Traitement de données volumineuses par ensembles d'arbres aléatoires

Pierre Geurts*
Université de Liège
Département d'Électricité, Électronique et Informatique
Service de méthodes stochastiques
Institut Montefiore
Sart Tilman B28
B-4000 Liège Belgique
* Chargé de recherches, FNRS, Belgique
p.geurts@ulg.ac.be
http ://www.montefiore.ulg.ac.be/~geurts

Résumé. Cet article présente une nouvelle méthode d'apprentissage basée sur un ensemble d'arbres de décision. Par opposition à la méthode traditionnelle d'induction, les arbres de l'ensemble sont construits en choisissant les tests durant le développement de manière complètement aléatoire. Cette méthode est comparée aux arbres de décision et au bagging sur plusieurs problèmes de classification. Grâce aux choix aléatoires des tests, les temps de calcul de cet algorithme sont comparables à ceux des arbres traditionnels. Dans le même temps, la méthode se révèle beaucoup plus précise que les arbres et souvent significativement meilleure que le bagging. Ces caractéristiques rendent cette méthode particulièrement adaptée pour le traitement de bases de données volumineuses.

1 Introduction

Actuellement, un des domaines de recherche les plus actifs en apprentissage automatique est l'étude des méthodes basées sur un ensemble de modèles (Dietterich 2000a). La plupart de ces méthodes consistent à se servir d'une méthode d'apprentissage classique pour construire plusieurs modèles et ensuite à agréger les prédictions de ces modèles pour donner une prédiction finale, potentiellement meilleure que les prédictions individuelles. Les méthodes d'ensemble diffèrent essentiellement par la manière dont sont construits les modèles ainsi que par la manière d'agréger leurs prédictions.

Une des catégories de méthodes les plus populaires est constituée par les méthodes de type perturbation et combinaison. Ces méthodes consistent à perturber un algorithme d'apprentissage de manière à ce qu'il fournisse des modèles différents à partir d'un même échantillon d'apprentissage. Les prédictions sont ensuite agrégées par un simple moyennage ou un vote à la majorité dans le cas de la classification. Le représentant le plus populaire de cette famille de méthode est le bagging (Breiman 1996) dans lequel les différents modèles sont obtenus en effectuant un ré-échantillonnage de l'ensemble d'apprentissage avant chaque étape d'induction. Ces méthodes d'ensemble ont beaucoup de succès récemment principalement à cause de l'amélioration de précision qu'elles apportent aux méthodes traditionnelles, telles que les arbres de

décision ou les réseaux de neurones artificiels. Néanmoins, l'utilisation de plusieurs modèles présente l'inconvénient de nécessiter plus d'effort de calcul lors de la phase d'apprentissage.

Dans le contexte des arbres de décision, la plupart des méthodes d'ensemble consistent en fait à perturber l'algorithme de recherche du test optimal en chaque nœud intérieur de l'arbre. Au sein d'une méthode d'ensemble, il est donc nécessaire de dégrader quelque peu l'algorithme d'induction classique de manière à obtenir l'ensemble de modèles. L'idée à la base de l'algorithme proposé dans cet article est d'exploiter cette propriété pour améliorer à la fois la précision et l'efficacité de la méthode d'arbres en l'utilisant au sein d'une méthode d'ensemble. Cette question est particulièrement intéressante dans le contexte du traitement de grands volumes de données où la méthode d'arbres n'a pas toujours la précision espérée et des méthodes d'ensemble telles que le bagging restent malgré tout relativement coûteuses en termes de temps de calcul.

Cet article est structuré comme suit. Dans la section 2, nous donnons une description générale de ce que nous appelons les algorithmes de type perturbation et combinaison (P&C). Plusieurs instances de ce paradigme proposées dans la littérature sont ensuite énumérées. La section 3, après un bref rappel de l'algorithme d'induction d'arbres de décision, passe en revue quelques méthodes P&C spécifiques à cet algorithme. La description de la nouvelle méthode d'ensemble que nous proposons est donnée dans la section 4. Cette méthode consiste à construire des arbres en choisissant les tests aux nœuds internes de l'arbre de manière complètement aléatoire. Dans la section 5, cette méthode est comparée aux arbres de décisions individuels et au bagging sur différents problèmes de classification.

Dans cet article, on se concentrera sur le problème de la classification avec des attributs d'entrée numériques. La généralisation de l'algorithme aux problèmes de la prédiction d'une sortie numérique et au traitement des attributs à valeurs symboliques peut se faire de manière relativement directe mais ne sera pas discutée ici. Dans la suite, on notera par LS l'échantillon d'apprentissage et par $(\underline{a}, c)$ un élément de cet ensemble décrit par son vecteur d'attributs $\underline{a}$ (aussi appelés entrées) et sa classification c (la grandeur de sortie). a_i désignera la i-ième composante du vecteur d'attributs $\underline{a}$.

2 Algorithmes de type perturbation et combinaison

Beaucoup de méthodes d'ensembles proposées dans la littérature satisfont au schéma de perturbation et combinaison (Breiman 1998, 2001) que voici :

- À partir d'un échantillon d'apprentissage LS, pour i allant de 1 à T :
  - On perturbe l'algorithme d'apprentissage (en perturbant les données et/ou les paramètres de la méthode) avec un ensemble de paramètres aléatoires notés ϵ_i tirés aléatoirement selon une distribution $P(\epsilon_i|LS)$.
  - On applique cette perturbation pour obtenir un modèle $f_{\epsilon_i,LS}$, fonction à la fois de l'échantillon LS et des paramètres aléatoires ϵ_i.
- On agrège les modèles obtenus à l'aide de ces perturbations
  - soit en prenant la classe majoritaire parmi les prédictions données par chacun des T modèles de l'ensemble,

- soit en effectuant une moyenne des estimées de probabilités conditionnelles de classes (bien sûr, si le modèle fournit ces estimées) et en renvoyant la classe la plus probable selon cette distribution conditionnelle moyenne.

Le bagging appartient à cette classe de méthode. Dans ce cas, le paramètre ϵ_i décrit l'échantillon d'apprentissage sélectionné par ré-échantillonnage dans l'échantillon de départ. Breiman (2000) a proposé également de perturber l'algorithme d'apprentissage en bruitant la classification des éléments de l'échantillon d'apprentissage. La méthode "random subspace" (Ho 1998) consiste à sélectionner un sous-ensemble aléatoire des attributs d'entrée avant de construire chaque modèle de l'ensemble. Alors que ces méthodes sont indépendantes de l'algorithme d'apprentissage, il existe également plusieurs variantes spécifiques qui consistent à perturber directement un algorithme d'apprentissage sans toucher aux données. Nous nous intéressons dans la section suivante aux variantes spécifiques aux arbres de décision.

3 Arbres de décision et méthode P&C dédiées

La méthode d'induction d'arbres de décision (Breiman et al. 1984) est une méthode d'apprentissage très populaire. C'est une des méthodes d'apprentissage les plus efficaces et elle fournit également des résultats interprétables sous la forme de règles logiques. Néanmoins, elle se caractérise par une variance très importante (Geurts 2002) qui l'empêche de concurrencer d'autres algorithmes du point de vue précision. Cette grande variance et sa grande efficacité ont fait de cette méthode la méthode de choix pour le développement de méthodes P&C. En effet, du fait d'une variance importante, l'amélioration de la précision à l'aide des méthodes d'ensemble est souvent très impressionnante. D'autre part, la construction de plusieurs modèles n'est pas rebutante du fait de l'efficacité de la méthode. C'est pourquoi il est logique de trouver dans la littérature un grand nombre de méthodes d'ensemble spécifiques à cet algorithme. Avant d'en décrire certaines, rappelons d'abord brièvement le principe de l'induction d'arbres de décision (pour plus de détails, voir par exemple (Zighed 2000)).

3.1 Algorithme d'induction d'arbres de décision

L'algorithme d'induction d'arbre consiste à diviser de manière récursive l'échantillon d'apprentissage en sous-ensembles d'éléments de même classe au moyen de tests (souvent binaires) basés sur les valeurs des attributs. Une mesure de score est définie pour évaluer la capacité d'un test à séparer l'échantillon en sous-échantillons aussi purs que possible en termes de classification. L'algorithme démarre avec l'échantillon d'apprentissage complet et recherche, pour chaque attribut d'entrée, le test qui maximise le score. Ensuite, globalement, on choisit l'attribut dont le test a le score le plus élevé. On sépare l'échantillon d'apprentissage à l'aide de ce test et on procède de la même manière pour séparer les sous-ensembles correspondant aux issues du test. L'algorithme s'arrête lorsque les sous-échantillons sont suffisamment purs.

Pour les attributs à valeurs numériques, on utilise généralement des tests binaires de la forme $[A < a_{th}]$ qui comparent la valeur de l'attribut A à un seuil a_{th}, appelé seuil de discrétisation. L'algorithme de recherche du meilleur test consiste donc à trouver

la valeur du seuil qui maximise le score. Généralement, les mesures de score ne sont sensibles qu'aux proportions d'objets de chaque classe et donc, la recherche du meilleur seuil se fera parmi au plus N seuils possibles pour un échantillon de taille N et demande le tri de l'échantillon selon la valeur de l'attribut.

3.2 Méhodes P&C spécifiques

Plusieurs méthodes d'ensemble perturbent directement cet algorithme pour générer les arbres de l'ensemble . Par exemple, Ali et Pazzani (1996) remplacent le choix du meilleur test par un tir aléatoire d'un test parmi ceux qui ont les plus hauts scores. Pour ce faire, si S^* est le score du meilleur test, un test est choisi aléatoirement, avec un probabilité proportionnelle à son score, parmi les tests qui présentent un score supérieur à $S^* - \beta.S^*$, où β est un paramètre constant compris entre 0 et 1. Dietterich (2000b) propose une approche très semblable qui consiste à choisir un test aléatoirement parmi les 20 meilleurs tests. Ho (1998) construit des arbres en choisissant localement, à chaque nouvelle recherche d'un meilleur test, un sous-ensemble des attributs. Breiman (2001) combine cette dernière idée avec le bagging dans son algorithme "random forests". Toutes ces méthodes ont montré des résultats tantôt comparables tantôt significativement meilleurs que ceux obtenus par le bagging.

4 Arbres extrêmement aléatoires

Une caractéristique commune aux méthodes d'ensemble décrites dans la section précédente est qu'elles consistent toutes à choisir un test lors de la construction de l'arbre qui n'est plus le test optimal, c'est-à-dire celui qui maximise le score. Une conclusion que l'on peut donc tirer du bon fonctionnement de ces algorithmes est le fait qu'il est possible de dégrader dans une certaine mesure la méthode d'arbres par rapport à l'algorithme classique qui recherche localement le test optimal et néanmoins améliorer la précision en agrégeant plusieurs de ces modèles "sous-optimaux".

Sur base de cette analyse, l'idée de l'algorithme qui est proposé ici est de perturber encore d'avantage l'algorithme d'induction d'arbres par rapport aux algorithmes existants de manière, d'une part, à améliorer la précision, et d'autre part, à profiter de ce degré de liberté pour améliorer l'efficacité de l'algorithme. Une description précise de l'algorithme proposé est donnée dans la sous-section suivante. Sa complexité en temps est discutée dans la section 4.2.

4.1 Algorithme proposé

Même si elles s'écartent de l'algorithme d'induction classique, les variantes décrites dans la section 3.2 demandent toujours la détermination des meilleurs tests pour chacun des attributs (pour les variantes de Ali et Pazzani et Dietterich) ou au mieux pour un nombre fixé d'entre eux (dans les variantes de Ho et Breiman). Or, on peut se poser la question de savoir si cette recherche est vraiment nécessaire dans le contexte des méthodes d'ensemble. Cette question est d'autant plus légitime que nous avons mis

en évidence dans (Geurts 2002) la très grande variabilité des paramètres qui servent à définir les tests d'un arbre avec des variations de l'échantillon d'apprentissage.

La méthode proposée dans cet article consiste à aller un pas plus loin que les autres méthodes d'ensemble en choisissant les tests de manière complètement aléatoire. L'algorithme de construction d'arbres extrêmement aléatoires, qu'on appellera "extra-trees" (pour "extremely randomized trees"), est décrit dans la table 1. Lors du développement d'un nœud, un attribut et un seuil de discrétisation sont sélectionnés aléatoirement. Pour éviter de choisir de trop mauvais tests, le choix de l'attribut est rejeté si le score du test est inférieur à un seuil donné (fixé par l'utilisateur).

Pour choisir le seuil de discrétisation, nous avons testé deux variantes décrites au bas de la table 1. La première consiste à tirer aléatoirement le seuil à partir d'une distribution Gaussienne de moyenne et d'écart-type donnés par les valeurs correspondantes empiriques estimées dans l'échantillon courant. Un des inconvénients de cette variante est qu'il reste nécessaire de calculer la moyenne et l'écart-type dans l'échantillon d'apprentissage courant à chaque étape de la construction. La deuxième variante court-circuite cette étape en utilisant comme valeur de seuil la valeur de l'attribut pour un objet sélectionné aléatoirement dans l'échantillon d'apprentissage.

La nouvelle méthode d'ensemble que nous proposons consiste donc à construire plusieurs arbres à l'aide de l'algorithme de la table 1 à partir de l'échantillon d'apprentissage. Leurs prédictions sont ensuite agrégées en moyennant les distributions conditionnelles estimées aux nœuds terminaux des arbres. Bien que cet algorithme paraisse naïf a priori, il construit néanmoins des arbres qui classent parfaitement l'ensemble d'apprentissage (bien sûr dans la mesure où les attributs le permettent). De plus, la dépendance des tests à l'échantillon devrait être fortement réduite par rapport au choix du test optimal. La variance introduite par le choix complètement aléatoire des tests devrait être compensée par le fait que plusieurs modèles sont agrégés. C'est pourquoi, globalement, on peut espérer une amélioration importante de la précision.

4.2 Complexité

Dans la première variante, la recherche d'un test aléatoire pour séparer un sous-échantillon de taille N demande le calcul de la moyenne et de l'écart-type pour un attribut tiré aléatoirement et est donc d'ordre N. Dans la seconde variante, cette recherche est d'ordre 1 puisqu'il suffit de choisir un objet au hasard parmi les N. Si le seuil sur le score est suffisamment faible, la plupart du temps, un très petit nombre seulement d'attributs devront être considérés et on peut donc dire que la complexité de l'algorithme est indépendant du nombre d'attributs. Une fois le test trouvé, il faudra encore de l'ordre de N opérations pour séparer l'échantillon en deux parties et cela dans les deux variantes. Cette opération doit être répétée à tous les nœuds de l'arbre. Si on suppose que l'arbre est relativement équilibré, sa profondeur qui est limitée par la taille de LS est d'ordre $\log(N)$. Or, pour augmenter la profondeur de un niveau, il faut considérer des sous-échantillons dont la somme totale des tailles vaut N. Donc, sous l'hypothèse d'arbres équilibrés, l'algorithme de construction d'arbres aléatoires est d'ordre $N.\log(N)$ et la construction de T arbres aléatoires demande de l'ordre de $T.N.\log(N)$ opérations. À complexité égale des arbres, le nombre d'opérations sera néanmoins plus faible dans le cas de la seconde variante.

Pour la construction d'un extra-tree :
- On crée un nœud $\mathcal{N}$ et on lui attache l'échantillon complet $ls(\mathcal{N}) = LS$;
- On initialise $L = \{\mathcal{N}\}$, la liste des nœuds ouverts.
- Tant que L n'est pas vide :
  1. On choisit et on enlève un nœud $\mathcal{N}$ de L ; on initialise L_a à la liste de tous les attributs.
  2. Si l'ensemble $ls(\mathcal{N})$ contient des éléments tous de la même classe C ou si tous les attributs sont constants dans l'ensemble, alors on attache une prédiction au nœud $\mathcal{N}$ qui devient un nœud terminal de l'arbre.
  3. Sinon, on choisit aléatoirement un attribut A_i qu'on retire de L_a.
  4. On tire aléatoirement un seuil de discrétisation a_{th} pour cet attribut.
  5. Si le score du test $[A_i < a_{th}]$ sur l'échantillon $ls(\mathcal{N})$ est inférieur au seuil s_{th} et si la liste L_a est non vide, on retourne à l'étape 3.
  6. Sinon, on note $[A_* < a^*_{th}]$ le meilleur test parmi ceux évalués jusque là et on calcule les sous-échantillons $ls_l = \{(\underline{a}, c) \in ls(\mathcal{N}) | a_* < a^*_{th}\}$ et $ls_r = \{(\underline{a}, c) \in ls(\mathcal{N}) | a_* \geq a^*_{th}\}$;
  7. On crée deux nœuds $\mathcal{N}_l$ et $\mathcal{N}_r$ qu'on insère dans L et on fixe $ls(\mathcal{N}_l) = ls_l$ et $ls(\mathcal{N}_r) = ls_r$;

Deux variantes pour le choix d'un seuil (étape 4) :
1. On calcule la moyenne empirique, $\overline{A_i}(ls(\mathcal{N}))$, et l'écart-type $\sigma_{A_i}(ls(\mathcal{N}))$ de cet attribut dans l'échantillon $ls(\mathcal{N})$ et on tire aléatoirement le seuil a_{th} selon la distribution $N(\overline{A_i}(ls(\mathcal{N})), \sigma_{A_i}(ls(\mathcal{N})))$.
2. On tire aléatoirement un objet $(\underline{a}, c)$ dans $ls(\mathcal{N})$ et on prend $a_{th} = a_i$.

TAB. 1 – Algorithme d'induction d'un extra-tree à partir d'un échantillon LS

Lors du développement d'un nœud dans l'algorithme d'arbres classique, la recherche du meilleur test demande, pour chaque attribut, le tri de l'échantillon et le calcul du score pour, au plus, N seuils différents. Ce tri peut se faire une fois pour toute avant de construire l'arbre et ensuite un tri par projection, linéaire, peut être utilisé pour la recherche des seuils de discrétisation. Néanmoins, dans le cadre du traitement de grands volumes de données, nous n'avons pas envisagé cette variante qui requiert le stockage de M vecteurs contenant l'ensemble d'apprentissage. Ainsi, l'algorithme de recherche du meilleur test demande de l'ordre de $M.N.\log(N)$ opérations. On devrait donc gagner, avec les extra-trees, un facteur $M.\log(N)$ pour le développement d'un nœud. Bien sûr, le gain en temps global dépendra de la complexité finale des arbres (fonction à la fois de la taille de la base de données, de l'algorithme et du problème).

En dehors des arbres qui peuvent être stockés au fur et à mesure sur disque, la méthode d'extra-trees se contente également d'un espace mémoire qui correspond à un vecteur d'index de la taille de l'échantillon d'apprentissage. Étant donné notre implémentation de la recherche du meilleur test, la demande en mémoire vive est identique pour la méthode d'arbres et le bagging, le même vecteur d'index étant trié au fur et à mesure de la construction.

En ce qui concerne le test, l'utilisation d'un ensemble d'arbres de décision demande la propagation du cas de test dans tous les arbres de l'ensemble. En moyenne, la profondeur des arbres sera d'ordre $\log(N)$. Il faudra donc de l'ordre de $T.\log(N)$ comparaisons pour faire une prédictions avec un ensemble de T arbres de décision. Donc, même si les arbres construits par l'algorithme complètement aléatoires sont très complexes en termes de nombre de nœuds, leur utilisation restera très efficace même pour des tailles très importantes d'échantillons d'apprentissage.

5 Expérimentations

Pour valider l'algorithme proposé dans la section précédente, nous avons utilisé 9 problèmes de classification disponibles pour la plupart publiquement (Black et Merz 1998). Nous les avons choisis principalement pour leur grande taille et parce qu'ils ne comportaient que des attributs numériques. Nos expérimentations sur ces problèmes auront deux buts. D'abord, valider la méthode proposée en termes de précision, ensuite évaluer son efficacité. Dans les deux cas, nous avons comparé les deux variantes à l'algorithme d'induction d'arbres classique et au bagging. Avant de décrire ces expérimentations, la sous-section suivante traite de l'implémentation pratique de ces trois algorithmes et du protocole d'expérimentation.

5.1 Choix des paramètres et protocole d'expérimentation

La mesure de score que nous utilisons pour développer les arbres est la mesure proposée par Wehenkel dans (Wehenkel 1998) qui est une normalisation particulière de l'information mutuelle de Shannon. Que ce soit pour les arbres classiques, pour le bagging ou pour les arbres extrêmement aléatoires, nous n'avons pas utilisé de critère d'arrêt ou d'élagage. Ainsi, les arbres sont toujours développés le plus profondément possible de manière à obtenir des nœuds terminaux purs en termes de classification.

Pour les méthodes d'ensemble, plus on agrège de modèles, plus le taux d'erreur diminue. Que ce soit pour le bagging ou notre méthode, nous nous sommes arrêté à $T = 50$ modèles. Sur la plupart des problèmes, le gain obtenu en augmentant encore le nombre de modèles est négligeable. Pour les arbres extrêmement aléatoires, nous avons utilisé dans tous nos tests un seuil s_{th} égal à 0.1. La méthode n'est pas très sensible à la valeur de ce paramètre mais il est néanmoins nécessaire de ne pas prendre une valeur nulle.

Pour chaque problème, nous avons divisé aléatoirement la base de données en deux échantillons d'apprentissage et de test de tailles données dans la table 5.1 (respectivement $\#LS$ et $\#TS$). L'expérience est répétée 10 fois sur chaque problème. La table 5.1 reprend les moyenne et écart-type des statistiques estimées sur ces 10 essais. La première colonne est le taux d'erreur, la deuxième colonne la complexité (le nombre total de nœuds de l'arbre ou de l'ensemble d'arbres) et les deux dernières colonnes reprennent les temps de calcul (en msec) respectivement pour la construction et le test des modèles [1]. Ces résultats sont analysés dans les sous-sections suivantes.

5.2 Précision

Quelle que soit la variante, les extra-trees améliorent de manière très significative la précision par rapport aux arbres de décision. À une ou deux exception près, ils sont également systématiquement meilleurs que le bagging. L'amélioration n'est pas significative sur les problèmes Waveform, Two-norm, Satellite et Isolet. Par contre, elle est importante sur les quatre autres problèmes, Pendigits, Dig44, Letter et Mnist (pour la variante 1 sur ce dernier problème). Notez qu'on peut encore diminuer le taux d'erreur sur certains problèmes en augmentant le nombre d'arbres de l'ensemble. Ainsi, par exemple sur la base de données isolet, si on construit 200 arbres extrêmement aléatoires avec la première variante, on passe à 6.42% d'erreurs en moyenne.

Les deux variantes sont presque équivalentes en termes de taux d'erreur. Cependant, la première variante est toujours meilleure (excepté sur Satellite) et produit aussi des arbres moins complexes que la seconde. Sur deux problèmes, la seconde variante est également moins bonne que le bagging.

5.3 Efficacité

Comme attendu, le bagging multiplie les temps de calcul de la méthode d'arbres par un facteur proche de 50. Les deux variantes d'extra-trees permettent de réduire les temps de calcul par rapport au bagging de manière très importante. Sur plusieurs problèmes, elles offrent même des résultats comparables (sur Waveform, Two-norm) ou meilleurs (sur Dig44, Isolet[2] et Mnist) que ceux de la méthode d'arbres classique et cela malgré le fait que 50 modèles doivent être contruits et que ces modèles sont très complexes. La seconde variante d'extra-trees est meilleure que la première malgré le

[1] Le programme est écrit en C sous linux et tourne sur un pentium 4 2.4GHz avec 1Go de mémoire vive. Dans nos expérimentations, les données et les modèles sont stockés en mémoire.

[2] Sur isolet, les attributs sont en très grand nombre et de plus à valeurs réelles. Le nombre de valeurs possibles de seuils de discrétisation à considérer par la méthode d'arbres est donc particulièrement important et expliquent les temps de calcul élevés de cette méthode.

Méthode	Erreur (%)	Complexité	Construction (msec)	Test (msec)
Waveform (#attributs=21, #classes=3, $\#LS = 4000$, $\#TS = 1000$)				
Arbre	25.04 ± 1.00	835 ± 20	1137 ± 36	1 ± 0
Bagging	16.67 ± 1.11	29885 ± 116	43290 ± 420	34 ± 1
Extra-trees 1	15.58 ± 0.93	127408 ± 872	2255 ± 42	45 ± 0
Extra-trees 2	15.56 ± 1.00	128058 ± 768	1588 ± 27	45 ± 1
Two-norm (#attributs=20, #classes=2, $\#LS = 8000$, $\#TS = 2000$)				
Arbre	14.79 ± 0.63	1011 ± 19	4097 ± 442	3 ± 0
Bagging	3.38 ± 0.39	36493 ± 199	141714 ± 3566	129 ± 3
Extra trees 1	3.31 ± 0.40	157424 ± 725	6503 ± 444	127 ± 3
Extra trees 2	3.68 ± 0.43	158972 ± 805	4518 ± 364	128 ± 3
Satellite (#attributs=36, #classes=6, $\#LS = 4435$, $\#TS = 2000$)				
Arbre	14.67 ± 0.485	733 ± 19	971 ± 38	2 ± 0
Bagging	9.51 ± 0.50	26361 ± 305	41166 ± 634	99 ± 3
Extra trees 1	9.19 ± 0.39	113993 ± 1106	1989 ± 37	120 ± 3
Extra trees 2	9.04 ± 0.54	121274 ± 889	1478 ± 43	122 ± 2
Pendigits (#attributs=16, #classes=10, $\#LS = 7494$, $\#TS = 3498$)				
Arbre	3.67 ± 0.38	487 ± 23	624 ± 19	3 ± 0
Bagging	1.8 ± 0.19	19734 ± 340	26408 ± 390	148 ± 2
Extra trees 1	0.85 ± 0.15	133368 ± 1626	1438 ± 17	199 ± 1
Extra trees 2	1.11 ± 0.19	171867 ± 1712	1330 ± 12	215 ± 2
Dig44 (#attributs=16, #classes=10, $\#LS = 14000$, $\#TS = 4000$)				
Arbre	13.18 ± 0.56	2301 ± 27	5370 ± 261	6 ± 0
Bagging	7.51 ± 0.41	84561 ± 697	190284 ± 8438	272 ± 12
Extra trees 1	4.34 ± 0.34	454173 ± 2609	4286 ± 153	352 ± 16
Extra trees 2	4.74 ± 0.46	527242 ± 3666	4019 ± 190	370 ± 13
Letter (#attributs=16, #classes=26, $\#LS = 16000$, $\#TS = 4000$)				
Arbre	11.66 ± 0.54	3705 ± 35	1290 ± 49	7 ± 1
Bagging	6.53 ± 0.50	141583 ± 773	57790 ± 2881	331 ± 21
Extra trees 1	3.87 ± 0.33	558427 ± 2300	6258 ± 413	451 ± 22
Extra trees 2	4.06 ± 0.32	582504 ± 2563	5786 ± 602	483 ± 64
Isolet (#attributs=617, #classes=26, $\#LS = 6238$, $\#TS = 1559$)				
Arbre	17.95 ± 0.79	938 ± 11	139110 ± 15580	3 ± 0
Bagging	8.7 ± 0.40	35232 ± 433.74	4548377 ± 168446	134 ± 8
Extra trees 1	8.6 ± 0.79	387438 ± 1348	4243 ± 467	253 ± 26
Extra trees 2	9.50 ± 0.73	422330 ± 3092	3956 ± 439	254 ± 33
Mnist (#attributs=784, #classes=10, $\#LS = 60000$, $\#TS = 10000$)				
Arbre	11.76 ± 0.20	7137 ± 65	582114 ± 58219	46 ± 14
Bagging	4.5 ± 0.26	263414 ± 556	24346159 ± 675316	1905 ± 112
Extra trees 1	3.32 ± 0.18	1755363 ± 4526	643615 ± 89725	2455 ± 366
Extra trees 2	4.47 ± 0.14	2149476 ± 7029	498868 ± 42266	2274 ± 249

TAB. 2 – Résultats des expérimentations

fait qu'elle produise des arbres un peu plus complexes. Néanmoins, l'amélioration n'est pas très importante. Notons enfin qu'on peut accélérer les deux variantes en réduisant le nombre de modèles, souvent sans diminuer significativement les taux d'erreur.

Même si les modèles construits par la méthode d'arbres aléatoires sont très complexes, leur utilisation reste extrêmement efficace. Sur aucun problème, il ne faut plus d'un millième de seconde pour faire une prédiction. Il est également à noter que l'augmentation du temps de test par rapport au bagging n'est pas proportionelle à l'augmentation de la complexité des arbres et que souvent, les temps de calcul sont comparables, voire meilleurs pour les extra-trees.

5.4 Discussion

Pour un même effort de calcul et une même utilisation de la mémoire, on a donc avec les extra-trees une méthode beaucoup plus précise que les arbres de décision. Entre les deux variantes, on préferrera la première qui, même si elle est légèrement moins rapide, produit des modèles moins complexes et de meilleure précision.

Il est cependant important de faire quelques remarques en ce qui concerne la manière dont ont été faites les expérimentations qui nous permettent de tirer ces conclusions :

- Nous n'avons pas utilisé d'élagage qui pourrait diminuer les taux d'erreur de manière significative par rapport aux arbres complets.
- Les temps de calcul de la méthode d'arbres pourraient être fortement diminués en triant a priori la base de données. De même, les algorithmes de bagging et des extra-trees peuvent être accélérés par une parallélisation de la construction des modèles (très simple à implémenter vu l'indépendance de ces modèles).
- La mesure de score utilisée est basée sur le calcul de logarithmes. Les temps de calcul pourraient être améliorés en utilisant une mesure de score, telle une entropie gini, qui n'utilise pas ces logarithmes. Bien sûr, la méthode d'arbres classique, qui calcule de nombreux scores, profiterait d'avantage de cette amélioration.

Néanmoins, nous ne pensons pas que ces améliorations changeraient autrement que quantitativement les résultats obtenus.

6 Conclusions et perspectives

Nous avons proposé dans cet article une nouvelle méthode d'ensemble basée sur les arbres de décision. Cette méthode consiste à construire des arbres en choisissant les tests de manière complètement aléatoire. Sur nos problèmes de test, elle s'est révélée tout aussi rapide mais beaucoup plus précise que la méthode d'arbres de décision. Elle est aussi beaucoup plus rapide et souvent plus précise que le bagging. De notre point de vue, cette méthode constitue une solution clé sur porte (dans la mesure où il n'y a qu'un seul paramètre à régler auquel la méthode est très peu sensible) particulièrement intéressante pour le traitement de grands volumes de données. Une autre contribution de cet article est de montrer, par le caractère extrême de l'algorithme qui est proposé, qu'il est possible de prendre de nombreuses libertés par rapport à la méthode d'arbres classique pour la conception d'une méthode d'ensemble. Ainsi, d'autres variantes peuvent être imaginées à côté des deux variantes proposées ici qui pourraient

être meilleures sur le plan de la précision et/ou de l'efficacité.

À côté du traitement de grandes volumes de données, un autre type d'applications où notre méthode pourrait se révéler particulièrement intéressante est l'apprentissage incrémental ou adaptatif. Dans ce type d'applications, on demande à l'algorithme d'être capable de traiter des données qui arrivent au fur et à mesure et éventuellement une classification qui évolue au cours du temps. Étant donné que la dépendance des tests à l'échantillon est très faible dans un arbre aléatoire, la mise à jour d'un modèle à l'aide de nouvelles données devrait pouvoir être faite de manière très efficace.

D'un point de vue plus théorique, la question du bon fonctionnement des méthodes d'ensemble est toujours une question ouverte en apprentissage que nous n'avons pas abordée dans cet article. Dans (Geurts 2002), nous effectuons une étude des méthodes d'ensemble du point de vue du biais et de la variance qui montrent que ces méthodes agissent essentiellement sur la variance des arbres. D'autres analyses sont données par exemple dans (Dietterich 2000b) ou (Breiman 2001) qui toutes deux relient l'amélioration de la précision à la non-corrélation entre les prédictions des modèles de l'ensemble.

Enfin, il existe également d'autres techniques d'apprentissage récentes qui possèdent des caractéristiques communes avec la méthode proposée ici (efficacité, traitement d'un grand nombre de variables d'entrée, robustesse, généralité...) et auxquelles il serait intéressant de la comparer. Par exemple, le boosting (voir (Hastie et al. 2001) pour les variantes récentes) est une autre famille de méthodes d'ensemble dans laquelle les modèles sont construits de manière séquentielle en modifiant les données en fonction des modèles déjà construits. Dans plusieurs études comparatives, le boosting s'est souvent révélé significativement meilleur que le bagging (par exemple (Bauer et Kohavi 1999)). Les machines à support vectoriel (Cristianini et Shawe-Taylor 2000) sont capables de résultats très impressionnants en termes de précision sur beaucoup de problèmes et il existe également des implémentations très efficaces de cet algorithme (Dong et al.).

Références

Ali K. et Pazzani M. (1996), Error Reduction through Learning Multiple Descriptions, Machine Learning, 24 (3), 1996, pp 173-206.

Bauer E. et Kohavi R. (1999), An Empirical Comparison of Voting Classification Algorithms : Bagging, Boosting, and Variants, Machine Learning, 36, 1999, pp 105-139.

Blake C. et Merz C. (1998), UCI Repository of machine learning databases, 1998, http ://www.ics.uci.edu/~mlearn/MLRepository.html.

Breiman L., Friedman J., Olsen R. et Stone C. (1984), Classification and Regression Trees, Wadsworth International (California), 1984.

Breiman L. (1996), Bagging Predictors, Machine Learning, 24(2), 1996, pp 123-140.

Breiman L. (1998), Arcing classifiers, Annals of statistics, 26, 1998, pp 801-849.

Breiman L. (2000), Randomizing outputs to increase prediction accuracy, Machine Learning, 40(3), 2000, pp 229-242.

Breiman L. (2001), Random forests, Machine learning, 45, 2001, pp 5-32.

Cristianini C. et Shawe-Taylor J. (2000), An introduction to support vector machines, MIT Press, Cambridge, MA, 2000.

Dietterich T. G. (2000a), Ensemble Methods in Machine Learning, Proceedings of the first International Workshop on Multiple Classifier Systems, 2000.

Dietterich T. G. (2000b), An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees : Bagging, Boosting, and Randomization, Machine Learning, 40(2), 2000, pp 139-157.

Dong J.-X., Krzyzak A. et Suen C. (2002), A fast SVM training algorithm, Springer, International workshop on pattern recognition with SVM, August 2002, pp 53-67.

Geurts P. (2002), Contributions to decision tree induction : bias/variance tradeoff and time series classification, PhD thesis, Université de Liège, mai 2002.

Hastie T., Tibshirani R. et Friedman J. (2001), The elements of statistical learning : data mining, inference and prediction, Springer, 2001.

Ho T. K. (1998), The Random Subspace Method for Constructing Decision Forests, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (8), 1998, pp 832-844.

Wehenkel L. (1998), Automatic learning techniques in power systems, Kluwer Academic, Boston, 1998.

Zighed D. et Rakotomalala R. (2000), Graphes d'Induction : Apprentissage et Data Mining, Editions Hermes, 2000.

Summary

This paper presents a new learning algorithm based on decision tree ensembles. In opposition to the classical decision tree induction method, the trees of the ensemble are built by selecting the tests during the induction fully at random. This method is compared to single decision trees and to bagging on several classification problems. Because of the extreme randomization, the computation times of this algorithm are comparable to those of the traditional tree induction method. At the same time, this method is much more accurate than single trees and often significantly better than bagging. These properties make this algorithm especially well adapted to handle very large databases.