Chapitre I

Amélioration du *Boosting* face aux données bruitées

E. Bahri, S. Lallich, N. Nicoloyannis, M. Maddouri, A hybrid approach of boosting against noisy data, Mining Complex Data, *Studies in Computational Intelligence*, Vol. 165, Springer, Heidelberg, Germany, 2009, 41-54 (A.D. Zighed, S. Tsumoto, Z. Ras, and H. Hacid (Eds.) : Mining Complex Data).

E. Bahri, M. Maddouri, Nouvelle approche du Boosting pour les données bruitées, *8ème Conférence Extraction et Gestion des Connaissances (EGC 08)*, Sofia Antipolis, Janvier 2008, 349-360, Revue des Nouvelles Technologies de l'Information, Cépaduès, Toulouse.

E. Bahri, N. Nicoloyannis, M. Maddouri, Improving boosting by exploiting former assumptions, *3rd International Workshop on mining complex data (MCD 07), with conjonction ECML/ PKDD 07*, Warsaw, Poland, 2007.

E. Bahri, N. Nicoloyannis, M. Maddouri, Amélioration du Boosting par combinaison des hypothèses antérieures, *14èmes Rencontres de la Société Francophone de Classification (SFC 07)*, Paris, Septembre 2007.

A. Stavrianou, E. Bahri, N. Nicoloyannis, Text Mining Issues and Noise Handling in Health Care Systems, *9th International Conference on System Science in Health Care*, Lyon, France, September 2008.

Sommaire

1. Introduction

L'émergence des bases de données actuelles et leur croissance exponentielle ainsi que l'évolution des systèmes de transmission contribuent à la construction de masses de données qui dépassent de loin les capacités humaines à les traiter. Ces données sont des sources d'information et de connaissance qui nécessitent des méthodes automatiques de synthèse et d'interprétation. Les recherches se sont orientées vers des systèmes d'intelligence artificielle puissants permettant l'extraction des informations pertinentes et aidant à la prise de décisions.

C'est ainsi que le*Data Mining* offre la possibilité de construire un modèle de prédiction d'un phénomène à partir d'autres phénomènes plus facilement accessibles qui lui sont liés. A cet effet, le data mining utilise des méthodes variées qui proviennent aussi bien de l'apprentissage machine [Mitchell, 1997] que des méthodes connexionistes [Benanni, 2006], de la reconnaissance des formes [Duda et al., 2001] ou des méthodes statistiques [Saporta, 2001].

Cependant, le modèle construit peut parfois engendrer des classifieurs qui ne sont même pas des classifieurs faibles puisque leur taux d'erreur dépasse 50% (classe binaire). En fait, lors de l'apprentissage, on teste le modèle soit sur un échantillon de données soit sur des données simulées qu'on garantit représentatives des données réelles, ce qui permet au modèle d'être performant. Cependant, un modèle construit qui fonctionne bien sur des données simulées ne fonctionne pas nécessairement bien sur des données réelles, dans la mesure où celles-ci sont complexes, hétérogènes et en particulier bruitées. De ce fait, le bruit est devenu un problème majeur en apprentissage automatique.

En fait, si beaucoup d'algorithmes et de cadres théoriques ont déjà été proposés, très peu d'études portant spécifiquement sur la définition du bruit et son traitement ont été effectuées. Pourtant, face à la quantité de données corrompues dans les bases de données, il semble impératif de détecter le bruit et d'avoir recours à des techniques appropriées pour espérer appliquer avec succès des méthodes d'apprentissage supervisé sur des données réelles. Sans de telles approches, l'apprentissage automatique ne pourra pas prétendre à une utilisation efficace dans ses domaines d'application naturels que sont par exemple la bio-informatique, la reconnaissance vocale ou l'exploitation des données commerciales en sortie de caisse des grandes surfaces. Les procédures qui procèdent par agrégation de classifieurs, ou ensemblistes, ont beaucoup contribué à améliorer l'efficacité des procédures de prédiction en *data mining*, en particulier le *Boosting* qui a la particularité d'être adaptatif au sens où il est forcé à se spécialiser sur les exemples difficiles à prédire, ce qui le rend très sensible au bruit. Dans ce chapitre, nous intéressons à l'apprentissage supervisé des données réelles bruitées lorsque l'on utilise des procédures ensemblistes adaptatives telle que le *Boosting*.

Ces méthodes d'agrégation sont efficaces d'un point de vue compromis Biais-variance, mais aussi grâce aux trois raisons fondamentales expliquées par [Dietterich, 2000] :

Raison Statistique Un algorithme d'apprentissage peut être considéré comme une

recherche d'un espace H des hypothèses pour identifier la meilleure hypothèse dans l'espace. Le problème statistique surgit quand la quantité de données d'apprentissage disponibles est trop petite comparée à la taille de l'espace d'hypothèse. Sans données suffisantes, l'algorithme d'apprentissage peut trouver plusieurs hypothèses dans H donnant la même exactitude sur les données d'apprentissage. En construisant un ensemble en se basant sur tous ces classifieurs, l'algorithme peut faire la moyenne de leurs votes et réduire le risque de choisir le mauvais classifieur.

Raison Informatique Différents algorithmes d'apprentissage fonctionnent en exécutant une certaine recherche locale qui peut se coincer dans des optimums locaux. Un ensemble construit en exécutant la recherche locale de différents points de départ peut fournir une meilleure approximation de la fonction cible inconnue que n'importe lequel des différents classifieurs.

Raison de représentation Dans la plupart des applications d'Apprentissage Machine(Machine Learning), la fonction cible F ne peut être représentée par aucune des hypothèses constituant H. En formant des sommes pondérées d'hypothèses de l'ensemble H, on peut augmenter l'espace de représentation des fonctions.

Ces méthodes d'agrégation de classifieurs peuvent être classées selon deux catégories :

- celles qui fusionnent des classifieurs prédéfinis ; on trouve notamment le vote simple [Bauer and Kohavi, 1999], le vote pondéré [Bauer and Kohavi, 1999] et le vote à la majorité pondérée [Littlestone and Warmuth, 1994].
- celles qui agrègent des classifieurs induits durant l'apprentissage ; parmi celles-ci, on trouve des stratégies induisant des classifieurs en parallèle (Bagging) [Breiman, 1996] ainsi que des stratégies adaptatives comme le *Boosting* et son algorithme de base AdaBoost [Shapire, 1990], consistant à induire séquentiellement de nouveaux classifieurs.

L'étude comparative de [Dietterich, 1999] montre bien que dans le cas où les données sont faiblement bruitées, l'algorithme AdaBoost, qui est la version la plus populaire du *Boosting*, semble être immunisé contre le sur-apprentissage. Cette constatation s'est traduite par le fait que non seulement l'erreur empirique sur l'échantillon d'apprentissage décroît exponentiellement avec le nombre d'itérations, mais encore l'erreur en généralisation décroît elle aussi à mesure que de nouveaux classifieurs sont ajoutés à l'ensemble.

Cependant, cette même étude montre que, sur des données fortement bruitées, AdaBoost présente un taux d'erreur en généralisation parfois plus important que le Bagging. Une raison à ces mauvaises performances, mise en évidence par Dietterich, vient du fait que le *Boosting* tend à augmenter le poids des exemples bruités à travers les itérations, se spécialisant ainsi sur la prédiction de ces exemples. La conséquence immédiate est un sur-apprentissage malencontreux puisque AdaBoost se focalise en partie sur les exemples bruités. La vitesse de convergence du *Boosting* se trouve également pénalisée sur ce type de données.

Ce chapitre se situe dans le cadre des procédures adaptatives d'agrégation de classifieurs en

ayant pour but l'amélioration des performances de l'algorithme AdaBoost. Nous proposons une nouvelle approche modifiant le critère d'induction de nouveaux classifieurs à chaque itération. Notre stratégie consiste à prendre en compte tout l'historique de génération de modèles afin d'induire un nouveau classifieur courant, contrairement à la version originale d'AdaBoost qui tient compte seulement de la dernière itération.

Ce chapitre est organisé comme suit. Aprés avoir présenté le contexte des données bruitées en deuxième section, nous exposons notre problématique et nos motivations en troisième section. Dans la quatrième section, nous évoquons les divers concepts justifiant le recours à des techniques ensemblistes, puis nous réalisons une synthèse des principales techniques mises en place pour construire des ensembles de classifieurs. En section cinq, nous détaillons l'approche ensembliste AdaBoost et nous proposons un état de l'art synthétique des principaux travaux visant à améliorer cette méthode. Dans la sixième section, nous présentons notre amélioration du *Boosting* qui aboutit à l'algorithme AdaBoost Hybride. Ensuite, nous effectueons une large étude expérimentale visant à comparer, sur de nombreuses bases de données réelles, les performances d'AdaBoost et AdaBoost Hybride, en termes de taux d'erreur, de précision-rappel et vitesse de convergence. Enfin, nous terminons par une conclusion et des perspectives.

2. Contexte des données bruitées

2.1. Notations

Nous nous intéressons ici au cadre de la classification supervisée. Formellement, nous avons accès à un échantillon de n exemples (ou individus) étiquetés $S = (x_1, y_1)...(x_n, y_n)$ issus d'une distribution (inconnue) P sur $X \times Y$ où X est un espace à d dimensions composé des descripteurs $X_1, ..., X_d$ et Y l'espace (discret) à K dimensions des réponses (ou étiquettes, classes). L'objectif de la classification supervisée est de trouver une fonction $h : X \rightarrow Y$ (appelée classifieur, hypothèse, modèle, apprenant) telle que l'erreur en généralisation $Pr_{(x,y)\equiv P}[h(x) \neq y]$ soit la plus faible possible.

De très nombreuses techniques ont été mises au point afin de trouver une fonction h qui ait la plus faible erreur en généralisation possible. La recherche d'une telle fonction se heurte à plusieurs écueils. Ces écueils sont inhérents au fait que la vraie distribution P est inconnue. Dans le cas contraire, les éléments issus de la statistique bayésienne nous permettent de déterminer une fonction h optimale. Une première possibilité pour contourner cette difficulté est de faire des hypothèses a priori sur la nature de la vraie distribution. Par exemple, on pourra supposer que les descripteurs sont issus d'une distribution gaussienne multidimensionnelle. Une fois les hypothèses posées, il s'agit ensuite d'estimer les paramètres de la distribution pour obtenir un résultat "optimal". Une technique très connue dans ce contexte est celle des modèles de mélanges [Hastie et al., 2001].

Toutefois, dans la réalité, il est extrêmement fréquent de n'avoir aucune connaissance a priori sur la vraie distribution. Dans ce cas, il est impossible d'émettre quelque hypothèses que ce soit et il est alors nécessaire d'utiliser une technique d'apprentissage adaptée, autrement dit, ne nécessitant aucune hypothèse a priori. Les méthodes ensemblistes introduites dans la section précédente entrent pleinement dans ce contexte. Elles se sont révélées très performantes en pratique.

2.2. Le concept de bruit

Cependant, un autre écueil vient perturber les heuristiques d'apprentissage n'émettant aucun hypothèse : le bruit présent dans les données. La présence de bruit signifie globalement que l'échantillon utilisé pour l'apprentissage est corrompu. Autrement dit, l'échantillon S ne reflète pas la vraie distribution P. Dans ce contexte, il est nécessaire de bien comprendre le bruit afin de mettre au point des techniques robustes face à ce problème. Dans la littérature, plusieurs définitions du bruit (complémentaires ou non) co-existent. Ces différentes définitions peuvent être classées selon la cause du bruit :

1. **Valeurs erronées des attributs** Il est fréquent que les valeurs réelles des descripteurs ne soient pas les valeurs observées. Dans certains cas, les valeurs des descripteurs correspondent à des mesures effectuées par des appareils générant une imprécision lors de la mesure d'une variable. Par exemple, un radar mesurant la vitesse d'un véhicule n'indique jamais la valeur exacte de la vitesse mais une approximation de celle-ci.
2. **Valeurs manquantes des attributs** Dans certains cas, il est possible que certains attributs ne soient pas ou plus mesurables. Il en résulte un échantillon ayant des valeurs manquantes.
3. **Erreur d'inclusion** Il arrive que certains individus soient inclus à tort dans l'échantillon, tout particulièrement lorsque les données sont acquises de façon automatique.
4. **Erreur sur la classe** Cette situation se produit lorsque un exemple est étiqueté par une mauvaise classe, ce qui résulte généralement d'une erreur humaine.
5. **Attributs manquants** Il s'agit d'un problème majeur en apprentissage automatique. Concrètement, cela signifie qu'il manque certaines variables pertinentes pour apprendre correctement ; autrement dit, l'espace X n'est pas suffisant pour apprendre correctement Y. Dans ce cas, deux exemples de classes différentes peuvent être confondus dans l'espace X.
6. **Données redondantes** Plusieurs exemples ont une même représentation dans X. Cet aspect de bruit n'est pas considéré comme un problème sauf dans le cas où les données redondantes sont contradictoires comme dans le cas où il y a erreur sur la classe.
7. **Bruit sur les attributs et bruit sur la classe**

Il faut distinguer le bruit sur les attributs et le bruit sur la classe. Le bruit sur les attributs ne doit pas être traité lorsque le même bruit apparaît dans les exemples à classer. En effet, [Quinlan, 1986] a montré que dans ce cas, le fait de débruiter les attributs diminue la performance en généralisation lorsque le niveau de bruit augmente. En revanche, le problème du bruit sur la classe doit être traité, car celui-ci concerne exclusivement l'ensemble d'apprentissage !

Le fait d'avoir un échantillon bruité peut grandement perturber l'apprentissage. Il apparaît donc nécessaire de traiter ce dernier par des techniques adaptées. Le bruit peut généralement être traité à plusieurs niveaux du processus de modélisation. Deux possibilités (non exclusives) sont envisageables : (1) on traite le bruit en amont (pré-traitement) afin de créer un nouvel échantillon plus "propre", S', que l'on utilisera pour l'induction, et/ou (2) on adapte le processus d'induction afin de gérer correctement les éventuelles données bruitées.

Lors du pré-traitement, les problèmes des points 3, 4 et 5 peuvent être traitées via la suppression des données contradictoires. Une solution employée par [Clark and Niblett, 1989] consiste à remplacer la classe des données contradictoires par la classe la plus fréquente. Cette solution peut être également utilisée pour les attributs ayant des valeurs manquantes. Par exemple, les valeurs manquantes d'un attribut continu peuvent être remplacées par la moyenne ou la médiane de cet attribut mesurée sur les exemples ayant une valeur renseignée pour cet attribut. Dans le cas d'une variable catégorielle, on peut remplacer les valeurs manquantes par le mode (valeur la plus fréquente).

On peut également traiter le bruit directement lors de la phase d'apprentissage. Les approches dans ce domaine utilisent diverses heuristiques les rendant plus ou moins résistantes au bruit. Les principales techniques employées consistent à tolérer jusqu'à un certain point les mauvaises classifications sur l'ensemble d'apprentissage S. En effet, celles-ci apparaissent inévitables lorsque le niveau de bruit est important. Un cas typique est celui des arbres de décision [Breiman et al., 1984]. Un arbre de décision consiste à partitionner récursivement l'espace X jusqu'à obtenir des partitions les plus pures possibles (des partitions contenant des exemples ayant tous la même classe). Le gestion du bruit consiste dans ce cas précis à préférer des partitions dont les groupes sont moins purs mais de plus grande taille (i.e, contenant plus d'exemples). En effet, des groupes de plus grande taille offrent de meilleurs garanties de généralisation. Par exemple, il n'est pas aberrant de préférer un groupe de 1000 exemples pur à 95% à un groupe de 5 exemples parfaitement pur, susceptible de contenir des exemples bruités et aboutissant à un sur-apprentissage. Ainsi, diverses stratégies comme le pré-élagage de l'algorithme ID3 [Quinlan, 1994] ou le post-élagage de C4.5 [Quinlan, 1993] ont été mises en place pour éviter ce sur-apprentissage.

Il faut toutefois noter que le bruit n'est pas nécessairement la seule cause au problème de sur-apprentissage. Les "garde-fous" utilisés lors des processus d'induction de certains classifieurs comme les arbres de décision sont également utiles lorsque les données ne sont pas

bruitées. Toutefois, comme nous le verrons dans les sections suivantes, certains algorithmes (en particulier les méthodes de type *Boosting*) sur-apprennent uniquement en présence de bruit alors qu'ils sont résistants dans le cas contraire. Comme nous l'avons souligné précédemment, le *Boosting* est extrêmement efficace lorsque le niveau de bruit est faible, mais beaucoup moins performant autrement. Il nous est donc apparu intéressant de nous focaliser sur le *Boosting* afin de le rendre plus résistant au bruit. Dans la suite de ce chapitre, nous présentons plus précisément les techniques ensemblistes, et plus particulièrement les diverses stratégies de *Boosting* mises en place dans la littérature. Puis, nous introduirons notre nouvelle approche de *Boosting* résistante au bruit et nous l'éprouverons sur des données réelles.

3. Problématique et motivation

La réduction de l'erreur en généralisation est l'une des principales motivations de la recherche en apprentissage automatique, car un coût induit par une mauvaise classification peut être élevé. Ces dernières années, un grand nombre de travaux en apprentissage automatique ont porté sur les méthodes d'agrégation de classifieurs. Ces dernières méthodes consistent à induire non pas un, mais plusieurs classifieurs, puis dans une seconde phase à utiliser un mécanisme d'agrégation des classifieurs pour produire une fonction de décision h unique. Nous allons présenter les notions d'agrégation de classifieurs en décrivant par quels mécanismes cette dernière aboutit généralement à de meilleurs résultats (une erreur en généralisation plus faible) que l'utilisation d'un classifieur unique.

3.1. Evaluation de l'apprentissage

Afin de juger de la pertinence d'un classifieur, il est nécessaire d'évaluer sa performance à l'issue de la phase d'apprentissage. L'évaluation consiste à mesurer l'adéquation entre les classes réelles des exemples et les classes prédites par le modèle. Une méthode a priori simple pour évaluer la qualité du classifieur induit consiste à comptabiliser la fréquence des étiquettes bien prédites par le modèle sur l'ensemble d'apprentissage S. Si par exemple 90% des étiquettes trouvées par le modèle concordent avec les étiquettes réelles, on peut s'attendre en première instance à ce que le modèle ait une erreur de 10%. Cette technique a cependant un sérieux défaut. En effet, dans la mesure où les exemples de l'échantillon d'apprentissage ont servi à construire le modèle, le fait d'évaluer la qualité du modèle à partir de ce même échantillon aboutirait à une estimation trop optimiste du taux d'erreur réel. Autrement dit, l'erreur en généralisation $Pr_{(x,y)\equiv P}[h(x) \neq y]$ n'est pas correctement estimée par l'erreur apparente mesurée sur l'ensemble d'apprentissage.

Afin d'obtenir une estimation honnête, il est nécessaire d'évaluer le taux d'erreur sur un échantillon test, c'est à dire sur un ensemble d'exemples n'ayant pas servi à construire le modèle. L'estimation ainsi obtenue sera plus proche de la vraie erreur en

généralisation. D'autres techniques plus sophistiquées ont été mises au point afin d'obtenir encore de meilleurs estimations telles que la validation croisée [Hastie et al., 2001] ou le .636-*bootstrap* [Efron, 1979]. Dans les expérimentations de ce chapitre, la validation croisée sera utilisée comme estimateur de l'erreur réelle. Nous allons donc la décrire brièvement.

Le principal problème de l'utilisation d'un échantillon test unique afin d'estimer l'erreur réelle réside dans le fait que celui-ci est généralement de faible taille (ne contient pas beaucoup d'exemples). En effet, dans certaines applications réelles, les exemples étiquetés sont rares et donc précieux. Il est alors gênant d'utiliser une proportion importante des exemples étiquetés dans le seul but d'évaluer le modèle. La validation croisée est un procédé intéressant pour contourner cette difficulté. L'idée directrice consiste ici à subdiviser l'ensemble d'apprentissage en p sous-ensembles (généralement $p = 2$, $p = 5$ ou $p = 10$). Ensuite, on apprend un modèle sur $p - 1$ sous-ensembles, puis on utilise le sous-ensemble restant pour obtenir une première estimation du taux d'erreur en généralisation. Ce processus est ensuite répété $p - 1$ fois en utilisant comme échantillon test chacun des p sous-ensembles. A la fin du processus, nous obtenons ainsi p estimations de l'erreur en généralisation que l'on peut moyenner pour obtenir l'estimation finale de l'erreur réelle. Nous renvoyons le lecteur intéressé à [Hastie et al., 2001] pour des informations plus complètes sur la validation croisée ainsi que sur les autres estimations possibles de l'erreur réelle.

Pour une évaluation plus détaillée de la performance d'une méthode face à ses concurrentes, outre le taux d'erreur, nous avons également calculé la précision et le rappel. La précision d'une classe est la proportion d'exemples correctement étiquetés parmi les exemples étiquetés comme appartenant à la classe. Le rappel d'une classe est la proportion des exemples de la classe considérée qui ont effectivement été étiquetés de cette classe. Nous fournissons le rappel moyen et la précision moyenne des différentes classes.

Tout ceci montre qu'un bon modèle ne doit pas se limiter à minimiser l'erreur sur l'échantillon d'apprentissage afin de minimiser l'erreur en généralisation. Malheureusement, le non-connaissance de la distribution des données P et la finitude de l'échantillon d'apprentissage nous empêche de minimiser directement cette dernière. Certains travaux ont ainsi cherché à aller plus loin pour comprendre s'il est possible d'obtenir des informations complémentaires sur le comportement de l'erreur en généralisation afin des les utiliser pour l'induction. Dans ce sens, plusieurs auteurs ont observé que l'erreur en généralisation pouvait se décomposer en trois termes : le taux d'erreur incompressible, le biais et la variance. C'est principalement à partir de cette décomposition que l'on parvient à comprendre pourquoi les méthodes ensemblistes peuvent s'avérer très performantes.

3.2. La décomposition biais-variance

Plusieurs auteurs ont remarqué que l'erreur en généralisation $Pr_{(x,y)\equiv P}[h(x) \neq y]$ du classifieur $h(x)$ peut se décomposer en trois termes : l'**erreur incompressible**, le **biais** et la **variance**. Nous adoptons ci-après la décomposition de l'erreur selon [Breiman, 1998]. La fonction cible, notée $f(x)$, désigne la fonction de classification idéale, c'est à dire le classifieur atteignant le plus faible taux d'erreur possible en généralisation (on parle dans ce cas de classifieur Bayésien). Idéalement, le processus d'apprentissage aboutit à un classifieur $h(x)$ strictement équivalent à $f(x)$. Dans ce cas, il subsiste néanmoins une **erreur incompressible** $\varepsilon = Pr_{(x,y)\equiv P}[f(x) \neq y]$ due au fait que les classes ne sont pas nécessairement parfaitement séparables dans X.

Lorsqu'un classifieur $h(x)$ a été induit, on voit alors que son erreur peut se décomposer tout d'abord en deux termes : l'un représentant l'erreur incompressible, l'autre l'erreur mesurant l'éloignement entre la fonction cible et le classifieur induit :

$$Pr_{(x,y)\equiv P}[h(x) \neq y] = \varepsilon + E[h(x) - f(x)] \tag{I.1}$$

où $E[.]$ dénote l'espérance mathématique. Il est bien entendu impossible d'agir sur l'erreur incompressible. Afin de réduire au maximum l'erreur en généralisation du classifieur, il est donc nécessaire de minimiser $E[h(x) - f(x)]$. De façon intéressante, ce terme peut se décomposer lui-même en deux parties qui sont le biais et la variance. Afin de bien appréhender ces deux concepts, on peut imaginer la situation suivante. Supposons que nous ayons l échantillons $S_1, ..., S_l$ de taille n tous issus de la même distribution P. Nous utilisons le même processus d'apprentissage sur chacun des échantillons pour obtenir l classifieurs $h_1(x), ..., h_l(x)$. Chacun des classifieurs donne en sortie une prédiction $\hat{y}_1, ..., \hat{y}_l$. A partir des ces l classifieurs, nous générons un classifieur "moyen" $h^*(x)$ donnant comme prédiction $\hat{y}^*$ la classe la plus fréquemment votée par les l classifieurs. Nous pouvons alors définir le biais de notre classifieur $h(x)$ comme [Breiman, 1998] :

$$Biais[h(x)] = Pr_{(x,y)\equiv P}[h^*(x) \neq y] - Pr_{(x,y)\equiv P}[f(x) \neq y] \tag{I.2}$$

Le biais capture la notion d'erreur systématique. Un classifieur non biaisé est un classifieur qui en moyenne, donne une classification équivalente à celle du classifieur idéal. Dans une autre perspective, le biais traduit l'incapacité du modèle à apprendre correctement le concept. Par exemple, si la fonction cible est non linéaire et que le processus d'apprentissage utilisé génère une fonction linéaire, il existera nécessairement un biais dû au fait que le classifieur induit choisit sa fonction dans l'espace H des fonctions linéaires alors que la vraie fonction appartient à l'espace F des fonctions non linéaires. Les deux espaces n'étant pas identiques, il subsistera une erreur systématique.

La variance du classifieur $h(x)$ se mesure comme étant l'écart moyen entre le classifieur moyen $h^*(x)$ et le classifieur $h(x)$ induit sur l'échantillon d'apprentissage initial S [Breiman, 1998] :

$$Var[h(x)] = Pr_{(x,y)\equiv P}[h(x) \neq y] - Pr_{(x,y)\equiv P}[h^*(x) \neq y] \tag{I.3}$$

La variance mesure le fait qu'un même processus d'apprentissage fournit des résultats différents selon l'échantillon utilisé pour le générer. Ainsi, un classifieur donnant des résultats proches en faisant varier l'échantillon utilisé a une faible variance alors qu'un classifieur donnant des écarts importants a une variance forte. De la sorte, un classifieur sans biais, c'est à dire cherchant une fonction de classification dans un espace contenant la fonction cible trouvera d'autant plus facilement la fonction cible qu'il a une faible variance.

Nous arrivons ainsi à la décomposition finale de l'erreur selon [Breiman, 1998] :

$$Pr_{(x,y)\equiv P}[h(x) \neq y] = Biais[f(x)] + Var[f(x)] + \varepsilon \tag{I.4}$$

Lorsque les données sont bruitées, une nouvelle source d'erreur vient s'ajouter aux trois composantes précédemment introduites. En effet, même si le processus d'apprentissage est capable d'apprendre la fonction cible, le fait de l'entraîner sur un échantillon pathologique S aboutira à une fonction de classification induite différente de $f(x)$. Cette erreur est parfois dénommée **erreur intrinsèque**. La figure I.1 tirée de [Cornuéjols et al., 2003] illustre les différentes sources d'erreur entre la fonction cible f et la fonction hypothèse h.

3.3. Les raisons de l'efficacité de l'agrégation de classifieurs

L'utilisation d'un ensemble de classifieurs plutôt que la recherche d'un seul apparaît intéressante dans le cadre de la décomposition de l'erreur en termes de biais-variance [Dietterich, 2000]. Comme nous l'avons vu, la recherche d'un classifieur peut être vue comme la recherche d'une fonction optimale dans un espace H contenant toutes les fonctions de classification pouvant être trouvées par l'algorithme. Lorsque l'ensemble d'apprentissage est de taille modeste, il est possible qu'il existe un grand nombre de fonctions de classification dans H aboutissant à un même taux d'erreur apparent (i.e mesuré sur l'échantillon d'apprentissage). Plutôt que de sélectionner une seule de ces fonctions de classification, il peut être intéressant de toutes les prendre en compte et de les faire voter pour obtenir une prédiction finale. Suivant le théorème du jury de Condorcet (1785), le recours à une telle procédure rapproche de 0 le risque d'erreur de la décision finale, d'autant plus que le nombre de classifieurs est grand, à condition que ceux-ci soient proches de l'indépendance et que leur risque d'erreur soit inférieur à 0.5. En effet, les erreurs des différents classifieurs sont alors minoritaires par rapport aux bonnes

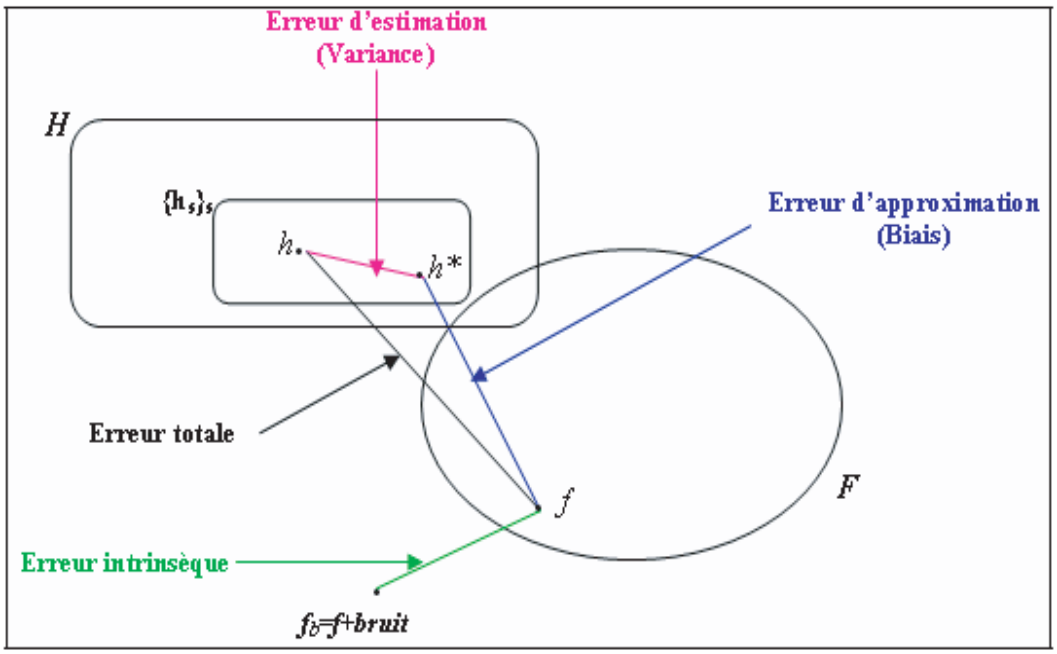

FIGURE I.1 – Les différents types d'erreur intervenant dans l'estimation d'une fonction à partir d'un échantillon d'apprentissage

décisions et sont bien réparties, le vote à la majorité assurant une bonne décision finale. En analysant la relation de cette stratégie avec le compromis biais-variance, on s'aperçoit que procéder à un vote de plusieurs classifieurs va avoir pour effet de rapprocher $h(x)$ de $h^*(x)$ et aboutira à une possible réduction de la variance du classifieur ainsi généré, et donc de l'erreur en généralisation.

Lorsque l'algorithme d'apprentissage est biaisé, il cherche une fonction de classification dans un espace H ne contenant pas la fonction de classification cible $f(x)$. En utilisant plusieurs classifieurs et en leur donnant à chacun un poids différent (vote pondéré), il devient possible de construire une fonction de classification beaucoup plus complexe que celle issue d'un algorithme d'apprentissage unique fortement biaisé. La nouvelle fonction de classification ainsi créée peut prendre la forme de la fonction cible ou du moins s'en rapprocher. Construire plusieurs classifieurs et leur donner un poids est donc un processus susceptible de diminuer le biais d'un classifieur simple.

L'utilisation d'un ensemble peut se justifier également selon le point de vue de la "recherche d'une solution optimale à un problème". En effet, certains algorithmes tels que les réseaux de neurones ou les arbres de décision résolvent le problème de l'apprentissage en utilisant des heuristiques telle que la descente du gradient. L'utilisation des heuristiques est une nécessité dans la mesure où l'espace des fonctions de classification possibles H est généralement de dimension infinie. En utilisant un ensemble, on garantit un balayage plus important de l'espace

des hypothèses et ainsi la découverte potentielle d'une meilleure solution.

La décomposition de l'erreur en généralisation montre que l'utilisation d'un ensemble peut se révéler intéressante en diminuant le biais et/ou la variance d'un classifieur unique. La question principale est de savoir quelle stratégie doit être mise en place pour obtenir un ensemble ayant une faible erreur en généralisation. Ceci est l'objet de la prochaine section.

4. Les méthodes d'agrégation de classifieurs

Les méthodes d'agrégation de classifieurs ont pour objectif de générer puis d'agréger un ensemble de modèles afin de construire un "méta-modèle" qui sera utilisé pour la classification de nouveaux exemples. Plus formellement, on induit T classifieurs $h_1(x), ..., h_T(x)$ qui sont ensuite agrégés par un processus adapté. Dans cette section, nous commençons par dresser un panorama des différentes méthodes d'agrégation de classifieurs puis nous nous intéressons aux stratégies complètes de génération d'ensembles du type *self-contained*. Parmi ces dernières nous distinguerons celles qui reposent sur des mécanismes aléatoires et fusionnent des classifieurs prédéfinis (notamment le *Bagging* et les forêts aléatoires) de celles qui sont adaptatives au sens où les résultats de chaque itération sont pris en compte dans l'itération suivante, ce qui est le cas du *Boosting*. Ces deux types de stratégies sont présentés avec les algorithmes correspondants et leurs variantes.

4.1. Agrégation des classifieurs existants

Ce type d'agrégation se base sur les algorithmes de vote. Il existe trois principales catégories de vote, décrites ci-dessous.

Vote simple : également appelé vote majoritaire, le vote simple consiste à affecter un poids équivalent à chaque classifieur et revient donc à construire un méta-classifieur correspondant à prédire la classe la plus fréquemment votée par les T modèles [Bauer and Kohavi, 1999].

Vote Pondéré : chaque classifieur reçoit un poids α_t habituellement proportionnel à sa performance estimée sur un échantillon test. Ce système de vote aboutit souvent à une meilleure performance que le vote simple [Bauer and Kohavi, 1999].

Majorité Pondérée : l'algorithme de Majorité pondérée [Littlestone and Warmuth, 1994] est une généralisation du HALVING algorithm [Littlestone, 1987]. Il est semblable au vote pondéré, à la différence que les poids α_t attribués aux différents classifieurs formant le comité sont appris eux-mêmes par un algorithme d'apprentissage.

4.2. Stratégies globales utilisant des mécanismes aléatoires

Nous décrivons dans cette partie les deux algorithmes les plus populaires basés sur des stratégies aléatoires : le *Bagging* [Breiman, 1996] et les Forêts aléatoires (*Random Forests*)

[Breiman, 2001a].

Le *Bagging*

Le *Bagging* (pour *Bootstrap Aggregating*) est une méthode qui combine des classifieurs pour obtenir un classifieur final (le classifieur agrégé). L'idée de base de cette méthode est de construire T réplications "perturbées" de l'échantillon d'apprentissage S puis d'induire un classifieur sur chacun de ces échantillons. Un vote à la majorité simple des T classifieurs est ensuite utilisé pour prédire la classe d'un nouvel exemple. La perturbation utilisée consiste à construire une réplication comme étant un échantillon *bootstrap* de l'échantillon initial (échantillon de même taille tiré aléatoirement avec remise dans l'échantillon initial).

L'algorithme du *Bagging* peut être résumé de la façon suivante :

Pour t **de** 1 **à** T **faire**
1. Construire un échantillon bootstrap S_t de S
2. Induire un classifieur h_t sur S_t

Fin Pour

Construire le méta-classifieur $h_{BAG}(x)$ donnant comme prédiction pour un individu x la classe la plus fréquemment votée par les T modèles.

Algorithme 1 – Le principe de Bagging

Le *Bagging* a pour effet de réduire fortement la variance d'une classifieur unique. L'effet de la réduction de variance se comprend par le fait qu'en entraînant des classifieurs sur des échantillons différents, on génère des classifieurs différents. Ainsi, en agrégeant les différents classifieurs par un vote à la majorité, on espère faire en sorte que $h_{BAG}(x)$ tende vers $h^*(x)$ (défini dans la précédente section) aboutissant à un classifieur ayant une variance très faible.

La notion de réduction de variance par le *Bagging* est très proche du concept de diversité entre classifieurs. En effet, il est préférable que les classifieurs induits soient les plus divers possibles. [Breiman, 2001a] a dans cette perspective donné une borne de l'erreur en généralisation du vote à la majorité d'un ensemble de classifieurs définie comme suit : $Pr_{(x,y)\equiv P}[h_{BAG}(x) \neq y] \leq \bar{\rho}(1-s^2)/s^2$, où $\bar{\rho}$ désigne la corrélation moyenne attendue entre les prédictions de chaque classifieur et s^2 est une fonction de leur performance individuelle. Cette borne est très suggestive, puisqu'elle indique que la performance du vote à la majorité sera d'autant meilleure que la corrélation entre les hypothèses sera faible et que leur force individelle sera forte en moyenne.

Le *Bagging* a ainsi pour objectif de construire des classifieurs "décorrélés" et parallèlement

assez forts individuellement, en les faisant apprendre sur des échantillons différents. Comme souligné dans [Breiman, 1996], le *Bagging* sera d'autant plus efficace que l'algorithme de base pour induire les classifieurs est instable, c'est à dire, produisant des résultats sensiblement différents lorsque l'échantillon d'apprentissage change. Classiquement, ce sont des arbres de décision qui sont utilisés en conjonction avec le *Bagging* puisqu'ils comptent parmi les algorithmes les plus instables. Dans ce contexte, les arbres sont généralement développés au maximum, c'est à dire jusqu'à l'obtention de feuilles les plus pures possibles (i.e, contenant des exemples de même classe). En effet, lorsque les arbres sont développés au maximum, le biais est généralement minimal et la variance forte. Toutefois, le *Bagging* agissant en réduisant la variance permettra d'obtenir un classifieur $h_{BAG}(x)$ avec un faible biais et une faible variance.

Les *Forêts Aléatoires*

[Breiman, 2001a] propose une amélioration du *Bagging* portant le nom de Forêts Aléatoires (*Random Forests*). Ces dernières consistent, comme le *Bagging*, à construire un ensemble d'arbres de décision, chacun induit sur un échantillon *bootstrap*. La différence réside dans le fait qu'à chaque niveau de chaque arbre, lorsque l'on recherche la meilleure coupure, au lieu de tester toutes les variables X_i possibles, on ne teste qu'un sous-ensemble de celles-ci, choisi aléatoirement. L'idée des forêts aléatoires est de diminuer la corrélation entre les arbres formant la forêt à travers ce nouveau mécanisme de randomisation des variables. Bien que ceci aboutisse également à une baisse de la pertinence individuelle des classifieurs, le résultat obtenu par les forêts aléatoires est généralement excellent [Breiman, 2001a].

4.3. Stratégies globales adaptatives

Principes du *Boosting*

L'idée principale du *Boosting*, en apprentissage machine, était d'améliorer les compétences d'un classifieur, supposé a priori instable, appelé *weak learner*. La méthode originale de [Shapire, 1990], améliorée par [Freund and Schapire, 1996], décrit AdaBoost (*Adaptative boosting*), l'algorithme de base du *Boosting*, pour la prédiction d'une variable binaire.

Le *Boosting* utilise le même principe que le *Bagging*, en construisant un ensemble de classifieurs qui sont ensuite agrégés par une moyenne pondérée des résultats ou un vote. La différence du *Boosting* par rapport au *Bagging* réside dans la façon de construire l'ensemble de classifieurs. En effet, le *Boosting* construit d'une façon récurrente et itérative l'ensemble des classifieurs. Chaque classifieur est une version adaptative du précédent en donnant plus de poids aux exemples mal prédits. Ce procédé permet à l'algorithme de se concentrer sur les exemples les plus difficiles à classifiés. L'agrégation de classifieurs permet au *Boosting* d'échapper au sur-apprentissage. Cette méthode réduit à la fois la variance et le biais.

Algorithme de base : AdaBoost (adaptative boosting)

Soit x_0 à prévoir $Z = (x_1, y_1), ..., (x_n, y_n)$ un échantillon

[Initialiser les poids]

$w = \{w_i = 1/n; i = 1, ..., n\}$.

Pour t **de** 1 **à** T **faire**

Estimer δ_t sur l'échantillon pondéré par w.

Calculer le taux d'erreur apparent :

$$\hat{\varepsilon_p} = \frac{\sum_{i=1}^{n} w_i 1\{\delta_t(x_i) \neq y_i\}}{\sum_{i=1}^{n} w_i}$$

Calculer les logit : $c_t = log((1\hat{\delta_p})/\hat{\delta_p})$.

les nouvelles pondérations : $w_i \leftarrow wi.exp[-c_t 1\{\delta_t(x_i) \neq y_i\}]; i = 1, ..., n$.

Fin Pour

Résultat du vote : $\phi_t(x_0) = signe[\sum_{t=1}^{T} c_t \phi_t(x_0)]$.

Algorithme 2 – Principe du *Boosting*

Initialement, chaque exemple possède le même poids, égale à $1/n$. Aprés la première classification, le poids des exemples évoluent à chaque itération et pour chaque nouvelle classification. Le poids d'un exemple w_i est inchangée si il est bien classé, par contre si il est mal classé, son poids croît d'une façon exponentielle. L'agrégation finale des prédictions $\sum_{t=1}^{M} c_t \phi_t(x_0)$ est une combinaison pondérée de la prédiction de chaque classifieur par son poids.

Variantes

L'algorithme Arcing (*Adaptively Resample and Combine*), proposé par [Breiman, 1998], est une combinaison entre la version aléatoire du *Bagging* et le principe adaptative du *Boosting*. Cette approche est une solution pour les classifieurs dans le cas où l'utilisation des pondérations pour les exemples soit impossible. En fait, Arcing, au lieu d'adapter les pondérations des exemples, tire à chaque itération un nouvel échantillon avec remise (*bootstrap*), mais selon des probabilités inversement proportionnelles aux poids du classifieur de l'itération précédente.

D'autres adaptations du *Boosting* ont été proposées :

- Pour le cas de la régression, Freund et Schapire [Freund and Schapire, 1996] ont proposé Adaboost.R.
- Pour le cas de catégorisation de textes (TC), on trouve les différentes versions d'AdaBoost

MH [Sebastiani et al., 2000a].
- AdaBoost M1, AdaBoost M2 [Freund and Schapire, 1996] sont des améliorations d'AdaBoost pour des données à classes multiples, où il y a introduction du degré de croyance.

4.4. Comparaison des méthodes d'agrégation

Dans cette partie on s'intéresse aux méthodes d'agrégation utilisées dans les algorithmes instables, donc pour le deuxième type d'agrégation, précisément le *Bagging* et le *Boosting*. En effet, le *Bagging* et le *Boosting* peuvent utiliser comme classifieur faible tout type de classifieur (régression, CART, réseaux de neurones) mais ils n'ont d'intérêt, et ne réduisent sensiblement l'erreur de prédiction, que dans le cas de modèles instables, donc plutôt non linéaires. Ils sont surtout mis en oeuvre en association avec des arbres comme classifieur de base.

L'étude comparative de G.Dietterich [Dietterich, 1999] montre bien que, dans le cas où les données sont faiblement bruitées, AdaBoost est plus performant que le Bagging et qu'il semble être immunisé contre le sur-apprentissage, car AdaBoost essaye directement d'optimiser les votes pondérés. Cette théorie est prouvée par (1) la diminution exponentielle de l'erreur empirique d'AdaBoost sur l'échantillon d'apprentissage avec le nombre d'itérations, (2) la diminution de l'erreur en généralisation qui continue à baisser même lorsqque l'erreur empirique a atteint son minimum. A l'opposé, le *Bagging* demande un grand nombre d'itérations pour que l'erreur en généralisation se stabilise. Contrairement au *Boosting*, l'erreur en généralisation ne diminue plus lorsque l'erreur en apprentissage a atteint son minimum. C'est ainsi que le *Bagging* est plus gourmand que le *Boosting* du point de vue espace mémoire puisque chaque classifieur doit être stocké pour la prédiction d'un nouvel exemple.

5. Le *Boosting* : faiblesses et améliorations

Certes l'étude comparative de G.Dietterich [Dietterich, 1999] a montré la supériorité d'*AdaBoost* sur le *Bagging* dans le cas de données faiblement bruitées, mais cette même étude montre aussi que sur des données fortement bruitées, *AdaBoost* présente un taux d'erreur parfois plus important que le *Bagging*. Ceci est expliqué par le fait que le *Boosting* tend à augmenter le poids des exemples bruités à chaque itération, puisqu'ils sont par nature mal prédits. La conséquence est que la combinaison des hypothèses faibles tend à sur-apprendre le bruit. La vitesse de convergence du *Boosting* se trouve également pénalisée sur ce type de données. Nous alons donc effectuer une étude des principales méthodes ayant vocation à améliorer le *Boosting* par rapport à ces deux faiblesses. Cet état de l'art sera réalisé selon deux axes de recherches. Le premier axe regroupe les approches abordant le problème de la gestion des données bruitées, sans laquelle des phénomènes de sur-apprentissage peuvent survenir. Le deuxième axe regroupe les approches portant sur les problèmes de vitesse de convergence.

5.1. Le sur-apprentissage

L'évolution des méthodes d'acquisition des bases de données rélles actuelle obligent les chercheurs à étudier l'impact du bruit sur le *Boosting* et améliorer les capacités de ce dernier. En effet, ces bases de données sont fortement bruitées en raison notamment des nouvelles technologies d'acquisition automatiques de données tel que le Web. En parallèle, des études telles que celle de Dietterich [Dietterich, 1999], Ratsch [Rätsch, 1998] et Schapire [Schapire and Singer, 1999] montrent bien qu'AdaBoost tend à sur-apprendre les données et surtout le bruit. De ce fait, un certain nombre de travaux récents ont tenté de limiter ces risques de sur-apprentissage. Ces améliorations se fondent essentiellement sur le fait qu'AdaBoost tend à augmenter le poids des exemples bruités de manière exponentielle. Deux types de solutions sont donc possibles pour réduire l'impact de ces données bruitées :

- Soit en détectant ces données et en les supprimant avant d'effectuer l'apprentissage, sur la base d'heuristiques efficaces. Une phase de prétraitement s'inspirant du domaine de sélection de prototypes est alors obligatoire ; c'est par exemple le cas des travaux de [Brodley and Friedl, 1996] ou de [Wilson and Martinez, 2000].
- Soit en détectant ces données tout au long du processus de *Boosting*, on parle alors d'une bonne gestion de bruit. Pour ce faire, les chercheurs se sont orientés vers l'amélioration des points forts du *Boosting* tels que la mise à jour des exemples mal classés, la maximisation de la marge, la signification des poids α_t qu'AdaBoost associe aux hypothèses et enfin le choix de l'apprenant faible.

On s'intéressera par la suite à la deuxième approche " gestion du bruit " tout en faisant un bilan de principales méthodes qui abordent le problème de la gestion de données bruitées.

Modification des poids des exemples

La mise à jour adaptative de la distribution des exemples, visant à augmenter le poids de ceux mal appris par le classifieur précédent, permet d'améliorer les performances de n'importe quel algorithme d'apprentissage. En effet, à chaque itération, la distribution courante favorise les exemples ayant été mal classés ($y(x) \neq h_t(x)$) par l'hypothèse précédente, ce qui caractérise bien l'adaptativité d'AdaBoost. De ce fait, plusieurs chercheurs ont proposé des stratégies portant sur une modification de la mise à jour des poids des exemples, pour éviter le sur-apprentissage. Celles-ci sont décrites ci-dessous.

Madaboost La modification proposée par [Domingo and Watanabe, 2000] essaye de réduire la croissance incontrôlée du poids de certains exemples (bruit). La solution proposée est de borner le poids de chaque exemple par sa probabilité initiale. De cette façon, les poids des exemples ne peuvent pas devenir arbitrairement grands par comparaison à ce que se produit dans AdaBoost. De ce fait, Madaboost demeure aussi simple qu'AdaBoost. De plus

cette approche a traité le problème d'utilisation d'AdaBoost dans le cadre de filtrage. Certes Madaboost fonctionne dans le cadre de filtrage. Cependant, il présente différentes faiblesses telles que des modifications de poids très faibles ou l'hypothèse que les avantages des classifieurs produits diminuent de façon monotonique. Bien que cette approche rende le *Boosting* résistant au bruit, des études montrent que la modification effectuée pénalise la convergence. En effet, la vitesse de convergence de Madaboost est beaucoup plus lente que celle d'AdaBoost.

Brownboost Brownboost est une version adaptative et opérationnelle de l'algorithme Boost-par-Majorité (BBM) [Freund, 2000]. Cet algorithme a été étendu au cas multi-classes par [McDonald et al., 2003] Les algorithmes de type BBM génèrent des hypothèses de même poids. L'algorithme obtenu incorpore un paramètre supplémentaire (le paramètre de temps), correspondant à la proportion de bruit dans les données d'apprentissage. Puisque l'algorithme sait à l'avance pour combien de temps il doit être exécuté, il n'essaiera pas d'adapter les exemples qui sont peu susceptibles d'être appris dans le temps restant, ce qui est généralement le cas des exemples bruités. Ainsi, au prix d'une bonne évaluation du paramètre de temps, BrownBoost est capable d'éviter le sur-apprentissage. En fait, les auteurs utilise une fonction de pondération assimilable à la probabilité d'une loi binomiale dépendant du nombre d'itérations finales k (temps total d'exécution), de l'itération courante i, du nombre de fois où l'exemple a déjà été correctement étiqueté r, et enfin de la probabilité de succès $1-\gamma$ imposée à toute hypothèse faible.

$\alpha_r^i = \binom{k-i-1}{k/2-r}(1/2+\gamma)^{(k/2)-r}(1/2-\gamma)^{(k/2)-i-1+r}$

On peut montrer qu'AdaBoost est un cas spécial de Brownboost car le paramètre de temps peut tendre vers l'infini. L'avantage de cette approche est que les données bruitées seront probablement détectées à un certain moment et leur poids cessera alors d'augmenter. Cependant, ces données bruitées peuvent parfois être détectées (trop) tardivement, entraînant une influence négative sur le classifieur final.

Logitboost L'algorithme Logitboost [Schapire and Singer, 1999] repose sur le principe qu'AdaBoost adapte un modèle de régression logistique aux données d'apprentissage. Un modèle additif est une approximation d'une fonction F(x) de la forme :

$$F(x) = \sum_{m=1}^{M} c_m f_m(x)$$

où les c_m des constantes à déterminer et les f_m des fonctions de base. F(x) étant l'hypothèse globale et f(x) les hypothèses faibles, on peut adapter un modèle à AdaBoost en minimisant le critère :

$$J(F) = E(e^{-yF(x)})$$

où y est la véritable étiquette appartenant à -1,1. LogitBoost minimise ce critère en employant les étapes de l'algorithme Newton-like pour adapter un modèle de régression logistique en optimisant le Log de la vraisemblance :

$$-log(1 + e^{-2yF(x)})$$

Modeste Boost Il s'agit d'un nouvel algorithme de *Boosting* [Vezhnevets and Vezhnevets, 2002], qui produit moins d'erreur en généralisation comparé à l'algorithme AdaBoost mais au prix d'une erreur d'apprentissage légèrement plus élevée. C'est une approche qui propose un nouveau schéma de *Boosting* conçu pour améliorer la généralisation. L'autre avantage de cette méthode est le critère d'arrêt normal, qui n'existe pas dans les autres techniques de *Boosting*. Cette approche utilise des distributions D_m , qui à chaque itération assignent des poids élevés aux exemples d'apprentissage mal classifiés par les premières itérations, et des distributions inverses $\overline{D_m}$, qui donnent au contraire des poids plus élevés aux exemples qui sont déjà correctement classifiés par des étapes antérieures. En plus, des mesures de la qualité de prédiction des classifieurs faibles sont calculées à chaque itération. De ce fait, la mise à jour repose sur la diminution de la contribution du classifieur, si cela fonctionne " trop bien " sur les données qui ont été déjà correctement classifiées. C'est pourquoi la méthode est appelée Modest AdaBoost - cet algorithme force les classifieurs à être "modeste" et travaille seulement dans leur domaine, défini par la distribution D_m. Une telle mise à jour peut réellement devenir nulle d'où le critère d'arrêt normal.

SmoothBoost SmoothBoost, suggéré par [Servedio, 2001], essaye de réduire l'effet de sur-apprentissage par des limites imposées à la distribution produite pendant le processus de *Boosting*. En particulier, un poids limité est assigné à chaque exemple individuellement pendant chaque itération. C'est contraire à AdaBoost, où les poids assignés aux différents exemples sont illimités. Ainsi, les exemples bruitées peuvent être excessivement soulignés pendant les itérations puisqu'ils sont assignés avec les poids extrêmement grands. En effet, cette version améliore l'algorithme d'AdaBoost en utilisant un poids spécial de coupure imposé aux valeurs de marge négatives (c'est-à-dire, les poids ne peuvent pas devenir trop grands). SmoothBoost converge plus lentement qu'AdaBoost mais la distribution résultante est beaucoup plus lisse que celle d'AdaBoost (d'où le nom). SmoothBoost est moins autonome qu'AdaBoost car il exige deux variables définies par l'utilisateur.

iAdaBoost Le principe de cette approche proposée par [Sebban and Suchier, 2003] est de construire autour de chacun des exemples une mesure d'information locale permettant d'évaluer les risques de sur-apprentissage, en utilisant un graphe de voisinage qui permet de mesurer l'information autour de chaque exemple. Grâce à ces mesures, une fonction qui traduit le niveau de nécessité de mettre à jour avec AdaBoost l'exemple. Cette fonction permet de gérer

à la fois les *outliers*, les recoupements de classes et les centres de clusters. Cet algorithme ressemble fortement à AdaBoost, puisque la principale modification relève de l'information recueillie au sein du graphe de voisinage et de son utilisation dans le calcul du coefficient de confiance du poids.

Modification de la marge

Certaines études, après une observation des algorithmes de *Boosting*, ont montré que l'erreur en généralisation décroît encore une fois que l'erreur en apprentissage est stable ou même nulle. L'explication est que même si tous les exemples d'apprentissage sont déjà bien classés, la poursuite du *Boosting* tend à maximiser les marges [Servedio, 2001] d'où la bonne performance du *Boosting*. À la suite de cette explication, certains ont cherché à modifier la marge explicitement soit en la maximisant soit en la minimisant dans le but d'améliorer les performances du *Boosting* face au sur-apprentissage. Plusieurs approches se sont succédées telles que :

AdaBoostReg AdaBoostReg, essaye d'identifier et d'enlever des exemples mal étiquetés de l'ensemble d'apprentissage, ou d'appliquer la contrainte de la marge maximale sur des exemples supposés mal étiquetés [Rätsch et al., 2001]. Il présente un nouveau concept, la marge molle (*Soft margin*), qui remplace la marge dure (*Hard margin*) employée dans la version traditionnelle d'AdaBoost et qui est moins susceptible de sur-apprentissage. Bien que cet algorithme manque d'une preuve formelle de convergence, AdaBoostReg généralise bien au-delà des données bruitées et empiriquement l'approche est vue comme l'une des méthodes de *Boosting* les plus performantes.

Marginal Boost Cette approche [Rätsch and Warmuth, 2001] considère que la combinaison linéaire des hypothèses faibles générées à chaque itération d'AdaBoost est un hyperplan dans l'espace des attributs ou chaque hypothèse est une dimension. L'amélioration se base sur l'observation expérimentale que les distances (marges) des exemples à l'hyperplan de séparation augmentent même lorsque l'erreur d'apprentissage est déjà nulle, c'est-à-dire tous les exemples sont du côté correct de l'hyperplan. En fait, il s'agit d'une version itérative d'AdaBoost qui maximise explicitement la marge minimum des exemples en limitant le nombre d'itérations et le nombre d'hypothèses utilisées dans la combinaison linéaire finale qui rapproche la marge maximale de l'hyperplan avec une certaine précision. Il s'agit de la première approche portant sur la convergence non-asymptotique d'un algorithme de *Boosting* pour la marge maximale qui est valide si l'espace d'hypothèses est infini. Cet algorithme converge rapidement au sens de la marge maximale, tandis que l'algorithme d'AdaBoost d'origine ne peut pas réaliser une grande marge.

Gentle AdaBoost Dans l'algorithme, proposé par Friedman en 1998, les auteurs font appel à une fonction de pondération [Friedman et al., 1998] qui croît moins vite que la fonction exponentielle. Gentle AdaBoost hérite des performances du *Boosting*, en se basant statistiquement d'une approximation d'un modèle additif de fonctions potentiellement simples, mais pouvant étendre la classe de fonctions approximables. Des expériences ont montré que cette approche donne des résultats meilleurs qu'AdaBoost sur des données bruitées mais pénalise la convergence.

Modification de poids des classifieurs

Lors de l'évaluation des performances du *Boosting*, des chercheurs se sont également interrogés sur la signification des poids $\alpha(t)$ qu'AdaBoost associe aux hypothèses produites. À la vue de l'algorithme, il s'agit d'une valeur déterminée sur une distribution artificielle, elle-même fonction des échecs et réussites des hypothèses précédentes. Cependant, ils ont noté lors d'expériences sur des données très simples que l'erreur en généralisation diminuait encore alors que l'apprenant faible avait déjà fourni toutes les hypothèses possibles. Autrement dit, lorsqu'une hypothèse apparaît plusieurs fois, elle vote finalement avec un poids, cumul de tous ses $\alpha(t)$, qui a peut-être un caractère plus absolu. De ce fait, plusieurs auteurs ont espéré approcher ces valeurs par un processus non adaptatif.

Localized Boost Cette approche se base sur l'observation que la combinaison des paramètres $\alpha(t)$ ne dépend pas des données d'entrées et qu'elle contribue de la même façon pour chaque point de l'espace (données). Pour remédier à ce problème, Locboost [Meir et al., 2000] est une alternative à la construction de l'ensemble de représentations d'experts et permet aux coefficients $\alpha(t)$ de dépendre des données. Ainsi on considère chaque hypothèse $h(t)$ comme un expert et $\alpha(t)$ comme un poids pour la prédiction du t_e expert. Donc pour chaque donnée, on a la prédiction de chaque expert h(t) et son poids $\alpha(t)$ local pour ce point de donnée. A chaque itération la fonction $\alpha(t)$ de l'ensemble de données est estimée en la limitant à la forme paramétrique simple.

Choix d'apprenant faible

Une question que plusieurs chercheurs ont posée à l'encontre des problèmes du *Boosting* est celle de l'apprenant faible : comment bien le choisir ? Les auteurs d'AdaBoost indiquent que la définition simple et claire d'un bon apprenant est :

"*an algorithm that gives small errors when run under AdaBoost*" [Freund and Schapire, 1996].

En particulier, faut-il autoriser ou même encourager l'apprenant faible à se tromper sur une partie des exemples d'apprentissage (les exemples de poids faibles) ? La réponse n'est pas évidente, surtout si l'on prend en compte qu'AdaBoost traite de manière privilégiée des

données non bruitées (son comportement naturel sur des données bruitées est de s'acharner sur les exemples difficiles à classer, donc sur le bruit). De ce fait, plusieurs travaux, décrits ci-dessous, se sont orientés vers l'analyse du choix du classifieur de base du *Boosting*.

GloBoost Dans cette approche, Fabien Torre a proposé d'utiliser un apprenant faible qui produit des hypothèses correctes, celles-ci peuvent donc s'abstenir sur une partie des exemples mais en aucun cas se tromper sur un exemple. Il s'agit de moindres généralisés maximalement corrects [Torre, 2004]. Ce classifieur est une nouvelle version du calcul de moindres généralisés corrects destinée à devenir un apprenant faible pour AdaBoost. Pour cela, il faut prendre en compte les poids des exemples, autrement dit favoriser la couverture des exemples de poids élevé. GloBoost présente une erreur faible, comparable à celle d'AdaBoost, avec l'avantage que le calcul peut être réparti sur plusieurs machines. Cependant, la seule particularité de cet apprenant faible est de fournir des hypothèses qui ne se trompent pas ; et cela simplifiait déjà le calcul des poids dans AdaBoost.

RankBoost Cette nouvelle approche est utilisée, pour résoudre le problème de combinaison de préférences. Son but est de classifier des tâches de préférences [Dietterich, 1999]. De ce fait, RankBoost se base sur un apprenant faible qui accepte comme données d'entrées des attributs de préférences (rank) qui ne sont que des fonctions. Donc l'objectif de l'apprenant est de produire un bon *ranking* de tout les instances même celles non observées.

5.2. La vitesse de convergence

A part le problème de sur-apprentissage rencontré par le *Boosting* dans les bases de données modernes déjà évoqué précédemment, il existe un autre problème celui de la vitesse de convergence des algorithmes de *Boosting*, spécialement AdaBoost. En effet, en cas de présence d'un fort recoupement entre les densités de probabilités des différentes modalités de la classe à apprendre, l'erreur optimale de l'algorithme d'apprentissage utilisé est atteinte tardivement (T très grand) [Sebban and Suchier, 2003]. En d'autres termes, AdaBoost "perd" du temps, et donc des itérations, à repondérer des exemples qui ne méritent en théorie aucune attention, puisqu'ils se situent à la frontière bayésienne des classes, où l'erreur est incompressible. Or, les bases de données modernes, sur lesquelles nous exerçons nos algorithmes d'apprentissage, présentent souvent une erreur bayésienne non nulle. Donc des recherches se sont faites pour détecter les exemples qui font partie de l'erreur bayésienne et améliorer ainsi les performances de *Boosting* en terme de convergence.

iBoost

Cet algorithme est conçu afin de spécialiser les classifieurs faibles sur les exemples sur lesquels leurs prédictions sont performantes [Kwek and Nguyen, 2002]. A chaque itération, iBoost utilise des variables indicatrices pour pondérer l'exemple en se basant sur les hypothèses faibles $h(t)$. Ces variables peuvent être considérées comme des coefficients présentants l'adéquation de l'exemple avec $h(t)$. Les résultats montrent la performance de iBoost surtout de point de vue taux de succès.

iAdaBoost

Cette approche est conçue non seulement pour améliorer AdaBoost face au risque de sur-apprentissage mais également pour améliorer sa convergence. En fait, l'idée de base de l'amélioration de la convergence de iAdaBoost est la modification du théorème de (Schapire et Singer,1998), qui prouve que la borne sur l'erreur d'apprentissage issu du classifieur final H est atteinte par minimisation du produit des Zt, obtenus pour chacune des T hypothèses faibles. Cette modification est réalisée afin d'intégrer le risque de Bayes et mettre en exergue les situations où certains exemples de classes différentes partagent la même représentation. Les effets de cette modification sont une convergence plus rapide vers le risque optimal et une réduction du nombre d'hypothèses faibles à construire.

RegionBoost

RegionBoost [Maclin, 1998] utilise une nouvelle stratégie de pondération pour chaque classifieur. Cette pondération se base sur les k plus proches voisins de l'exemple à étiqueter lors du vote. Cette approche permet de spécialiser chaque classifieur sur des régions de l'ensemble d'apprentissage d'où son nom. Malgré cette amélioration, les résultats de RegionBoost montrent une certaine sensibilité au bruit.

Robust Boost

Robust Boost est une amélioration de *Boosting*, proposé par [Nock and Lefaucheur, 2002]. Dans le cas des exemples ayant reçu la même proportion de vote parmi les T hypothèses, Robust Boost attribut à chaque nouvel exemple la classe de la majorité (pondérée). Une étude théorique de sa résistance au bruit ainsi que la croissance des ses marges est réalisée.

6. Contributions : Amélioration du *Boosting* face aux données bruitées (GloutonBoost, AdaBoost Hybride)

Comme nous l'avons plusieurs fois souligné dans ce chapitre, l'un des points faibles d'AdaBoost est sa forte dégradation de performance en présence de bruit. Intuitivement, on comprend qu'après un certain nombre d'itérations, l'algorithme se concentre sur les exemples les plus difficiles à classer. Ceci résulte en une mise à jour de la distribution P_t qui tendra à donner des poids forts sur les exemples bruités, puisque ceux-ci seront sans doute les plus difficiles à classer.

Une récente et intéressante étude de [Rudin et al., 2004] a montré que dans certaines situations, et notamment en présence de données bruitées, AdaBoost avait un comportement chaotique à partir d'un certain nombre d'itérations. [Breiman, 2001b] avait déjà remarqué qu'AdaBoost ne convergeait que très rarement vers une solution stable, mais que son comportement ressemblait plus à celui d'un système ergodique. En fait, [Rudin et al., 2004] ont montré que même en présence de données non bruitées et dans de bonnes conditions, AdaBoost ne convergeait jamais. Dans ces derniers cas, le comportement de la mise à jour des distributions P_t avait un comportement cyclique modifiant les poids d'un petit groupe précis d'exemples.

Afin de rendre AdaBoost résistant au bruit et de lui éviter ces comportements cycliques chaotiques empêchant la convergence, il apparaît nécessaire de modifier sa mise à jour des poids (distributions). Notre idée consiste à prendre en compte à une itération t, non pas seulement la dernière hypothèse générée pour mettre les poids à jour, mais toutes les hypothèses précédemment générées.

6.1. GloutonBoost

Une première solution consiste à utiliser le processus glouton suivant, que nous nommons **GloutonBoost** :

Soit X_0 à prévoir et $S = (x_1, y_1), ..., (x_n, y_n)$ un échantillon

Pour i **de** 1 **à** n **faire**

  Initialiser les poids $p_0(x_i) = 1/n$

Fin Pour

Pour t **de** 1 **à** T **faire**

  Tirer un échantillon d'apprentissage S_t dans S selon les probabilités p_t.

  Construire une hypothèse h_t sur S_t par un algorithme d'apprentissage A

  Soit ϵ_t l'erreur apparente de h_t sur S avec $\epsilon_t = \sum poids \;\; des \; exemples$ tel que $argmax(\sum_{i=1}^{t} \alpha_i h_i(x_i) \neq y_i)$

  Calculer $\alpha_t = 1/2ln((1 - \epsilon_t)/\epsilon_t)$.

  Pour i **de** 1 **à** m **faire**

    $P_{t+1}(x_i) \leftarrow (p_0(x_i)/Z_t)e^{-\alpha_t}$ si $argmax(\sum_{i=1}^{t} \alpha_i h_i(x_i)) = y_i$ (bien classé)

    $P_{t+1}(xi) \leftarrow (p_0(x_i)/Z_t)e^{+\alpha_t}$ si $argmax(\sum_{i=1}^{t} \alpha_i h_i(x_i)) \neq y_i$ (mal classé)

    Z_t est une valeur de normalisation telle que $\sum_{i=1}^{n} p_t(x_i) = 1)$

  Fin Pour

Fin Pour

Fournir en sortie l'hypothèse finale : $H(x) = argmax \;\; y \in Y \sum_{t=1}^{T} \alpha_t$

Algorithme 3 – Pseudo code de GloutonBoost

6.2. Explication de GloutonBoost

GloutonBoost est un pur générateur d'hypothèses qui cherche à chaque itération l'hypothèse qui prédit le mieux les exemples mal prédits par la combinaison linéaire des hypothèses précédemment générées. Deux différences le distinguent d'AdaBoost. Tout d'abord, le classifieur courant n'est plus une simple hypothèse mais une combinaison linéaire d'hypothèses. Ensuite, l'algorithme calcule la mise à jour de la distribution sans se baser sur la précédente distribution (celle de l'itération précedente) mais sur la distribution initiale (tous les exemples avec le même poids). Autrement dit, on évalue les exemples initiaux non pondérés mal classés par le classifieur courant. Cela produit un nouvel échantillon qui sera utilisé pour générer une nouvelle hypothèse. L'intérêt d'un tel processus est qu'il converge nécessairement vers une solution stable sous l'hypothèse qu'à chaque itération on soit capable de trouver un classifieur faible, i.e, avec une erreur $\epsilon_t < 1/2$ sur l'échantillon pondéré.

Théorème 1. *Sous l'hypothèse qu'à chaque itération, l'apprenant généré par GloutonBoost soit au moins un apprenant faible, i.e,* $\forall t$, $\epsilon_t < 1/2$, *GloutonBoost converge nécessairement vers une solution stable lorsque* $T \to \infty$.

Preuve : La preuve repose sur un résultat de [Freund and Schapire, 1997] qui garantit qu'à l'itération $t+1$, l'erreur sur l'échantillon d'apprentissage du classifeur combinée H_{t+1} d'AdaBoost est toujours inférieure à celle de H_t, le classifieur combiné de l'itération t. Plus précisément, nous avons [Freund and Schapire, 1997] :

$$err(H_{t+1}) < err(H_t) \times \sqrt{1 - 4(1-\epsilon_t)^2} \tag{I.5}$$

où err(.) désigne l'erreur empirique. Pour $\epsilon_t < 1/2$, nous avons $\sqrt{1-4(1-\epsilon_t)^2} < 1$ et donc $err(H_{t+1}) < err(H_t)$. Commençons par observer que la solution de GloutonBoost est strictement équivalente à la solution d'AdaBoost pour les deux premières itérations $H_1^{GLO}(x) = H_1^{ADA}(x)$. Ensuite, à chaque itération, on modifie la distribution P_t de façon équivalente à AdaBoost, à l'exception que la distribution est modifiée de façon à surpondérer les exemples mal prédits par la combinaison linéaire des hypothèses précédemment générées. Cette façon de procéder revient donc à appliquer une itération d'AdaBoost sur le classifieur courant H_t^{GLO}. Ainsi, conformément à $(I.5)$, nous avons $H_{t+1}^{GLO} < H_t^{GLO} \times \sqrt{1-4(1-\epsilon_t)^2}$. Par récurrence, à l'itération T, nous avons :

$$H_T^{GLO} \leq \prod_{t=1}^{T} \sqrt{1-4(1-\epsilon_t)^2}. \tag{I.6}$$

Suivant [Freund and Schapire, 1997], nous avons également :

$$\prod_{t=1}^{T} \sqrt{1-4(1-\epsilon_t)^2} \leq \exp\left(-2\sum_{t=1}^{T}(1-\epsilon_t)^2\right) \tag{I.7}$$

Finalement, pour $\epsilon_t < 1/2$, il vient :

$$\exp\left(-2\sum_{t=1}^{T}(1-\epsilon_t)^2\right)_{T\to\infty} \to 0 \tag{I.8}$$

Le point (I.8) garantit la non modification de la distribution P_t et donc la convergence de l'algorithme GloutonBoost.

L'équation $(I.8)$ suggère, de façon analogue à AdaBoost, que GloutonBoost fait décroître l'erreur d'apprentissage à vitesse exponentielle. Toutefois, contrairement à AdaBoost, l'algorithme s'arrête lorsque celle-ci est égale à 0. En pratique, AdaBoost parvient à obtenir une erreur nulle très rapidement, disons autour d'une dizaine d'itérations. Cela laisse supposer que GloutonBoost est un algorithme très rigide comportant peu de classifieurs. L'un des problèmes potentiels à son utilisation peut être, de façon analogue aux algorithmes de type descente du gradient, la convergence vers une solution trop peu régularisée, i.e, pas assez généralisable.

Plusieurs auteurs, notamment [Friedman, 1999], ont remarqué que l'injection de randomization dans AdaBoost pouvait en améliorer les résultats. Une autre solution, également proposée par [Friedman et al., 1998] suggère l'emploi de processus de *shrinkage* pour une meilleure régularisation. Afin de régulariser plus fortement GloutonBoost, nous proposons de modifier encore la procédure de mise à jour des distributions P_t pour une convergence plus lente et donc une meilleure régularisation.

6.3. AdaBoost Hybride

Notre idée directrice consiste à conserver le processus de mémorisation de GloutonBoost qui fait évoluer le classifieur faible d'itérations en itérations. Toutefois, la recherche des hypothèses sera effectuée sur une distribution P_t modifiée pas à pas comme dans AdaBoost et non pas sur la pondération initiale P_0. Nous appelons cet algorithme AdaBoost Hybride [Bahri and Maddouri, 2008], [Bahri et al., 2009], décrit par l'algorithme suivant :

Soit X_0 à prévoir et $S = (x_1, y_1), ..., (x_n, y_n)$ un échantillon
Pour i **de** 1 **à** n **faire**
  Initialiser les poids $p_0(x_i) = 1/n$
Fin Pour

Pour t **de** 1 **à** T **faire**
  Tirer un échantillon d'apprentissage S_t dans S selon les probabilités p_t.
  Construire une hypothèse h_t sur S_t par un algorithme d'apprentissage A
  Soit ϵ_t l'erreur apparente de h_t sur S avec $\epsilon_t = \sum poids \;\; des \; exemples$ tel que $argmax(\sum_{i=1}^{t} \alpha_i h_i(x_i) \neq y_i)$
  Calculer $\alpha_t = 1/2ln((1-\epsilon_t)/\epsilon_t)$.
  Pour i **de** 1 **à** m **faire**
    $P_{t+1}(x_i) \leftarrow (p_t(x_i)/Z_t)e^{-\alpha_t}$ si $argmax(\sum_{i=1}^{t} \alpha_i h_i(x_i)) = y_i$ (bien classé)
    $P_{t+1}(xi) \leftarrow (p_t(x_i)/Z_t)e^{+\alpha_t}$ si $argmax(\sum_{i=1}^{t} \alpha_i h_i(x_i)) \neq y_i$ (mal classé)
    Z_t est une valeur de normalisation telle que $\sum_{i=1}^{n} p_t(x_i) = 1)$
  Fin Pour
Fin Pour

Fournir en sortie l'hypothèse finale : $H(x) = argmax \;\; y \in Y \sum_{t=1}^{T} \alpha_t$

Algorithme 4 – Pseudo code d'AdaBoost Hybride

AdaBoost Hybride modifie le classifieur faible employé en ajoutant les hypothèses comme

GloutonBoost. Toutefois, la modification d'itération en itération de la mise à jour des distributions aura pour effet de ralentir la convergence de l'algorithme en découplant la phase de génération des hypothèses de la mise à jour des distributions. De façon analogue à GloutonBoost, AdaBoost Hybride est aussi convergent.

Conjecture. *Sous l'hypothèse qu'à chaque itération, l'apprenant généré par AdaBoost Hybride soit au moins un apprenant faible, i.e,* $\forall t$, $\epsilon_t < 1/2$, *AdaBoost Hybride converge nécessairement vers une solution stable lorsque* $T \to \infty$.

Explication : Cette conjecture repose sur des éléments équivalents à ceux du Théorème 1. Toutefois, pour que cette conjecture soit vraie, il est nécessaire que la combinaison linéaire générée à une itération t soit un apprenant faible de l'échantillon pondéré courant. Bien que n'ayons aucune garantie théorique sur ce point (à cause du découplage mise à jour des distributions - construction de l'hypothèse courante), nous verrons plus loin qu'AdaBoost Hybride a toujours convergé vers une solution stable sur tous les jeux de données sur lequel nous l'avons testé.

6.4. Explication d'AdaBoost Hybride

La modification au sein de l'algorithme porte sur la repondération des exemples et sur le calcul de l'erreur.

- Modification des poids des exemples : à chaque itération, on fait appel aux avis des experts déjà utilisés (hypothèses des itérations antérieures) pour mettre à jour les poids des exemples. En effet, on ne compare pas seulement la classe prédite par l'hypothèse à l'itération courante avec la classe réelle mais la somme des hypothèses pondérées depuis la première itération jusqu'à l'itération courante. Si cette somme vote pour une classe différente de la classe réelle alors une mise à jour exponentielle similaire à celle d'*AdaBoost* est appliquée à l'exemple mal classé. Ainsi, cette modification laisse-t-elle l'algorithme s'intéresser seulement aux exemples qui sont soit mal classés soit pas encore classés. De ce fait, des résultats portant sur l'amélioration de la vitesse de convergence sont attendus, de même pour la réduction de l'erreur de généralisation étant donnée la richesse de l'espace des hypothèses à chaque itération.
- Modification du calcul de l'erreur $\epsilon(t)$ de l'hypothèse à l'itération t : cette modification s'intéresse plutôt au calcul de $\alpha(t)$ le coefficient de classifieur (hypothèse) à chaque itération . En effet, ce coefficient dépend du calcul de l'erreur apparente $\epsilon(t)$. A chaque itération, AdaBoost Hybride prend en considération les hypothèses précédentes lors du calcul de $\epsilon(t)$.

  De ce fait, l'erreur apparente à chaque itération est le poids des exemples prédits de façon erronée par les hypothèses pondérées des itérations antérieures. Cette modification

a un effet de lissage et laisse l'algorithme à chaque itération très dépendant des autres itérations. Des résultats améliorant surtout l'erreur en généralisation sont attendus puisque le poids de chaque hypothèse (coefficient $\alpha(t)$) est calculé à partir des hypothèses précédentes.

7. Performances de GloutonBoost et AdaBoost Hybride

Dans cette section, nous comparons les performances de nos algorithmes GloutonBoost et AdaBoost Hybride avec celles d'AdaBoost, l'algorithme de référence du *boosting*, et celles de BrownBoost qui est l'algorithme de *Boosting* résistant au bruit le plus connu en raison de l'efficacité de son paramètre de temps.

Cette comparaison utilise comme critères d'appréciation le taux d'erreur en généralisation, le couple rappel-précision et la vitesse de convergence. L'apprenant faible utilisé est l'algorithme C4.5, choisi suite à l'étude de [Dietterich, 1999] qui a montré que C4.5 est très sensible au bruit. Pour estimer sans biais les taux de succès en généralisation, nous avons fait appel à une procédure de validation croisée en 10 segments.

Pour mener ces expériences, nous avons volontairement retenu des jeux de données variés, récupérés sur le site de l'UCI [Stolfo et al., 1999]. En effet, certains jeux de données ont des valeurs manquantes (NHL, VOTE, HEPATITIS, HYPOTHYROID), d'autres ont une classe à prédire qui comporte plusieurs modalités (IRIS : variable de classe à 3 modalités, DIABETES : 4 modalités, ZOO : 7 modalités, IDS : 12 modalités). En outre, ces jeux présentent un nombre d'attributs très différent (STRAIGHT : 2 attributs, contre IDS : 35 attributs). Le tableau II.9 décrit les 15 bases de données utilisées dans ces expérimentations.

Nous avons divisé nos expérimentations en plusieurs parties. Dans la première partie, nous avons calculé pour chacun des 15 jeux de données choisis, l'erreur en généralisation, le rappel et la précision de chacun des 4 algorithmes retenus. Dans la deuxième partie, nous avons bruité aléatoirement ces bases de données avec un taux de bruit de 20%, pour observer le comportement des différents algorithmes. Ce taux a été choisi en se basant sur l'étude de [Dietterich, 1999] qui mettait en évidence les résultats décevants d'AdaBoost en présence de bruit. Dans la dernière partie, nous avons établi un diagnostic de convergence de ces différents algorithmes en se fondant sur le nombre d'itérations effectuées.

Afin de tester la signification des résultats obtenus (les différences sont-elles suffisamment importantes pour que l'on puisse considérer que les conclusions ont une portée générale qui ne dépend pas des jeux de données utilisées ?), il est nécessaire de pratiquer des tests, tout en évitant de multiplier les tests afin de limiter le risque de fausse découverte. Pour un critère donné, par exemple le rappel, les résultats des différentes méthodes figurant dans le tableau correspondant se présentent comme des échantillons appariés. Pour chaque tableau, nous avons pratiqué d'abord une analyse de variance à un facteur fixe (méthode) et un facteur aléatoire

(jeu de données) ayant pour but de prendre en compte l'appariement. Ce type d'analyse est relativement robuste face au défaut de normalité. Les différentes anova pratiquées ont donné une p-value significative. Nous avons alors utilisé le test de Student pour les comparaisons de moyennes 2 à 2. Nous avons doublé ce test par le test du signe pour échantillon appariés, qui a l'avantage d'être *distribution-free* et d'avoir une interprétation immédiate. Par exemple, supposons que la méthode A l'emporte 12 fois sur 15 sur la méthode B. on se demande si ce ce résultat peut être le fruit du hasard, comme le serait le résultat du lancer d'une pièce équilibrée (hypothèse nulle, notée H_0). On calcule la $p-value$ du résultat observé en formant $p-value = 2p\prime_3$ où $p\prime_3$ désigne la probabilité cumulée de 3 dans la table de la loi binomiale de paramètres 15 et 0,5. Avec $p-value = 0.0352$, au risque de se tromper 5 fois sur 100 en refusant H_0, on en conclut qu'il y a bien une différence entre les deux méthodes. Toutes les $p-values$ associées à une différence de moyenne sont donc calculées par le test de Student pour échantillons appariés, alors que toutes les $p-values$ associées à une comparaison des nombres de victoires et de défaites sont calculées à partir du test du signe pour échantillons appariés.

Bases de données	**Nb. Inst**	**Attrib**	**Cl. Pred**	**Val manq**
IRIS	150	4 numeric	3	non
NHL	137	8 numeric and symbolic	2	oui
VOTE	435	16 boolean valued	2	oui
WEATHER	14	4 numeric and symbolic	2	non
CREDIT-A	690	16numeric and symbolic	2	oui
TITANIC	750	3 symbolic	2	non
DIABETES	768	8 numeric	2	non
HYPOTHYROID	3772	30 numeric and symbolic	4	oui
HEPATITIS	155	19 numeric and symbolic	2	oui
CONTACT-LENSES	24	4 nominal	3	non
ZOO	101	18 numeric and boolean	7	non
STRAIGHT	320	2 numeric	2	non
IDS	4950	35 numeric and symbolic	12	non
LYMPH	148	18 numeric	4	non
BREAST-CANCER	286	9 numeric and symbolic	2	oui

TABLE I.1 – Caractéristiques des bases de données

7.1. Comparaison avant bruitage

Dans cette partie, les quatre algorithmes ont été appliqué à chacun des 15 jeux de données en choisissant d'effectuer 20 itérations pour chacun des algorithmes. Le choix du nombre d'itérations sera expliqué dans la dernière partie de ces expériences. Ce sont les mêmes échantillons qui ont été utilisés en validation croisée pour les différents algorithmes afin d'assurer une meilleure comparaison.

Comparaison en terme d'erreur de généralisation

Le tableau I.2 présente les taux d'erreur en généralisation pour les algorithmes AdaBoost M1, BrownBoost, GloutonBoost et AdaBoost Hybride sur chacun des 15 jeux de données.

L'analyse des résultats montre les effets positifs de l'approche GloutonBoost et AdaBoost hybride. En effet, face à AdaBoostM1, GloutonBoost et AdaBoost Hybride provoquent une diminution du taux moyen d'erreur de respectivement 2,8 points ($p - value = 0,0317$) et 2,4 points ($p - value = 0.0373$), contre 2,5 points pour BrownBoost ($p - value = 0.0424$). Le gain de AdaBoost Hybride sur BrownBoost est en moyenne de 0,25 point, ce qui n'est pas significatif sur 15 jeux de données.

AdaBoost Hybride est battu seulement 1 fois sur 15 par AdaBoostM1 (avec 2 égalités, $p - value = 0.0034$), alors que GloutonBoost est battu 2 sur 15 (avec 2 égalités, $p - value = 0.0225$) par AdaBoostM1 (pour les bases WEATHER et HYPOTHYROID). AdaBoost Hybride l'emporte 9 fois sur 15 sur BrownBoost (avec en plus 2 égalités), ce qui n'est pas significatif.

C'est seulement pour la base LYMPH que l'approche hybride donne une erreur de généralisation plus élevée que l'approche classique, mais la différence est faible. Nous remarquons des améliorations importantes de l'erreur en généralisation sur les bases NHL, CONTACT-LENS et BREAST-CANCER pour l'approche gloutonne et surtout pour l'approche hybride. Par exemple l'erreur en généralisation de la base BREAST-CANCER passe de 45.81% à 30.41%.

Ce gain en faveur de AdaBoost Hybride montre bien qu'en exploitant des hypothèses générées aux itérations antérieures pour corriger les poids des exemples, il est possible d'améliorer les performances du *Boosting*. Ceci peut être expliqué par la précision du calcul de l'erreur apparente $\epsilon(t)$ et par conséquent du calcul du coefficient du classifieur $\alpha(t)$ ainsi que par la richesse de l'espace des hypothèses à chaque itération puisqu'il s'agit de l'ensemble des hypothèses générées aux itérations précédentes et de l'itération courante.

Comparaison en termes de rappel et de précision

Les résultats encourageants, trouvés précédemment, nous amènent à approfondir l'analyse de notre nouvelle approche. Dans cette partie nous essayons de connaître l'impact de GloutonBoost et AdaBoost Hybride sur le rappel et la précision pour mieux comprendre le fonctionnement de ces algorithmes et nous assurer que l'amélioration de l'erreur ne se double pas d'un recul sur le rappel ou la précision.

Le tableau I.3 présente les valeurs du rappel obtenues avec les algorithmes AdaBoost M1, BrownBoost, GloutonBoost et AdaBoost Hybride sur chacun des jeux de données. Les résultats obtenus confortent les résultats précédents. En effet, AdaBoost Hybride augmente en moyenne le rappel de 4.8 points ($p - value = 0.0129$), alors que GloutonBoost l'augmente de 2.3 points ($p - value = 0.0277$). BrownBoost n'augmente le rappel que de 1.4 points en moyenne par rapport à AdaBoostM1, ce qui n'est pas significatif. De façon plus remarquable, on doit noter

Bases de Données	AdaBoost M1	BrownBoost	GloutonBoost	AdaBoostHyb
IRIS	6.00	3.89	3.57	**3.00**
NHL	35.00	30.01	30.65	**28.00**
VOTE	4.36	4.35	4.25	**4.13**
WEATHER	21.42	**21.00**	21.54	**21.00**
CREDIT-A	15.79	**13.00**	14.09	13.91
TITANIC	**21.00**	24.00	**21.00**	**21.00**
DIABETES	27.61	**25.05**	25.13	25.56
HYPOTHYROID	0.53	0.6	0.55	**0.42**
HEPATITIS	15.62	14.10	14.65	**14.00**
CONTACT-LENSES	25.21	**15.86**	17.98	16.00
ZOO	**7.00**	7.23	**7.00**	**7.00**
STRAIGHT	2.40	**2.00**	2.24	**2.00**
IDS	1.90	0.67	0.69	**0,37**
LYMPH	19.51	**18.54**	19.13	20.97
BREAST-CANCER	45.81	31.06	30.89	**30.41**

TABLE I.2 – Erreur en généralisation

que AdaBoost Hybride gagne en moyenne 3.4 points de rappel sur BrownBoost ($p - value = 0,0343$).

AdaBoost Hybride n'est jamais battu par AdaBoostM1 (13 victoires et 2 égalités, $p - value = 0,0002$) et il est battu seulement 1 fois (sur la base ZOO) par BrownBoost (13 victoires et 1 égalité, $p - value = 0,0018$), ce qui atteste d'une supériorité très significative d'AdaBoost Hybride en termes de rappel. Ajoutons que AdaBoost Hybride n'est jamais battu par GloutonBoost (11 victoires, 4 égalités) et que son avantage moyen de 2.5 points est significatif.

On notera que le rappel est d'autant plus amélioré par AdaBoost Hybride que l'on opère sur des jeux de données ayant un taux d'erreur en généralisation élevé et un rappel faible (BREAST CANCER, LYMPH et CONTACT LENSES).

Le tableau I.4 présente les valeurs obtenues pour la précision sur les différents jeux de données avec chacun des quatre algorithmes étudiés. On peut opérer les constations suivantes :

- BrownBoost, GloutonBoost et AdaBoost Hybride améliorent de façon très significative la précision d'AdaboostM1 (*p-values* inférieures à 0.01).
- empiriquement, c'est AdaBoost Hybride qui obtient la meilleure précision, avec notamment un avantage moyen de 3.4 point de pourcentage sur AdaBoostM1, alors que les résultats de BrownBoost et GloutonBoost sont équivalents.
- l'avantage de AdaBoost Hybride sur BrownBoost qui est $1,3$ point n'est pas significatif ($p - value = 0.1185$).

En résumé, cette première partie de nos expériences a mis en évidence le bon comportement d'AdaBoost Hybride qui l'emporte très significativement sur AdaBoostM1 pour chacun des critères d'erreur en généralisation, de précision et de rappel. Face à BrownBoost, les résultats

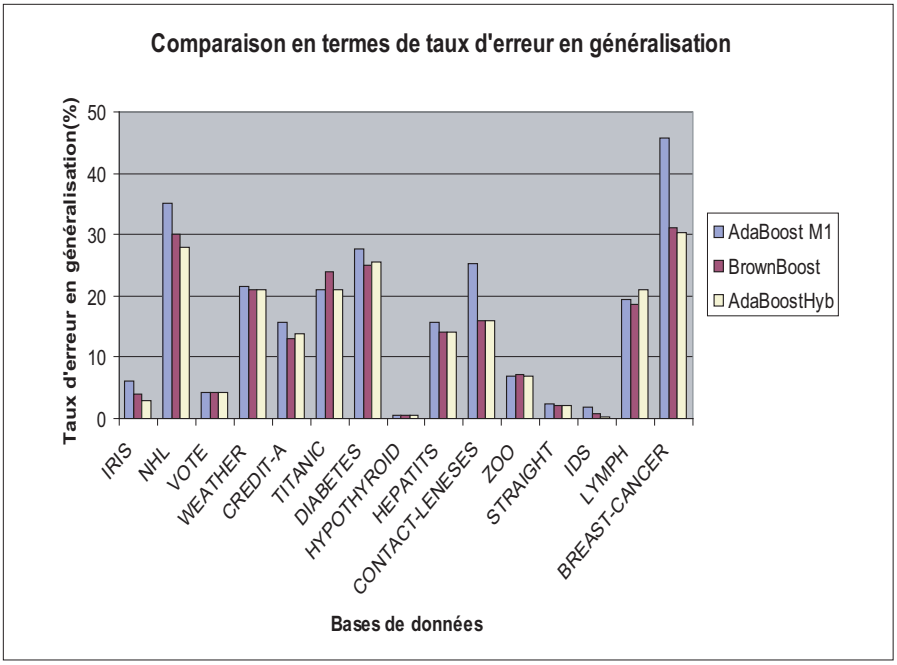

FIGURE I.2 – Comparaison en terme d'erreur en généralisation

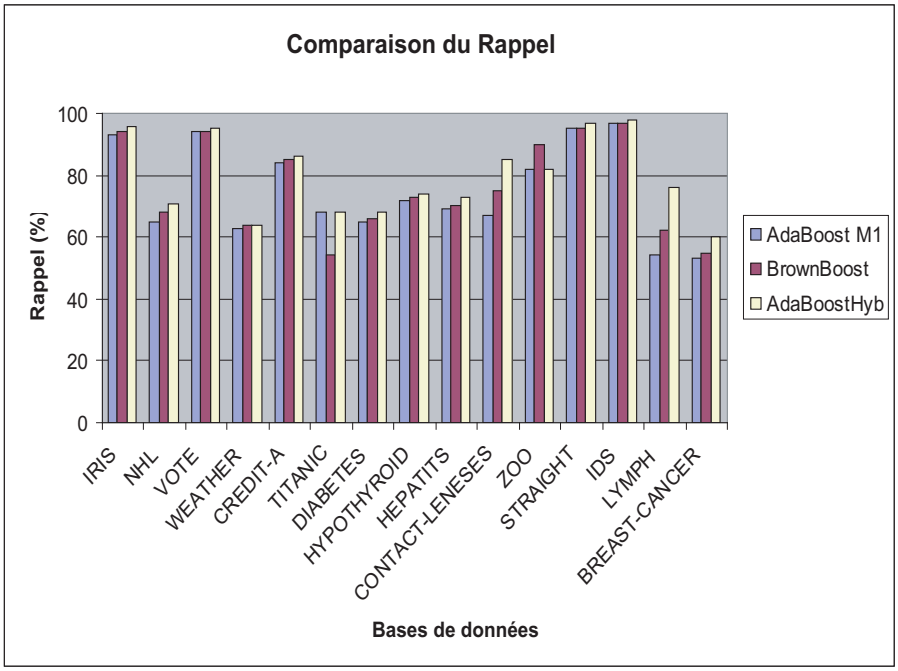

FIGURE I.3 – Comparaison en terme de rappel

Databases	AdaBoost M1	BrownBoost	GloutonBoost	AdaBoostHyb
IRIS	0.93	0.94	0.95	**0.96**
NHL	0.65	0,68	0.68	**0.71**
VOTE	0.94	0.94	0.95	**0.95**
WEATHER	0.63	0.64	0.64	**0.64**
CREDIT-A	0.84	0.85	0.85	**0.86**
TITANIC	**0.68**	0.54	0.68	**0.68**
DIABETES	0.65	0.66	0.66	**0.68**
HYPOTHYROID	0.72	0.73	0.73	**0.74**
HEPATITIS	0.69	0.70	0.71	**0.73**
CONTACT-LENSES	0.67	0.75	0.72	**0.85**
ZOO	0.82	**0.9**	0.82	0.82
STRAIGHT	0.95	0.95	0.94	**0.97**
IDS	0,97	0.97	0.97	**0,98**
LYMPH	0,54	0.62	0.68	**0,76**
BREAST-CANCER	0,53	0.55	0.56	**0.60**

TABLE I.3 – Rappel d'Adaboost M1, BrownBoost, GloutonBoost et AdaBoost Hybride

Bases de Données	AdaBoost M1	BrownBoost	GloutonBoost	AdaBoostHyb
IRIS	92.11	95.35	95.89	**96.77**
NHL	61.56	67.89	68.12	**79.75**
VOTE	85.88	86.45	87.09	**88.13**
WEATHER	64.32	**67.53**	65.45	**67.53**
CREDIT-A	78.89	**80.03**	79.31	79.87
TITANIC	**69.20**	67.56	68.57	**69.20**
DIABETES	62.56	**65.65**	64.12	65.43
HYPOTHYROID	96.31	96.75	97.03	**97.21**
HEPATITIS	76.31	76.87	77.12	**77.63**
CONTACT-LENSES	62.32	**69.89**	67.78	69.56
ZOO	92.54	91.23	**92.67**	**92.67**
STRAIGHT	95.76	**96.71**	95.89	**96.71**
IDS	88.14	90.98	92.43	**92.69**
LYMPH	78.69	**79.76**	78.65	78.77
BREAST-CANCER	62.34	65.45	65.23	**65.78**

TABLE I.4 – Précision d'Adaboost M1, BrownBoost, GloutonBoost et AdaBoost Hybride

empiriques d'AdaboostHyb sont toujours meilleurs, mais seule l'amélioration du rappel est suffisamment importante pour être significative. Alors que GloutonBoost obtient des résultats similaies à ceux de BrownBoost, la force d'AdaBoost Hybride face à BrownBoost est donc sa capacité à augmenter le rappel, en particulier sur les jeux de donnés difficiles.

7.2. Comparaison sur des données bruitées

Dans cette partie, nous nous sommes fondés sur l'étude de Dietterich [Dietterich, 1999] en ajoutant du bruit aléatoire aux données. Cet ajout de 20% de bruit est opéré pour chacune des bases, en changeant aléatoirement la valeur de la classe à prédire. Le tableau I.5 nous montre le

comportement des algorithmes vis- à-vis du bruit. Nous remarquons que l'approche Gloutonne ainsi que l'approche hybride sont sensibles elles aussi au bruit puisque le taux d'erreur en généralisation est augmenté pour tous les jeux de données. Cependant cette augmentation n'est que d'environ 8 points en moyenne pour AdaBoost Hybride, GloutonBoost et BrownBoost, alors qu'elle est de 10 points pour AdaBoostM1. En outre, sur les données bruitées, les trois premières méthodes diminuent d'environ 4 points en moyenne le taux d'erreur d'AdaBoostM1, ce qui est significatif pour chaque méthode. Les meilleurs résultats sont obtenus par AdaBoost Hybride, mais son avantage sur BrownBoost et GloutonBoost n'est pas significatif.

On remarquera qu'AdaBoost Hybride surpasse significativement AdaBoostM1 (12 fois sur 15, sur les données bruitées, $p - value = 0.0352$) mais connaît 3 défaites qui se produisent pour les bases CREDIT-A, HEPATITIS et HYPOTHYROID. Nous avons donc étudié de près ces bases de données et nous avons noté un point commun, les valeurs manquantes. En fait, CREDIT-A, HEPATITIS et HYPOTHYROID possèdent respectivement 5%, 6%et $5,4\%$ de valeurs manquantes. Nous constatons ainsi que l'amélioration apportée par l'approche hybride perd de son effet avec l'accumulation de deux types de bruit, les valeurs manquantes et le bruit artificiel. Considérant BrownBoost, nous remarquons qu'il améliore l'erreur en généralisation de toute les bases de données en comparaison avec AdaBoostM1. Cependant, AdaBoost Hybride surpasse BrownBoost 9 fois sur 15, sans que cette supériorité soit significative.

Bases de Données	AdaBoost M1	BrownBoost	GloutonBoost	AdaBoostHyb
IRIS	33.00	**26.00**	28.50	28.00
NHL	45.00	40.00	35.67	**32.00**
VOTE	12.58	**7.00**	7.65	7.76
WEATHER	25.00	22	21.78	**21**
CREDIT-A	22.56	**20.99**	23.14	24.00
TITANIC	34.67	28.08	27.45	**26.98**
DIABETES	36.43	32.12	31.63	**31.20**
HYPOTHYROID	0.92	**0.86**	2.05	2.12
HEPATITIS	31.00	**27.38**	38.95	41.00
CONTACT-LENSES	33	30.60	27.45	**25**
ZOO	18.84	14.56	13.12	**11.20**
STRAIGHT	3.45	**2.79**	2.84	2.81
IDS	2.40	1.02	0.78	**0.50**
LYMPH	28.73	24.57	25.08	**24.05%**
BREAST-CANCER	68.00	50.98	49.59	**48.52**

TABLE I.5 – Erreur en généralisation en données bruités

7.3. Comparaison de la vitesse de convergence

Dans cette partie, on s'intéresse au nombre d'itérations à partir duquel les algorithmes convergent, c'est-à-dire où le taux d'erreur se stabilise. Les résultats de GloutonBoost ne sont pas pris en compte car il converge dans les mêmes intervalles qu'AdaBoost Hybride, seul le

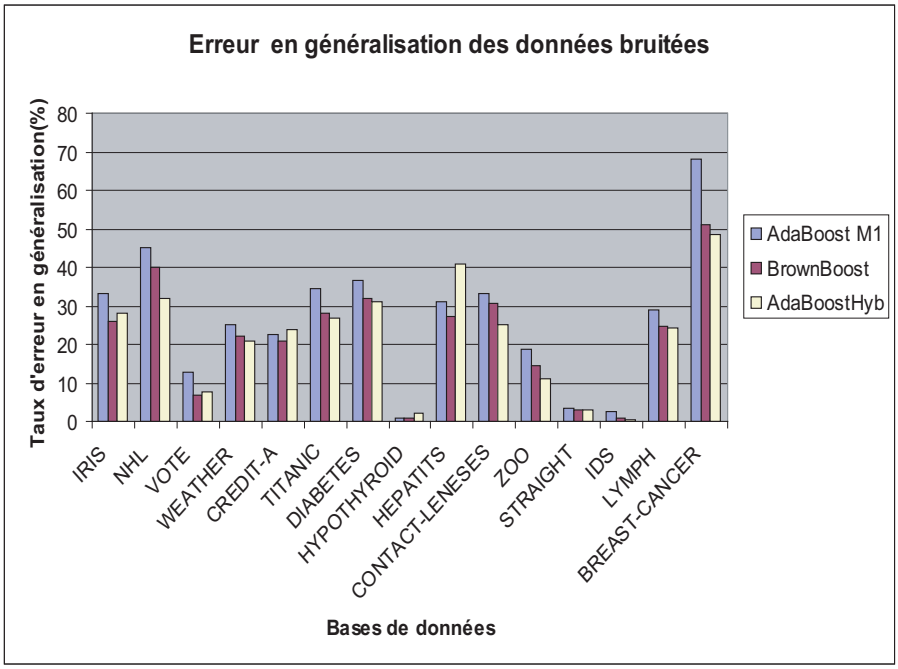

FIGURE I.4 – Comparaison en erreur en généralisation en données bruités

taux d'erreur variant.

Le tableau I.6 nous montre que l'approche hybride permet à AdaBoost de converger plus rapidement. En effet, le taux d'erreur d'AdaBoostM1 ne se stabilise pas même à la $1000e$ itération, alors que AdaBoost Hybride converge à la $20e$ itération ou même avant. C'est pour cette raison que nous avons choisi de pratiquer 20 itérations lors de la première partie de nos expériences.

Ces résultats sont aussi valables pour la base HEPATITIS. En fait, cette base est riche par les valeurs manquantes (taux de 6%). Ces valeurs manquantes présentent toujours un problème de convergence pour les algorithmes d'apprentissage. De plus, ces même résultats ont été obtenus sur des bases de données de type varié (nombreux attributs, classe à prédire comportant plusieurs modalités, taille importante). Ceci nous laisse penser qu'en raison du mode de calcul de l'erreur apparente qui tient compte des hypothèses antérieures, l'algorithme atteint plus rapidement la stabilité.

Finalement, nous remarquons que même après 1000 itérations, la convergence de BrownBoost n'est pas vraiment assurée. Cette remarque confirme le problème de vitesse de convergence de BrownBoost.

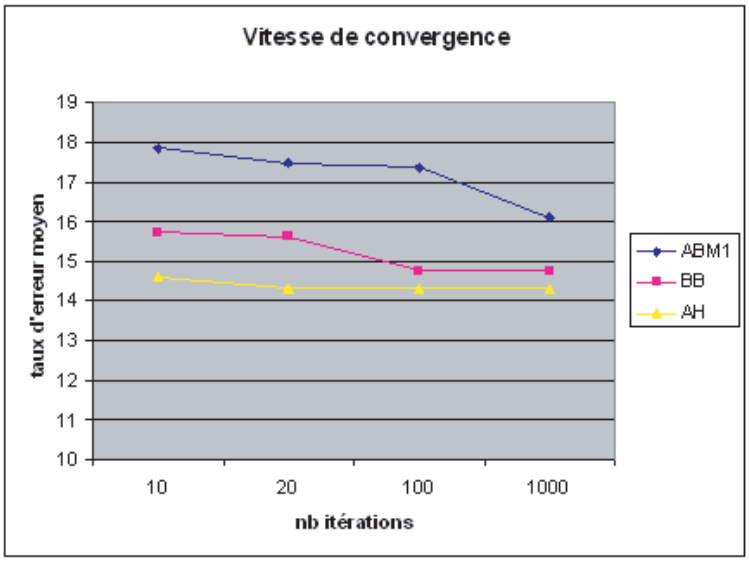

FIGURE I.5 – La vitesse de convergence

-	AdaBoost M1				BrownBoost				AdaBoost hyb			
Nb. iterations	10	20	100	1000	10	20	100	1000	10	20	100	1000
Iris	7,00	6,00	5,90	5,85	3.96	3.89	3,80	3,77	3,50	3,00	3,00	3,00
Nhl	37,00	35,00	34,87	34,55	30,67	30,01	29,89	29,76	31,00	28,00	28,00	28,00
Weather	21,50	21,42	21,40	14,40	21,10	21,00	20,98	21,95	21,03	21,00	21,00	21,00
Credit-A	15,85	15,79	15,75	14,71	13,06	13,00	12,99	12,97	14,00	13,91	13,91	13,91
Titanic	21,00	21,00	21,00	21,00	24,08	24,00	23,89	23,79	21,00	21,00	21,00	21,00
Diabetes	27,70	27,61	27,55	27,54	25,09	25,05	25,03	25,00	25,56	25,56	25,56	25,56
Hypothyroid	0,60	0,51	0,51	0,50	0,62	0,60	0,59	0,55	0,43	0,42	0,42	0,42
Hepatitis	16,12	15,60	14,83	14,19	14,15	14,10	14,08	14,04	14,03	14,00	14,00	14,00
Contact-Lenses	26,30	24,80	24,50	16,33	15,90	15,86	15,83	15,80	16,00	16,00	16,00	16,00
Zoo	7,06	7,00	7,00	7,00	7,25	7,23	7,19	7,15	7,00	6,98	7,00	7,00
Straight	2,50	2,46	2,45	2,42	2,12	2,00	1,98	1,96	0,42	0,42	0,42	0,42
IDS	2,00	1,90	1,88	1,85	0,7	0,67	0,65	0,63	0,7	0,67	0,65	0,63
Lymph	19,53	19,51	19,51	19,50	18,76	18,54	18,50	18,45	18,76	18,54	18,50	18,45
Breast-Cancer	45,89	45,81	45,81	45,79	31,10	31,06	31,04	31,00	31,10	31,06	31,04	31,00

TABLE I.6 – Comparaison de la vitesse de convergence

8. Conclusion et perspectives

Dans ce chapitre, nous nous sommes intéressée au bruit, l'un des problèmes majeurs que posent les données réelles et nous avons proposé une amélioration du *Boosting*, une procédure d'apprentissage supervisé fondée sur l'agrégation de classifieurs. Nous avons choisi cette dernière méthode à cause de sa sensibilité particulière au bruit liée à son caractère adaptatif.

La démarche proposée a abouti à la proposition de deux algorithmes de *boosting* résistant au

bruit, GloutonBoost et surtout son amélioration AdaBoost Hybride, qui reposent sur le lissage des résultats des itérations successives du *Boosting*. Les expérimentations pratiquées sur 15 *benchmarks* de l'UCI ont montré que notre démarche permet d'améliorer significativement les performances de l'algorithme de base du *Boosting*, tant en bonne prédiction et rappel-précision qu'en vitesse de convergence, tout en étant préférable aux versions du *Boosting* résistantes au bruit proposées dans la littérature, en particulier BrownBoost, la plus connue qui est dominée par AdaBoost Hybride en termes de rappel et de vitesse de convergence.

Dans le chapitre suivant, nous nous intéressons à un autre problème ardu que posent les données réelles, celui de la prédiction lorsque la distribution de la variable de classe est très déséquilibrée. La synthèse des résultats de ces deux chapitres en un algorithme opérationnel est notre principale perspective de recherche à la suite de cette thèse.