	0	0.5

En effectuant la validation croisée pour le *t*-test, nous avons obtenu dix échantillons du nombre d'itérations, de l'erreur d'entraînement et de l'erreur de validation optimales, ainsi que de l'erreur de ré-entraînement et l'erreur de test associées. Afin de synthétiser tous ces résultats, nous avons tenté de calculer les moyennes et les écarts-types et avons retenu les nombres d'itérations optimaux ainsi que les erreurs de test pour chaque domaine et chaque algorithme testé. Le tableau 2 présente ces moyennes et écarts-types. Dans le cas de l'ensemble ECHO, la comparaison est difficile, car les algorithmes sur-apprennent rapidement. L'ensemble de données semble trop petit pour obtenir des résultats intéressants et le choix d'un algorithme plutôt qu'un autre n'a que peu d'importance. Pour les deux autres ensembles, en moyenne, l'arbre de décision a besoin de moins d'itérations pour atteindre son erreur optimale que les méthodes de boosting. Ceci s'explique par le fait que l'algorithme ne peut plus apprendre lorsque toutes les feuilles de l'arbre sont pures. Lorsque l'erreur d'entraînement est nulle, l'erreur de test ne peut plus diminuer. Avec ADABOOST, l'erreur de test continue à diminuer même lorsque l'erreur d'entraînement est nulle. Il est presque toujours possible d'ajouter un classifieur faible dans la prédiction. Avec l'arbre de décision alternant, si les écart-types sont pris en compte, le nombre d'itérations est semblable à celui de l'ADABOOST.

Tableau 2 – Valeurs des erreurs moyennes

	CMC				ECHO				ION			
	T^*		E		T^*		E		T^*		E	
	$\bar{T}^*$	σ_{T^*}	$\bar{E}$	σ_E	$\bar{T}^*$	σ_{T^*}	$\bar{E}$	σ_E	$\bar{T}^*$	σ_{T^*}	$\bar{E}$	σ_E
ADABOOST avec stumps	34.3	10.81	0.2946	0.0055	1.2	0.20	0.0933	0.0109	38.6	18.11	0.1469	0.0117
ADABOOST avec Perceptron	39.5	12.91	0.3023	0.0066	1.0	0.00	0.16	0.0371	19.4	7.37	0.1377	0.0102
Arbre de décision avec stumps	6.1	1.83	0.3341	0.0068	1.0	0.00	0.1	0.0149	2.9	0.50	0.1171	0.0107
Arbre de décision avec Perceptron	2.5	0.40	0.3052	0.006	1.0	0.00	0.18	0.0249	1.9	0.31	0.1451	0.0111
Arbre de décision alternant avec stumps	60.8	22.44	0.2917	0.0052	1.7	0.70	0.1733	0.046	13.3	4.03	0.1314	0.0092

Dans le cas de CMC, l'erreur de test est plus petite avec l'arbre de décision Perceptron qu'avec l'arbre stump. L'erreur de test est un peu plus élevée pour l'ADABOOST avec Perceptron que l'ADABOOST avec stumps. Le relation inverse est observée pour le domaine ION. Il semble plus optimal d'utiliser, dans ce cas, le Perceptron avec l'ADABOOST et le stump avec l'arbre de décision. Cette relation est déterminée par la capacité du Perceptron à bien classifier les données. Parfois, l'erreur minimale retournée par le Perceptron se trouve supérieure au stump optimal pour un même ensemble de données. Il se peut que plus les données sont linéairement séparables, plus le Perceptron surpasse le stump. La taille de l'ensemble d'apprentissage influençant sa séparabilité, cela semble expliquer pourquoi le Perceptron surpasse le stump sur l'ensemble ION et non sur CMC. D'un autre côté, si les données étaient linéairement séparables ou presque linéairement séparables, l'arbre de décision ne contiendrait que peu de feuilles, car elles seraient rapidement pures. Ceci se confirme par le plus petit nombre d'itérations de l'arbre de décision Perceptron par rapport à l'arbre stump dans le domaine ION. Dans le cas de CMC, si nous examinons les courbes d'apprentissage (figure 1), nous constatons que le classifieur cesse d'apprendre avant d'atteindre une erreur d'entraînement nulle. Avec le Perceptron, nous observons le même phénomène, mais l'erreur d'entraînement diminue plus lentement. Ainsi, le Perceptron sépare moins bien les données que le stump puisque l'erreur diminue plus lentement avec ce dernier. Dans le cas du domain ION, la courbe d'erreur atteind rapidement 0, comme le montre la figure 2. Avec le Perceptron, l'erreur d'entraînement diminue encore plus rapidement pour ce domaine. Il semble que plus l'erreur d'entraînement atteind rapidement son

pallier, plus l'erreur de test sera réduite. Le Perceptron sépare mieux les données qu'avec le stump.

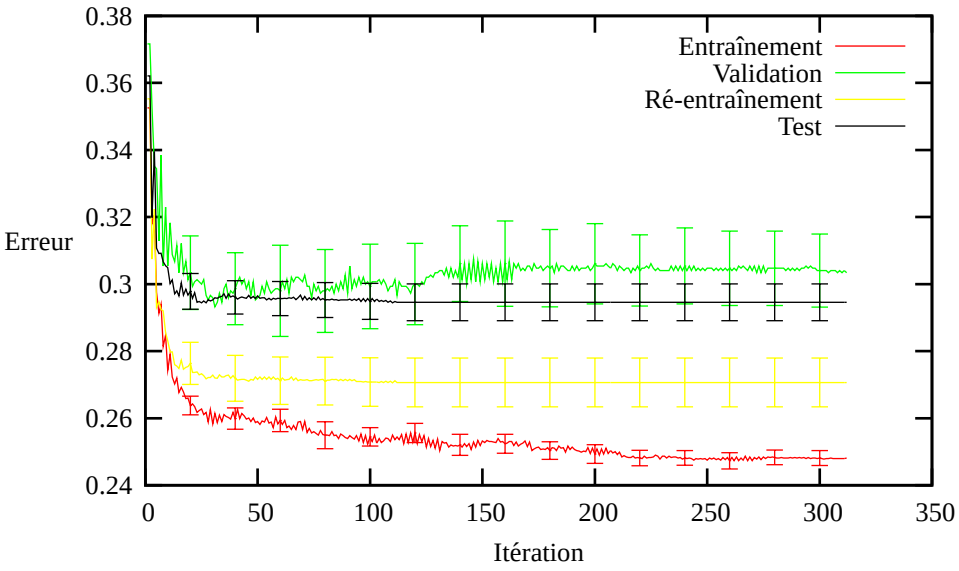

Figure 1 – Courbes d'apprentissage pour l'ADABOOST avec decision stumps, Contraceptive Method Choice

Dans le domaine CMC, ADABOOST surpasse l'arbre de décision. Toutefois, avec ION, l'arbre de décision surpasse ADABOOST. Il est possible qu'ADABOOST généralise mieux lorsque le nombre d'exemple est grand tandis qu'avec un petit domaine, l,arbre de décision s'avère meilleur. Parfois, l'arbre de décision alternant peut surpasser l'arbre de décision et l'ADABOOST classiques. Par exemple, dans le domaine CMC, l'arbre de décision alternant constitue le meilleur algorithme parmi ceux testés. Toutefois, dans le cas du domaine ION, l'arbre de décision classique avec stumps constitue le meilleur choix et l'ADABOOST fonctionne le moins bien.

5 Conclusion

Nous avons implanté et testé l'algorithme ADABOOST, l'arbre de décision CART ainsi que l'arbre de décision alternant en tentant de faire varier le classifieur faible sous-jacent. La supériorité d'un classifieur faible sur l'autre n'a pu être établie de façon indépendante du domaine. Dans le cas d'ADABOOST, un ensemble d'entraînement assez grand et un bon classifieur faible est nécessaire pour parvenir à une bonne erreur de généralisation. Dans le cas de l'arbre de décision, il semble plus approprié d'utiliser un classifieur faible moins bon afin d'éviter que les feuilles deviennent pures trop rapidement. L'arbre de décision généralise bien avec de petits domaines tandis que l'ADABOOST le surpasse lorsque le nombre d'exemples s'accroît. L'arbre de décision alternant est capable de surpasser l'ADABOOST, mais dans les petits jeux de données, l'arbre de décision classique demeure meilleur. Les différences d'erreurs sont toutefois assez petites et, surtout dans le cas d'ADABOOST, il semble plus simple d'utiliser le stump puisque plusieurs classifieurs faibles sont combinés.

Il existe plusieurs autres variantes de l'ADABOOST dont l'ADABOOST M1 pour le multi-classes [7]. Il serait intéressant de comparer ces méthodes en modifiant le classifieur faible.

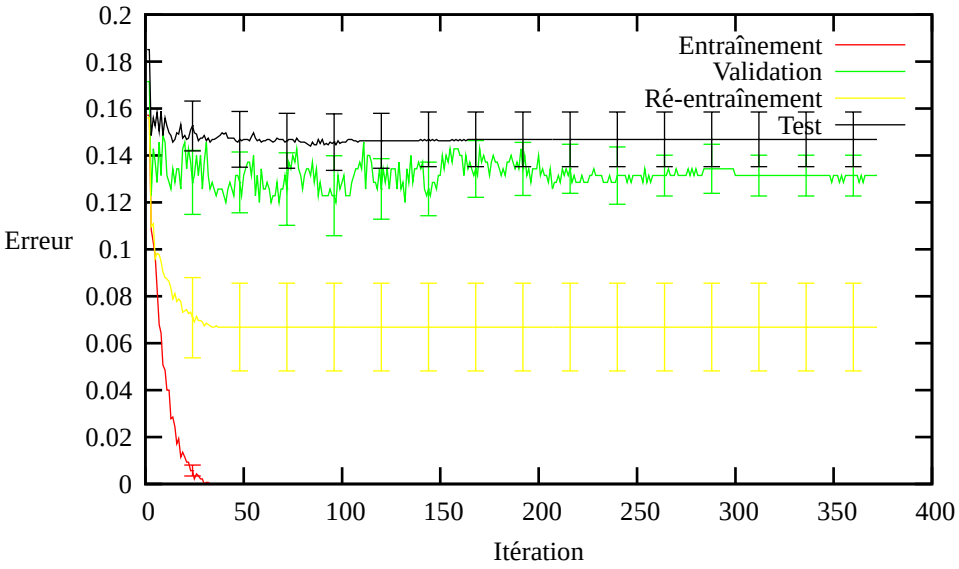

Figure 2 – Courbes d'apprentissage pour l'ADABOOST avec decision stumps, Johns Hopkins University Ionosphere database

Lors des expérimentations, nous avons constaté que le Perceptron ne retournait pas toujours l'hyperplan optimal puisque dans certains cas, le stump donnait une plus petite erreur. Il serait envisageable de remplacer le Perceptron par un autre algorithme tel qu'une machine à vecteurs de support linéaire capable de trouver l'hyperplan séparateur optimal. Bien que l'utilisation de SVMs avec ADABOOST ne revêt pas un grand intérêt en pratique puisque les noyaux permettent déjà de rendre les SVMs non linéaires, il serait intéressant d'examiner l'impact de l'optimalité du classifieur sur le méta-algorithme.

Du côté des arbres de décision, il existe un grand nombre de techniques de construction et une seule a été explorée dans ce projet. Dans l'algorithme CART, il est possible d'introduire du pruning qui permet de fusionner deux feuilles séparées préalablement. Il est aussi possible d'utiliser d'autres algorithmes de construction tels que C4.5 ou C5.0. La stabilité des arbres de décision n'a pas bien été comparée lors de ce projet. La variance des erreurs ou du nombre d'itérations ne constitue pas une bonne mesure de stabilité et il semble qu'il serait plus judicieux de définir une mesure de distance entre deux arbres et utiliser comme mesure de performance la distance moyenne entre plusieurs arbres de décision créés lors de la validation croisée en k blocs.

Dans le cas des arbres de décision alternants, il n'a pas été possible de trouver un algorithme Perceptron adapté à la fonction de coût à minimiser. Il serait peut-être possible de parvenir à un tel algorithme en dérivant une approximation lisse de la fonction de coût ou une technique de ré-échantillonnage adaptée à cet algorithme.

Références

[1] Y. Freund and R. Schapire. A short introduction to boosting. *Journal of Japaneese Society for Artificial Intelligence*, 14(5) :771–780, September 1999.

[2] Duda, Hart, and Stork. *Pattern Classification*. Wiley Inter-Science, 2001.

[3] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm,. In *Proceedings of the 16th International Conference on Machine Learning*, pages 124–133. Morgan Kaufmann, San Francisco, CA, 1999.

[4] Ian H. Witten and Eibe Frank. *Data Mining : Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, CA, 2000.

[5] Thomas G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7) :1895–1923, 1998.

[6] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. University of California, Irvine, Deptartment of Information and Computer Sciences, 1998.

[7] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1) :119–139, Augest 1997.