Forêts Aléatoires: De l'Analyse des Mécanismes de Fonctionnement à la Construction Dynamique

Simon Bernard

Laboratoire d'Informatique,
de Traitement de l'Information et des Systèmes
Université de Rouen

Thèse en vue de l'obtention du titre de
Docteur en Informatique de l'Université de Rouen

Forêts Aléatoires : De l'Analyse des Mécanismes de Fonctionnement à la Construction Dynamique

Simon Bernard

Présenté le 02 décembre 2009 devant le jury composé de :

Marc Sebban	-	Université Jean Monnet, Saint-Étienne	-	Président
Thierry Artières	-	Université Pierre et Marie Curie, Paris	-	Rapporteur
Yves Grandvalet	-	Université de Technologie de Compiègne	-	Rapporteur
Louis Wehenkel	-	Institut Montefiore de Liège	-	Examinateur
Laurent Heutte	-	Université de Rouen	-	Directeur
Sébastien Adam	-	Université de Rouen	-	Encadrant

Remerciements

Avant tout, j'aimerais exprimer mes plus sincères remerciements à Laurent Heutte et à Sébastien Adam. J'ai pu au cours de ces trois années m'enrichir de leurs grandes qualités scientifiques et humaines, et pour cela je leur suis très reconnaissant. Plus particulièrement, j'aimerais les remercier de m'avoir accordé autant de temps et d'énergie, et surtout de m'avoir si vite considéré comme un chercheur à part entière au delà du doctorant que j'étais.

Je souhaiterais ensuite remercier les membres de mon jury : Thierry Artières et Yves Grandvalet pour avoir rapporté ces travaux de thèse ; Marc Sebban et Louis Wehenkel pour avoir accepté d'en être examinateur. Je les remercie tous pour le temps qu'il m'ont accordé et pour leurs retours très enrichissants.

Ensuite, je remercie Jean-Marc Ogier, Professeur au laboratoire L3i de La Rochelle, pour m'avoir initié à la recherche scientifique. Il a joué un rôle important dans ma décision de poursuivre dans cette voie et a notamment permis ma venue au LITIS. J'associe par ailleurs à ces remerciements Karell Bertet et Stéphanie Guillas qui l'ont accompagné dans l'encadrement de mon stage de Master à La Rochelle. J'ai beaucoup appris à leur contact et je leur en suis reconnaissant.

J'aimerais à présent remercier tous les membres du laboratoire LITIS qui m'ont soutenu et accompagné au cours de ces trois années. Je remercie notamment mes collègues et voisins de bureau pour avoir contribué à entretenir cette ambiance de travaille si plaisante. Je ne peux tous les citer ici mais j'espère qu'ils se reconnaitront, qu'ils soient toujours membres du LITIS ou qu'ils aient depuis changé d'horizon.

Je me dois de remercier plus particulièrement Laurence Savouray et Fabienne Bocquet, nos deux secrétaires de choc et de charme. Les raisons que j'ai de les remercier sont trop nombreuses pour que je puisse toutes les énnoncer ici, mais elles savent je l'espère qu'elles ont joué un rôle aussi important dans ma thèse qu'elles sont indispensable à la vie du laboratoire.

Je voudrais ensuite remercier mes parents pour leur soutiens indéfectible. Ils m'ont été précieux à chaque étape de mes études et plus particulièrement de cette thèse. Je remercie également tous mes amis qui m'ont accompagné dans les moments de détentes, que ce soit à La Rochelle, à Rouen ou à Paris. Et pour finir j'adresse une "spéciale dédicasse" toute particulière à Gérard Majax en souvenir de notre rencontre et à Jean-Jacques Goldman pour l'ensemble de son œuvre.

Table des matières

Introduction Générale

Marvin Minsky, scientifique américain du MIT et l'un des plus grand chercheurs dans le domaine des sciences cognitives et de l'intelligence artificielle, écrivait en 1991 à propos de l'apprentissage automatique[1] :

> *"Pour résoudre des problèmes vraiment difficiles, nous aurons besoin d'utiliser plusieurs représentations différentes... il est temps d'arrêter d'argumenter sur quel type de technique de classification est la meilleure... Nous devrions plutôt travailler à un plus haut niveau d'organisation et découvrir comment construire des systèmes managériaux pour exploiter les différents avantages et contourner les différentes limitations de chacune de ces techniques."*

Ce que Minsky a exprimé il y a maintenant presque une vingtaine d'années, c'est le fait que l'on disposait déjà à l'époque d'une très grande variété de méthodes d'apprentissage automatique, sans qu'aucune d'elles toutefois ne parvienne seule à résoudre tous les problèmes d'apprentissage. La piste qu'il a proposé de suivre, les chercheurs du domaine avaient déjà commencé à y travailler en montrant que la combinaison de différentes méthodes d'apprentissage/classification permettait de tirer avantage de leurs forces tout en contournant leurs faiblesses. Aujourd'hui, force est de constater que ce qu'on appelle maintenant **systèmes multi-classifieurs** (désignés souvent par l'acronyme **MCS** pour *Multiple Classifier Systems*) constitue une des voies les plus prometteuses de l'apprentissage automatique.

En particulier, les **Ensemble de Classifieurs**, ou en anglais *Ensemble of Classifiers* (EoC), une des approches multi-classifieurs les plus populaires et les plus efficaces qui consiste à combiner un ensemble de classifieurs de même type (par exemple un ensemble de réseaux de neurones, un ensemble d'arbres de décision, ou un ensemble de discriminants), ont fait l'objet de nombreux travaux et il existe aujourd'hui un grand nombre de méthodes capables de générer automatiquement des ensembles de classifieurs : Bagging, Boosting, Random Subspaces, ECOC, pour ne citer que les plus courantes, sont autant de méthodes différentes dont l'objectif commun est de créer de la diversité au sein d'un ensemble de classifieurs performants tout en cherchant à établir le meilleur consensus possible entre ces classifieurs. Cependant, il faut bien avouer que notre compréhension de cette notion de diversité dans les ensembles est encore à ce jour très sommaire et les procédés pour parvenir à produire cette diversité tant désirée encore assez mal maîtrisés. De sorte que, si chacune de ces procédures d'induction d'EoC dispose d'hyperparamètres pour le contrôle de la construction des EoC, il n'est pas toujours évident d'en règler les valeurs. Par exemple, la question du nombre d'ensembles bootstrap qu'il faut générer dans un Bagging pour parvenir à créer un ensemble le plus performant possible, n'est pas un problème résolu aujourd'hui. De même, le nombre de caractéristiques sélectionnées aléatoirement pour chaque classifieur

[1] M.L. Minsky. *Logical versus analogical or symbolic versus connectionist or neat versus scruffy.* AI Magazine, vol. 12, no. 2, pages 34–51, 1991.

élémentaire est un hyperparamétre de la méthode Random Subspaces pour lequel nous ne disposons pas de règle de paramétrisation.

Parmi les méthodes d'induction d'EoC, les méthodes de forêts aléatoires n'échappent pas à cette critique. C'est donc principalement de ces problématiques du règlage des hyperparamètres et de la compréhension de leurs effets sur le comportement des forêts aléatoires dont il est question dans ce manuscrit.

Les méthodes de forêts aléatoires sont basées sur la combinaison de classifieurs élémentaires de types arbres de décision. Individuellement, ces classifieurs ne sont pas réputés pour être particulièrement efficaces, mais ils possèdent des propriétés intéressantes à exploiter au sein d'un EoC : ils sont particulièrement instables. La spécificité des arbres utilisés dans les forêts aléatoires est que leur induction est perturbée d'un facteur aléatoire, et ce dans le but de générer de la diversité dans l'ensemble. C'est sur la base de ces deux éléments — utiliser des arbres de décision comme classifieurs élémentaires et faire intervenir l'aléatoire dans leur induction — qu'a été introduit le formalisme des forêts aléatoires.

Bien que les concepts qui y sont définis avaient déjà été mis en œuvre auparavant, c'est l'article fondateur de Breiman[2] qui a le plus contribué à populariser ces méthodes. Une des raisons à cela est qu'il y propose un algorithme d'induction de forêts aléatoires appelé Forest-RI, aujourd'hui considéré comme l'algorithme de référence. Cet algorithme a par ailleurs prouvé au fil des études qu'il était particulièrement compétitif avec les principales méthodes d'ensembles. Cependant, son utilisation pose toujours aujourd'hui d'importantes difficultés. Sa mise en œuvre nécessite notamment de fixer la valeur de deux paramètres[3] principaux :

1. le nombre de caractéristiques choisies aléatoirement pour chaque nœud des arbres
2. le nombre d'arbres aléatoires à induire dans la forêt

Si l'influence de chacun de ces paramètres est avérée, il existe très peu de travaux dans la littérature qui s'y intéressent. S'agissant du nombre de caractéristiques aléatoires par exemple, il n'existe dans la littérature que des valeurs arbitraires traditionnellement utilisées dans les expérimentations, mais aucune règle de paramétrisation, ni aucun élément de compréhension de l'influence de ce paramètre sur les performances n'en permettent le réglage. Le nombre d'arbres quant à lui a fait l'objet par Breiman d'une démonstration mathématique qui prouve que les performances convergent pour un nombre croissant d'arbres dans la forêt, mais aucun élément n'est fourni pour déterminer la quantité nécessaire pour atteindre cette convergence.

En définitive, les utilisateurs des forêts aléatoires ne disposent aujourd'hui d'aucun outil de paramétrisation qui leur permettrait de règler les valeurs de ces

[2] L. Breiman. *Random Forests*. Machine Learning, vol. 45, no. 1,pages 5–32, 2001.

[3] On aurait pu ici utiliser le terme hyperparamètre. Nous les désignons cependant tout au long de ce document de paramètres

paramètres *a priori*. C'est la raison pour laquelle nous avons décidé avec ces travaux de thèse de les étudier en profondeur, en suivant la même démarche pour chacun d'eux : (i) étudier leur influence sur les performances des forêts de la façon la plus rigoureuse possible (ii) apporter des explications à cette influence (iii) exploiter ces explications pour améliorer le processus d'induction de forêt Forest-RI.

Ce manuscrit de thèse est organisé en 5 chapitres. Les deux premiers chapitres présentent une étude bibliographique des méthodes de forêts aléatoires, en commençant par en détailler le cadre global. Nous introduisons donc dans le chapitre 1 le contexte de la combinaison de classifieurs et présentons une taxonomie qui encapsule plusieurs familles de systèmes multi-classifieurs, dont celle qui nous intéresse dans ce document, à savoir les Ensembles de Classifieurs (EoC). Nous présentons ensuite dans le chapitre 2 un état de l'art des méthodes de forêts aléatoires, en détaillant notamment leurs principes de fonctionnement et en listant les principaux algorithmes d'induction. Nous discutons également plus en détails le fonctionnement de l'algorithme Forest-RI et introduisons les problématiques auxquelles nous nous sommes intéressé dans ces travaux de thèse.

Dans le chapitre 3, nous présentons notre première contribution qui a porté sur le paramètre qui désigne le nombre de caractéristiques tirées aléatoirement à chaque nœud des arbres, caractéristiques parmi lesquelles est ensuite déterminée la régle de partitionnement. Ce paramètre contrôle la "quantité" d'aléatoire injectée dans l'induction des arbres par le processus de randomisation Random Feature Selection pour générer de la diversité. Traditionnellement, les chercheurs utilisent dans leurs travaux certaines valeurs par défaut, qui ont initialement été choisies de façon arbitraire, et qui n'ont à notre connaissance jamais été justifiées dans la littérature. Nous montrons que, si l'une de ces valeurs permet d'obtenir des forêts performantes dans la plupart des cas, il existe des problèmes pour lesquels elle est, comme les autres valeurs par défaut, clairement sous-optimale. Nous mettons ensuite en évidence que la "qualité" de l'espace de description permet d'expliquer ces différences de comportement des forêts vis à vis de ce paramètre. Et enfin, nous proposons un algorithme original d'induction de forêts aléatoires, appelé Forest-RK, qui exploite ces conclusions pour remédier au problème que pose le réglage de ce paramètre. Nous évaluons ses performances et montrons qu'il est une alternative intéressante à l'utilisation de Forest-RI avec l'une ou l'autre des valeurs arbitraires de la littérature.

Nous présentons ensuite dans le chapitre 4 notre deuxième contribution qui a porté cette fois sur l'étude du nombre d'arbres qui composent les forêts et leur contribution aux performances de l'ensemble. En générant un large panel de sous-forêts et en étudiant leurs performances dans un premier temps, nous montrons qu'une proportion parfois importante des arbres qui composent une forêt peuvent en détériorer les performances. Nous expliquons dans un second temps ces variations de performances à l'aide de deux propriétés importantes introduites par Breiman : la *force* et la *corrélation* des forêts. Nous traduisons les variations de performances des forêts à l'aide de ce compromis *force/corrélation* et donnons ainsi des éléments importants pour la compréhension des mécanismes de fonctionnement de ces méthodes. Nous

montrons en définitive avec ces travaux l'intérêt de remplacer les processus d'induction statique, traditionnellement mis en œuvre pour l'apprentissage des forêts aléatoires, par un processus d'induction dynamique. Nous fournissons par la même occasion des pistes importantes pour la mise en œuvre d'une telle procédure.

Notre dernière contribution, que nous présentons dans le chapitre 5, prolonge les travaux du chapitre 4 en mettant directement en œuvre les conclusions qui y sont dressées. Nous présentons une procédure originale d'induction dynamique de forêts aléatoires dont le principe est de diriger l'induction des arbres de la forêt pour que ceux-ci complètent au mieux les arbres déjà induits. Nous proposons alors plusieurs exemples de mise en œuvre de ce principe qui s'avèrent pour certaines particulièrement concluantes. En étudiant ensuite l'évolution des performances en généralisation de ces forêts dynamiques pour un nombre croissant d'arbres, nous mettons en évidence la convergence de ces performances. Et enfin, nous expliquons ce gain de performances en comparaison avec les processus d'induction statique, en montrant que la procédure d'induction dynamique parvient à progressivement augmenter la *force* des forêts, tout en atténuant leur *corrélation*.

Finalement, nous concluons ce manuscrit par un bilan critique des contributions apportées. Nous dégageons également en perspectives un ensemble de pistes qu'il nous semble intéressant de creuser, à court, moyen et plus long termes.

Chapitre 1

Ensembles de Classifieurs

Sommaire

1.1 Introduction

Dans le domaine de la combinaison de classifieurs, la littérature est abondante. L'idée de combiner plusieurs experts pour former un comité est assez ancienne puisque l'on juge les premiers travaux allant dans ce sens datant du milieu des années 60 ([Nilsson 1965]). Depuis, l'idée s'est développée pour prendre son essor dans les années 90. Aujourd'hui, nombreux sont les travaux qui tentent de formaliser, analyser, comprendre et faire avancer ce domaine. Certains ouvrages y sont entièrement dédiés ([Kuncheva 2004]), des *workshops* y sont consacrés chaque année ([Kittler 2000, Kittler 2001, Roli 2002a, Windeatt 2003, Roli 2004, Oza 2005, Haindl 2007, Benediktsson 2009]), et les journaux internationaux du domaine de l'apprentissage automatique, et plus largement de la reconnaissance de formes, y consacrent des numéros spéciaux ([Fairhurst 1990, Roli 2002b, Dasarathy 2005]). Si bien qu'aujourd'hui, on voit se dessiner une harmonisation des différentes taxonomies qui ont été proposées pour catégoriser la multitude de travaux qui s'y rapportent.

À un premier niveau d'organisation, on distingue bien souvent les différentes architectures que l'on peut utiliser pour la conception d'un système multi-classifieurs. Combinaisons parallèle, série ou hybride servent en général de point de départ. De ces trois approches, le paradigme le plus répandu est incontestablement la combinaison parallèle. C'est celui qui semble correspondre le plus naturellement à l'idée de faire participer à une même décision plusieurs experts, d'égale importance. Chacun donne son avis pour la prédiction d'une donnée, et tous ces avis sont pris en compte dans la décision finale, le plus souvent à l'aide d'un vote.

À un second niveau d'organisation, on affine généralement cette catégorisation des méthodes en fonction du type de classifieurs qui composent le comité. On oppose alors les comités de classifieurs de types différents, comme ceux qui combinent des réseaux de neurones, des arbres de décision et des k-plus proches voisins dans un même comité par exemple ([Giacinto 2001, Ruta 2004]), aux ensembles composés de classifieurs de même type, comme les ensembles d'arbres de décision — dont il sera largement question dans ce manuscrit — par exemple. Cette dernière catégorie est généralement désignée sous le nom d'Ensemble de Classifieurs (EoC pour *Ensemble of Classifiers*).

Dans la catégorie des EoC, puisque les classifieurs sont de même type, il est évidemment important de générer des "différences" entre chacun d'eux. Avoir un comité de classifieurs qui fournissent tous exactement les mêmes prédictions n'a pas réellement de sens. Un dernier niveau d'organisation catégorise donc les méthodes d'induction d'EoC en fonction de la façon dont elles génèrent ces différences (on parlera alors de diversité) au sein du comité.

Dans ce chapitre, nous détaillons chacun de ces niveaux de granularité jusqu'à présenter quelques unes des méthodes les plus populaires pour induire des ensembles de classifieurs. Dans une première section nous commençons par exposer les raisons qui ont poussé les chercheurs à combiner plusieurs classifieurs au sein d'un même système de classification. Dans une deuxième section ensuite, nous présentons le pre-

mier niveau de catégorisation avec les différentes architectures de combinaison possibles. Nous abordons également les différents paradigmes pour l'induction d'EoC, pour finir par présenter dans une dernière section les 4 principales approches pour construire un ensemble de classifieurs.

1.2 Pourquoi combiner plusieurs classifieurs ?

Depuis la fin des années 90, les EoC suscitent un intérêt particulier de la communauté scientifique pour la simple raison qu'ils se montrent au fil des études particulièrement performants. Notamment, il a été démontré à travers plusieurs travaux de recherche que les EoC sont une solution efficace à de nombreux problèmes auxquels se confrontent les algorithmes qui induisent des classifieurs uniques. Dietterich fournit dans [Dietterich 2000] une catégorisation de ces problèmes, en trois types de limitation :

Limitation Statistique (variance) : la première de ces limitations est dite statistique. On peut voir un algorithme d'apprentissage comme un algorithme de recherche d'un classifieur (ou hypothèse) dans un espace $\mathcal{H}$ des classifieurs possibles. La tâche de l'algorithme consiste alors à trouver le classifieur qui approche le mieux la véritable fonction $y = f(\mathbf{x})$. Cette limitation statistique concerne les situations pour lesquelles l'espace de recherche des classifieurs induits par les algorithmes d'apprentissage est particulièrement grand proportionnellement au nombre de données disponibles en apprentissage. Dans une telle situation, plusieurs classifieurs, tous aussi performants en apprentissage les uns que les autres, peuvent donc être induits et l'algorithme d'apprentissage est donc contraint d'en "choisir" un et un seul. Or, il est très souvent possible de réduire le risque de choisir un "mauvais" classifieur, *i.e.* un classifieur assez peu performant en généralisation, en combinant les prédictions de plusieurs des classifieurs qui se sont montrés performants en apprentissage. La figure 1.1 illustre cette situation en représentant l'espace des classifieurs possibles $\mathcal{H}$, à l'intérieur duquel est dessiné le sous-ensemble des classifieurs les plus performants en apprentissage. Le classifieur combinant, noté h_c, est alors bien plus intéressant que l'ensemble des classifieurs individuels qu'il combine, en ce sens qu'il approche bien mieux f.

Limitation de représentation (biais) : dans beaucoup de problèmes d'apprentissage automatique, la véritable fonction f ne peut être correctement représentée par aucun des classifieurs de $\mathcal{H}$. Il est alors possible d'étendre l'espace de recherche $\mathcal{H}$, en y ajoutant des classifieurs combinant plusieurs classifieurs individuels de l'espace (cf. figure 1.2). De cette manière, on peut arriver à approximer la fonction f à l'extérieur de l'espace $\mathcal{H}$.

On peut citer par exemple les nombreux travaux sur la résolution de problèmes de classification multi-classes à l'aide d'algorithmes d'apprentissage destinés aux problèmes à deux classes. Lorsque l'on souhaite résoudre un problème de ce type

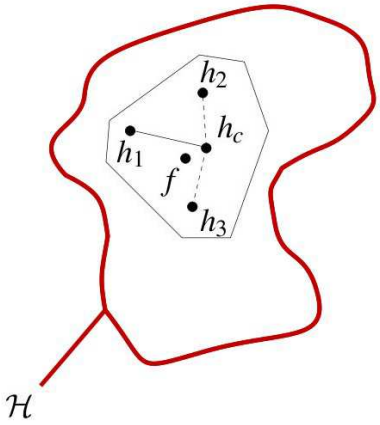

FIG. 1.1: Illustration schématique de la limitation statistique des classifieurs uniques

avec des classifieurs exclusivement binaires — c'est-à-dire ne prenant en charge que des problèmes à deux classes — une stratégie souvent adoptée consiste à résoudre séparément plusieurs des sous-problèmes binaires qu'il contient. La décision finale peut alors être prise par combinaison des décisions prises pour les problèmes intermédiaires. C'est une approche souvent utilisée par exemple avec les classifieurs de type Support Vector Machine (SVM), puisqu'initialement, ces méthodes ne prennent en charge que les problèmes à deux classes ([Hsu 2002]).

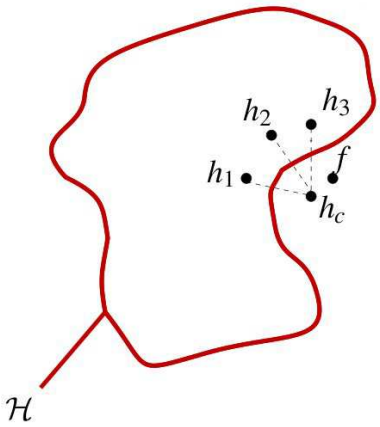

FIG. 1.2: Illustration schématique de la limitation de représentation des classifieurs uniques

Limitation computationnelle : la troisième et dernière limitation est computationnelle. Beaucoup d'algorithmes d'apprentissage procèdent par recherche heuristique locale dans $\mathcal{H}$, au risque de finir par choisir un classifieur qui est localement optimal mais globalement sous-optimal. En d'autres termes, le classifieur

choisi semble approcher f selon les critères de recherche de l'algorithme mais n'est en fait qu'une solution intermédiaire. Dietterich cite en exemple dans son article ([Dietterich 2000]) les réseaux de neurones qui utilisent souvent une méthode de descente de gradient pour minimiser l'erreur en apprentissage, ou encore les arbres de décision qui partitionnent l'espace de description de façon récursive et donc non exhaustive. Le problème ici est qu'il est alors très coûteux pour un algorithme d'apprentissage d'effectuer une recherche plus "large" afin d'éviter les *optima* locaux. Or, il est également possible dans ce type de situation de réduire le risque de choisir un mauvais classifieur en combinant plusieurs des classifieurs sous-optimaux — localement optimaux — comme l'illustre la figure 1.3. De cette façon, un ensemble de classifieurs obtenus en lançant plusieurs recherches heuristiques à partir de différents points de l'espace $\mathcal{H}$ permet d'obtenir une meilleure approximation de f.

Puisque Dietterich cite les arbres de décision en exemple de classifieurs basés sur une recherche heuristique, un bon exemple de combinaison de classifieurs qui pallie cette difficulté pourrait être les arbres de décision "baggés". Il s'agit d'utiliser le principe d'apprentissage de Bagging (cf. section 1.4.1) avec des classifieurs individuels de type arbre de décision pour induire un ensemble d'arbres. De cette manière, chacun des arbres est induit à partir d'un sous-ensemble d'apprentissage différent et, par conséquent, à partir d'un point de l'espace $\mathcal{H}$ différent. De nombreux travaux ont mis en évidence le gain de performances que permet d'obtenir cette méthode, à commencer par le premier article de Breiman qui introduisit le principe de Bagging [Breiman 1996] [Amit 2000, Bauer 1999, Banfield 2006].

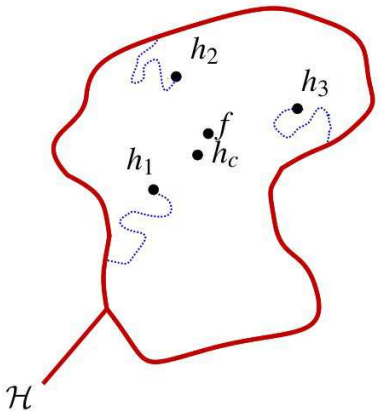

FIG. 1.3: Illustration schématique de la limitation computationnelle des classifieurs uniques

Ces trois limitations fondamentales constituent selon Dietterich les principales situations pour lesquelles les algorithmes induisant un classifieur unique échouent. Les ensembles de classifieurs fournissent alors une solution intéressante à ces difficultés.

Cette catégorisation de Dietterich est très fortement ancrée dans la théorie de

l'apprentissage statistique qui définit un cadre théorique très populaire pour formaliser les problèmes d'apprentissage automatique. On peut cependant donner des interprétations plus pragmatiques aux raisons qui peuvent pousser à utiliser des combinaisons de classifieurs à la place de classifieurs uniques. La première d'entre elles — celle qui semble la plus immédiate — concerne le gain de précision. L'idée est de combiner les sorties de plusieurs classifieurs simplement dans l'espoir d'obtenir une décision plus fiable. Elle rejoint en ce sens la première des limitations mentionnées par Dietterich. En confrontant l'avis de plusieurs experts à propos d'un même problème, on peut espérer compenser les erreurs de quelques-uns d'entre eux par une majorité de bonnes prédictions données par les autres.

Une autre raison pour vouloir combiner des classifieurs est la complexité du problème traité. Si le problème est trop complexe pour être traité par un classifieur unique, il peut être intéressant de le décomposer en plusieurs sous-problèmes plus simples que l'on pourrait donc ensuite résoudre séparément par des classifieurs différents. La prise de décision est ensuite effectuée par combinaison de ces classifieurs.

Une dernière raison enfin nous est expliquée par le fameux *no free lunch theorem*, très apprécié des spécialistes de l'optimisation numérique — la raison principale en étant qu'il prouve que ceux-ci ne seront jamais à cours de problèmes à résoudre. Transposé à l'apprentissage automatique, ce théorème démontre en définitive qu'il n'existe aucun modèle unique, et donc classifieur unique, capable de résoudre tout type de problèmes ([Wolpert 1997]). La perspective d'un tel résultat est qu'il est plus pertinent dès lors de travailler à un plus haut niveau de résolution des problèmes d'apprentissage qui s'appuierait sur des systèmes managériaux pouvant tirer le meilleur parti de plusieurs sytèmes de résolution plus "bas niveau" ([Minsky 1991]) : les combinaisons de classifieurs.

En résumé, on peut vouloir combiner plusieurs classifieurs pour les raisons suivantes :

- Ils permettent de fiabiliser les décisions en s'appuyant sur l'avis de plusieurs experts au lieu d'un seul.
- Ils élargissent l'ensemble des solutions possibles en proposant des modèles plus complexes que l'on ne pourrait pas obtenir avec des classifieurs uniques.
- Ils permettent d'éviter les *optima* locaux.
- Ils permettent de traiter des problèmes trop complexes pour être appréhendés dans leur globalité.
- Ils sont plus génériques que les classifieurs uniques, et peuvent appréhender plus efficacement un plus grand nombre de problèmes.

Tous ces points forts sont liés à une propriété importante des systèmes multi-classifieurs, appelée **diversité**. C'est une propriété dont la communauté s'accorde à dire qu'elle est importante pour expliquer chacun de ces avantages par rapport aux classifieurs uniques ([Kuncheva 2003b, Kuncheva 2003a, Kuncheva 2004]).

S'il n'existe pas de définition formelle sur laquelle toute la communauté scientifique s'accorde, on peut définir la diversité de façon plus intuitive : elle exprime les différences entre les classifieurs élémentaires au sein d'un ensemble. On ne peut obtenir de gain de performances avec un ensemble de classifieurs que si ceux-ci sont raisonnablement différents les uns des autres. La diversité est le nom que l'on donne dans le domaine de l'apprentissage automatique à la quantification de ces différences.

Le terme "différences" entre les classifieurs exprime ici non pas les différences dans leur nature, mais les différences dans leurs prédictions. Plus précisément, et dans le cas de la classification plus particulièrement, on considère communément que la diversité d'un ensemble de classifieurs exprime leur capacité à s'accorder sur les bonnes prédictions, et à être en désaccord sur les erreurs de prédiction.

L'enjeu de la combinaison de classifieurs est bien souvent de générer de la diversité dans l'ensemble des classifieurs élémentaires que l'on combine ([Kuncheva 2000, Banfield 2003, Kuncheva 2003b, Kuncheva 2004, Brown 2005]). Pour cela, il existe beaucoup d'approches différentes. Dans la section suivante, nous abordons les différents paradigmes de combinaison de classifieurs et les différentes approches pour construire des systèmes multi-classifieurs basés sur des comités de classifieurs divers.

1.3 Combinaisons de classifieurs

1.3.1 Architectures de Combinaison

Après la question du "pourquoi", intéressons nous maintenant à la question du "comment".

Il existe plusieurs schémas de combinaison très différents les uns des autres et qui ont chacun un intérêt différent. Tous ces schémas ont par ailleurs fait l'objet de plusieurs tentatives de catégorisation ([Heutte 1994, Moobed 1996, Rahman 1999, Dietterich 2000, Amit 2000, Roli 2001, Rahman 2003, Kuncheva 2004, Zouari 2004]). À partir de ces travaux, on distingue au premier niveau de formalisation trois grands schémas de combinaison de classifieurs adoptant des architectures différentes :

- **Combinaison Séquentielle ou Série :** ce schéma de combinaison organise les classifieurs élémentaires en niveaux successifs de décision, de sorte que chacun d'eux prenne en compte la prédiction du classifieur placé en amont (*cf.* figure 1.4). Les classes candidates sont ainsi progressivement évincées jusqu'à ce qu'il ne reste qu'une décision possible. Rahman *et al.* dans [Rahman 2003] ou encore Kittler dans [Kittler 2004] dressent un panorama assez complet des méthodes de combinaison adoptant ce type d'approches. Nous n'entrerons pas plus dans les détails de tous ces travaux ici, puisque les méthodes qui nous intéressent, *i.e.* les forêts aléatoires, ne correspondent pas, par définition, à ce type de stratégie.
- **Combinaison Parallèle :** dans ce schéma, les classifieurs élémentaires opèrent indépendamment les uns des autres et prennent leurs décisions sans

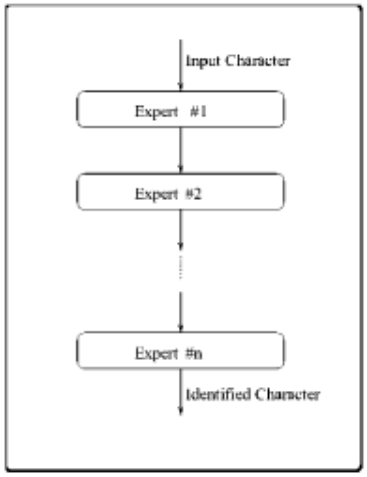

FIG. 1.4: Combinaison séquentielle de classifieurs ([Rahman 2003])

tenir compte du reste du comité. Les décisions individuelles sont ensuite fusionnées à l'aide d'un opérateur de combinaison, comme par exemple un vote à la majorité simple, le but étant la recherche d'un consensus (*cf.* figure 1.5). Il existe plusieurs taxonomies des méthodes de combinaison parallèle ([Kuncheva 2004, Zouari 2002, Zouari 2004, Kuncheva 2001, Ruta 200, Ho 1992, Duin 2000]) dont nous détaillons quelques aspects importants dans la suite de cette section.

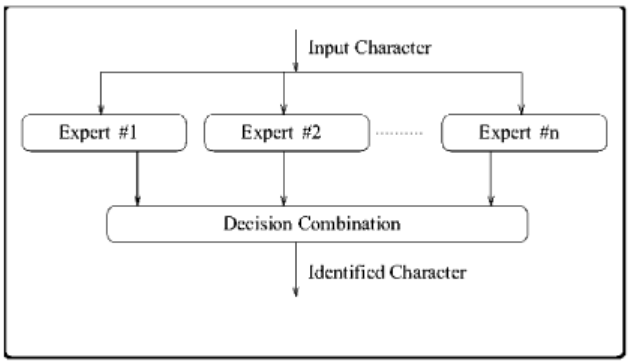

FIG. 1.5: Combinaison parallèle de classifieurs ([Rahman 2003])

- **Combinaison Hybride (Séquentielle-Parallèle) :** comme son nom le suggère, il s'agit avec cette approche d'utiliser un schéma qui reprend simultanément les principes de combinaison séquentielle et parallèle. Les méthodes appartenant à cette catégorie sont généralement conçues pour des applications spécifiques, comme c'est le cas par exemple de la méthode proposée par Kim *et al.* dans [Kim 2000] pour la reconnaissance de mots cursifs anglais extraits de chèques bancaires. Dans cette méthode, deux niveaux de classification sont

mis en œuvre ; le premier pouvant traiter parallèlement deux espaces de description concurrents, et le deuxième ayant pour rôle de fusionner les décisions du niveau précédent. Rahman et Fairhurst donnent dans [Rahman 2003] une liste conséquente de références, dont certaines sur ce type d'approche, dans leur état de l'art des méthodes de combinaison de classifieurs pour la reconnaissance d'écriture manuscrite. La figure 1.6 illustre un exemple de ce type d'approches.

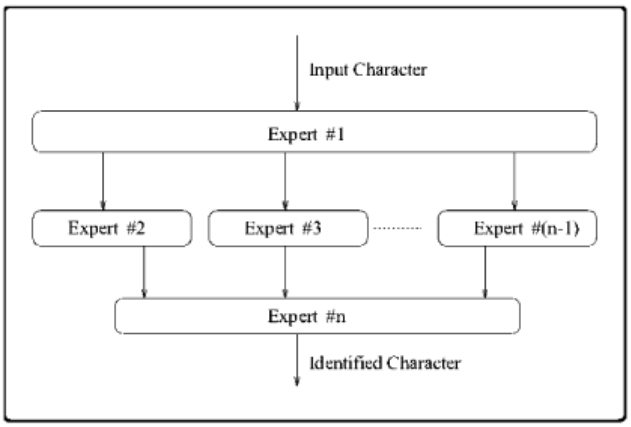

FIG. 1.6: Combinaison hybride de classifieurs ([Rahman 2003])

Parmi ces trois approches, celle qui suscite le plus grand intérêt de la communauté scientifique est la combinaison parallèle de classifieurs. Il suffit pour s'en rendre compte de faire un tour d'horizon des principaux articles publiés depuis 2000 dans le cadre de la série de Workshops **Multiple Classifier Systems** ([Kittler 2000, Kittler 2001, Roli 2002a, Windeatt 2003, Roli 2004, Oza 2005, Haindl 2007, Benediktsson 2009]). Certaines des taxonomies qui y ont été présentées ne tiennent tout simplement pas compte des deux autres schémas existants [Dietterich 2000, Roli 2001]. La raison principale à cela est que les combinaisons de classifieurs qui s'appuient (même partiellement) sur une architecture séquentielle sont très difficiles à optimiser. Le caractère séquentiel qui rend chaque classifieur élémentaire dépendant des classifieurs précédents rend difficile la manipulation du système multi-classifieurs dans son ensemble.

A noter tout de même que quelques travaux de Kittler, par ailleurs très actif dans le domaine de la combinaison de classifieurs et dans l'organisation de ces workshops, ayant au préalable mis en évidence cette prédominance des systèmes de combinaison parallèle, s'intéressent un peu plus en détails aux méthodes de combinaison série [Kittler 2003, Kittler 2004].

Dans la catégorie des méthodes de combinaison parallèle, la littérature se divise ensuite en deux types d'approches :

- les systèmes qui utilisent plusieurs classifieurs élémentaires de types diffé-

rents, comme par exemple dans [Kittler 1998] où les auteurs s'intéressent entre autres à la combinaison de trois classifieurs de types différents, *i.e.* un classifieur Bayésien, un Réseau de Neurones (NN), et un Modèle de Markov Cachés (HMM), pour la reconnaissance d'écriture manuscrite. D'autres exemples de ce type de combinaison peuvent être trouvés dans [Bahler 2000, Tax 2000, Tax 2001, Boinee 2005a].

- les systèmes qui s'appuient sur un comité de classifieurs élémentaires de même type. C'est principalement de cette approche qu'il va s'agir dans la suite de ce document. Cela consiste à utiliser le même algorithme d'apprentissage, et donc le même type de classifieur, pour tous les classifieurs élémentaires de l'ensemble, et à créer des différences dans leurs prédictions en utilisant par exemple des ensembles d'apprentissage différents ou des espaces de description différents pour ces données. Ce type de combinaison parallèle est généralement désigné par le terme **"Ensemble de Classifieurs" (EoC)**, et les méthodes qui produisent ces systèmes multi-classifieurs sont appelées "méthodes d'induction d'ensembles de classifieurs".

Les méthodes de forêts aléatoires, qui nous intéressent plus particulièrement dans ces travaux de thèse, sont des méthodes qui appartiennent à la famille des EoC, puisqu'elles combinent par définition plusieurs classifieurs élémentaires de type arbre de décision. C'est pourquoi dans la section suivante, nous abordons plus en détails cette famille de méthodes, ainsi que la problématique de l'induction d'EoC, et détaillons également quelques-unes des principales méthodes d'induction d'EoC.

1.3.2 Induction d'Ensembles de Classifieurs

L'induction d'EoC consiste à générer, à partir d'un unique algorithme d'apprentissage, un ensemble de classifieurs capables de donner des prédictions différentes sur un même ensemble de données à classer. L'enjeu de l'induction d'EoC est donc de définir un moyen de créer ces différences parmi le comité de classifieurs, pourtant tous générés avec le même algorithme d'apprentissage.

Dans son ouvrage **Combining Pattern Classifiers** ([Kuncheva 2004]), Ludmila Kuncheva présente une catégorisation des méthodes d'induction d'EoC que l'on retrouve dans plusieurs des principaux états de l'art sur ce sujet ([Dietterich 2000, Banfield 2004, Banfield 2006]). Cette taxonomie présente les différentes contributions en terme d'induction d'EoC, selon trois approches par niveaux d'action[1] :

- **Le niveau "donnée" :** cette catégorie regroupe les méthodes d'induction d'ensembles qui sont basées sur la manipulation des données d'apprentissage, et plus particulièrement de leur distribution, pour induire des classifieurs élémentaires différents. Il s'agit principalement de générer des sous-ensembles de

[1] A noter que dans la taxonomie de Kuncheva est mentionnée une quatrième approche qui regroupe les méthodes agissant au niveau de l'opérateur de combinaison. De notre point de vue ces méthodes ne répondent pas à la problématique de l'induction des ensembles de classifieurs, mais plus à la problématique des opérateurs de combinaison

données différents pour l'apprentissage de chacun des classifieurs individuels. On retrouve parmi ces méthodes les principes de Bagging et de Boosting, dont nous parlons plus en détails respectivement dans les sections 1.4.1 et 1.4.4.

- **Le niveau "caractéristique" :** il s'agit ici des méthodes qui manipulent cette fois les espaces de description des données. Classiquement, comme avec les méthodes du niveau "donnée", les classifieurs individuels seront appris sur des sous-espaces de description différents. La méthode la plus représentative de cette catégorie est la méthode des Random Subspaces ([Ho 1998]) pour laquelle les sous-espaces sont sélectionnés aléatoirement (*cf.* section 1.4.2).
- **Le niveau "classifieur" :** les méthodes de ce niveau s'intéressent plus particulièrement quant à elles à l'algorithme d'induction des classifieurs. L'idée est généralement d'utiliser des paramètrages différents chaque fois qu'un classifieur élémentaire est induit avec l'algorithme d'apprentissage. Par exemple, lorsque l'on génère un ensemble de réseaux de neurones, on initialise souvent les poids de ces réseaux de façon aléatoire, comme l'expliquent Opitz *et al.* dans leur étude comparative de quelques méthodes d'induction d'ensembles de classifieurs ([Opitz 1999a]).

S'il s'agit là des trois principales catégories de méthodes d'induction d'ensembles de classifieurs, Kuncheva cite cependant quelques méthodes qu'elle ne considère appartenir à aucune d'entre elles. C'est le cas notamment des méthodes de la famille des **Error Correcting Output Codes (ECOC)** (*cf.* section 1.4.3), qui manipulent les étiquettes des données d'apprentissage pour générer des sous-problèmes aléatoires différents pour chaque classifieur élémentaire.

Dietterich quant à lui préfère proposer avec sa taxonomie une catégorie à part entière pour ce type d'approche ([Dietterich 2000]), qu'il nomme *Manipulating the Output Label* — en français **Manipulation des étiquettes**. On retrouve par ailleurs dans cette autre catégorisation deux approches nommées **Manipulation des données** et **Manipulation des caractéristiques**, correspondant respectivement aux niveaux "donnée" et "caractéristique" de Kuncheva. En plus de ces trois catégories, Dietterich en ajoute une quatrième qu'il intitule *Injecting Randomness* — que nous traduisons à l'aide du terme **Randomisation** qui exprime le fait d'injecter de l'aléatoire dans le processus d'induction de l'ensemble. Il réunit dans cette catégorie les méthodes qui injectent de l'aléatoire dans l'algorithme d'apprentissage des classifieurs individuels. Ces méthodes sont donc en général spécifiques à certains types de classifieurs élémentaires, contrairement aux principes précédemment mentionnés qui fonctionnent pour tout type de classifieur élémentaire. Il cite en exemple les ensembles de réseaux de neurones basés sur l'utilisation de rétropropagation du gradient dont les poids sont initialisés aléatoirement pour chaque réseau. Il cite également quelques principes de randomisation d'arbres de décision sur lesquels nous reviendrons en détails dans la section 2.6, puisque la plupart de ces principes interviennent dans les algorithmes d'induction de forêts aléatoires.

En résumé, en recoupant ces deux taxonomies prédominantes dans les travaux concernant l'induction d'ensembles de classifieurs, on peut distinguer quatres principales stratégies, non exclusives, pour créer des EoC :

- Manipuler la distribution des données d'apprentissage.
- Manipuler les espaces de description.
- Manipuler les étiquettes de classes.
- Manipuler les paramètres de l'algorithme d'apprentissage.

Dans la section suivante, nous détaillons les principales méthodes d'induction d'EoC, qui s'appuient chacune sur un ou plusieurs de ces quatre paradigmes.

1.4 Principes d'induction d'ensembles de classifieurs

1.4.1 Bagging

1.4.1.1 Principes de fonctionnement

La méthode de Bagging dans sa version "classique" fait partie de la catégorie des méthodes de manipulation de données dans la taxonomie des principes d'induction d'ensemble de classifieurs que nous avons présentée dans la section précédente. Elle applique le principe de **Bootstrap** ou **Bootstrapping** à l'aggrégation de classifieurs ; d'où son nom Bagging pour **B**ootstrap **Agg**regat**ing**.

Le Bootstrap est un principe de rééchantillonnage statistique ([Efron 1993]) traditionnellement utilisé pour l'estimation de grandeurs ou de propriétés statistiques. Il permet, en plus de fiabiliser les estimations statistiques, de fournir plus d'indications sur ces estimations. Il permet, par exemple, de calculer la dispersion (via l'écart-type ou la variance) des mesures, des intervalles de confiance, ou encore des tests d'hypothèses. En statistiques, lorsque l'on souhaite approximer la distribution d'une population de données, on utilise généralement une distribution empirique de données observées. L'idée du bootstrap est d'utiliser non plus une unique distribution empirique, mais plusieurs ensembles de données rééchantillonnées à partir de l'ensemble des données observées et ce à l'aide d'un tirage aléatoire avec remise.

Supposons que l'on dispose d'un ensemble $T = \{x_1, x_2, x_3..., x_N\}$ de N données observées de notre population, et que l'on s'intéresse à un statistique notée $S(T)$. Le bootstrap va consister à former L échantillons $T_k^* = (x_1^*, x_2^*, x_3^*, ..., x_{N'}^*)$ pour $k = 1, .., L$, où chaque T_k^* est constitué par tirage alétoire avec remise de N' données dans T (*cf.* figure 1.7). Ces L échantillons sont usuellement appelés les **échantillons bootstrap**.

On peut alors calculer $S(T_k^*)$ pour chaque échantillon bootstrap, et obtenir ainsi L estimations de notre statistique. Au lieu donc de disposer d'une seule estimation pour une réalisation d'un échantillon, on dispose d'une distribution empirique de notre statistique. On peut alors calculer la moyenne empirique à partir de toutes ces valeurs (*cf.* équation 1.1), qui nous donnera alors une estimation plus précise

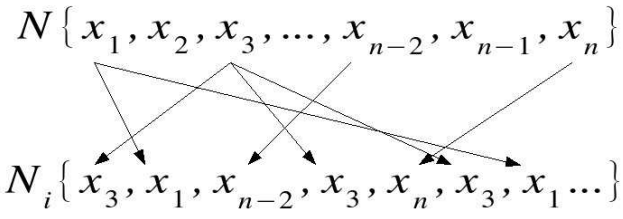

FIG. 1.7: Illustration d'un tirage aléatoire avec remise pour la formation d'un échantillon bootstrap

de la statistique, ou encore son erreur standard pour en mesurer la dispersion (*cf.* équation 1.2).

$$S_{boot} = \sum_{k=1}^{L} S(T_k^*)/L \tag{1.1}$$

$$\widehat{se}_{boot} = \sqrt{\frac{\sum_{k=1}^{L}(S(T_k^*) - S_{boot})^2}{(L-1)}} \tag{1.2}$$

À partir de ce principe de rééchantillonnage, Breiman en 1996 introduit la méthode de Bagging ([Breiman 1996]). Il s'agit simplement de considérer que la statistique que l'on cherche à étudier est un algorithme d'apprentissage noté $h(\mathbf{x})$ et d'appliquer alors le principe de bootstrap tel que nous venons de l'expliquer. Ainsi chaque classifieur élémentaire $h_k(\mathbf{x})$ de l'ensemble sera entrainé sur un des L échantillons bootstrap de sorte qu'ils soient tous entrainés sur un ensemble d'apprentissage différent. La figure 1.8 illustre le procédé de Bagging appliqué à un ensemble d'arbres de décision.

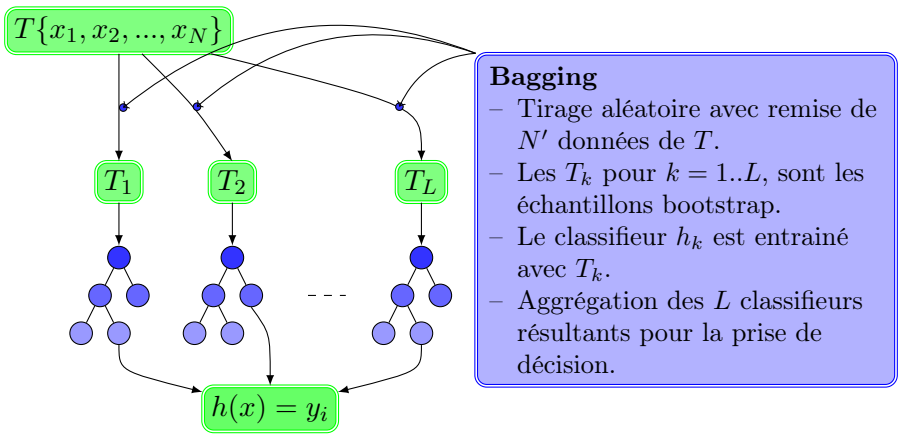

FIG. 1.8: Illustration du principe de Bagging pour un ensemble d'arbres de décision

Bien que ce principe de "méta-apprentissage" fonctionne avec tout type d'algorithmes d'apprentissage, Breiman l'expérimente essentiellement avec des arbres de décision ([Breiman 1996]). La raison en est que la principale force du Bagging est de réduire **l'instabilité** d'un classifieur. Dans ce cas précis, nous entendons par instabilité qu'un petit changement dans la base d'apprentissage provoque un changement important dans la structure de l'arbre et donc dans ses performances en généralisation. Réduire l'instabilité permet alors dans ce cas de fiabiliser les prédictions et d'améliorer les performances en généralisation. Or, les arbres de décision sont des classifieurs très instables.

Les effets du bagging sur l'instabilité d'un classifieur font l'objet d'une longue extrapolation mathématique de Breiman, à partir de laquelle il conclut que l'efficacité du Bagging en classification[2] est conditionnée par l'instabilité du classifieur. Si "bagger" un classifieur instable permet d'en améliorer en moyenne l'erreur en généralisation, "bagger" un classifieur stable n'est pas aussi efficace ([Breiman 1996, Skurichina 2001, Grandvalet 2004]). Skurichina dans [Skurichina 2001] fournit de nombreux détails sur la problématique de stabilité des classifieurs, et expérimente largement l'utilisation du Bagging pour la réduction de l'instabilité de différents classifieurs. Elle démontre expérimentalement par ailleurs que le Bagging ne permet pas d'améliorer les performances des classifieurs stables, comme par exemple les performances d'un k-Plus Proche Voisins.

La principale force du bagging est donc de réduire l'instabilité pour augmenter les performances en généralisation. Mais il y a un autre point qui fait la force du bagging, ce sont les mesures out-of-bag. Nous expliquons maintenant cet outil très pratique en classification.

1.4.1.2 Mesures Out-Of-Bag

La méthode de bootstrap introduit un paramètre dont nous n'avons pas encore parlé : le nombre N' de données tirées aléatoirement pour chaque ensemble bootstrap. Il est en fait classiquement fixé à N par les statisticiens, puisqu'il s'agit initialement de simuler des rééchantillonnages successifs pour une même expérience statistique. Le nombre d'individus dans chaque réplicat doit donc être identique au nombre d'individus dans l'échantillon de départ, pour pouvoir en étudier les propriétés statistiques.

Dans le cadre du Bagging, il est également presque systématiquement fixé à N, mais peut aussi être inférieur. En revanche il n'est pas conseillé de le fixer avec une valeur très supérieure à N. La raison en est qu'avec le bagging, la diversité est introduite dans l'ensemble de classifieurs par les différences créées dans chaque échantillon bootstrap, qui produit des différences dans les prédictions des classifieurs élémentaires — c'est d'autant plus le cas pour les classifieurs très instables. Or plus N' est grand par rapport à N et plus les échantillons bootstrap tendront à être

[2] nous n'abordons pas ici le cas de la régression, pour lequel les démonstrations et les conclusions de Breiman sont sensiblement différentes

similaires.

Un résultat mathématique intéressant lorsque $N' = N$ est que chaque échantillon bootstrap ne contient asymptotiquement — *i.e.* pour N très grand — que $63,2\%$ des données d'apprentissage de T. En effet, si l'on effectue un tirage aléatoire avec remise des données, il n'est pas impossible de tirer plusieurs fois la même donnée pour le même échantillon bootstrap. La conséquence est que pour $N' = N$, toutes les données ne seront pas forcément présentes dans tous les échantillons. Et pour une valeur de N relativement grande, on peut démontrer qu'un peu plus d'un tiers des données n'apparaissent pas dans un échantillon bootstrap donné ([Efron 1993]).

Ce résultat est intéressant car il indique que chaque classifieur n'apprend qu'une partie des données. Les autres données, celles qu'il ne connait pas, sont usuellement appelées les données **out-of-bag** du classifieur. Elles fournissent alors un bon moyen d'obtenir des mesures sur les classifieurs élémentaires, comme par exemple une estimation de leurs performances en généralisation. Dans le cadre d'un ensemble de classifieurs par exemple, si on souhaite estimer l'erreur en généralisation via un vote à la pluralité, il suffit de classer chaque donnée d'apprentissage de T en ne laissant voter que les classifieurs élémentaires pour lesquels elle fait partie de l'ensemble out-of-bag.

A la suite de son article de 2001, Breiman a proposé un grand nombre d'outils basés sur l'utilisation des forêts aléatoires[3], et beaucoup d'entre eux utilisent les données out-of-bag pour éviter d'avoir recours à un ensemble de données de validation.

1.4.2 Random Subspaces

La méthode des Random Subspaces (usuellement notée RSM pour Random Subspace Method) est assez similaire dans l'idée au Bagging ([Ho 1995, Ho 1998, Brill 2003]). Cette fois cependant, il ne s'agit plus de manipuler les données d'apprentissage, mais de manipuler les caractéristiques. Le principe de base est d'entrainer chaque classifieur élémentaire sur un sous-espace aléatoire de l'espace de description. Chacun de ces sous-espaces aléatoires est de même dimension P, avec $P < M$ où M est la dimension de l'espace de description original.

Plus concrètement, pour l'induction d'un classifieur élémentaire h_k, il s'agit :

1. d'effectuer un tirage aléatoire sans remise de P caractéristiques parmi les M caractéristiques disponibles.
2. de projeter toutes les données d'apprentissage dans ce nouveau sous-espace de caractéristiques.
3. d'entrainer le classifieur h_k sur ces projections des données d'apprentissage.

La figure 1.9 illustre l'application de cette procédure à l'induction de forêts de décision.

[3] http ://www.stat.berkeley.edu/users/breiman/RandomForests/

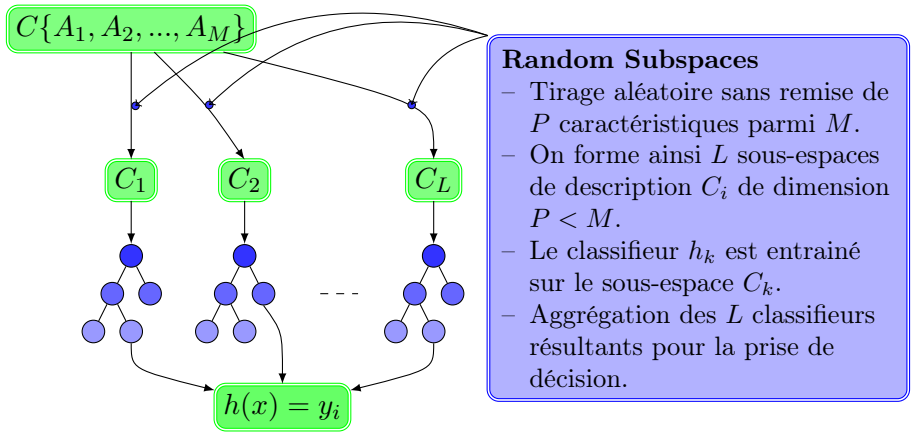

FIG. 1.9: Illustration du principe de Random Subspaces pour un ensemble d'arbres de décision

Dans [Ho 1998], Ho montre à propos du paramètre P que les meilleurs résultats sont généralement obtenus pour $P \approx M/2$ caractéristiques. Elle a de plus mis en évidence que cette méthode était particulièrement efficace quand l'espace de description présente une certaine redondance d'information dispersée sur l'ensemble des caractéristiques, plutôt que concentrée sur un sous-ensemble d'entre elles.

On constate également que cette procédure RSM est applicable pour tout type de classifieur élémentaire. Dans de nombreux travaux, elle a été appliquée à l'induction d'ensembles d'arbres de décision [Ho 1992, Ho 1995, Banfield 2006, Latinne 2000, Latinne 2002], car elle présente un avantage sur d'autres méthodes d'induction de forêts de décision : elle permet de réduire la dimension de l'espace de description des données d'apprentissage. Or, l'induction automatique d'un arbre de décision nécessite des parcours répétés de cet espace. Le coût computationnel est par conséquent considérablement réduit lorsque cette méthode est utilisée, en comparaison avec d'autres méthodes d'induction de forêts de décision qui ne modifient pas l'algorithme d'induction d'arbres de décision.

Au delà de ces considérations computationnelles, cette réduction de la dimension de l'espace de description sur lequel l'algorithme d'apprentissage va travailler, est un point important pour expliquer le bon fonctionnement de la méthode RSM. Cette méthode réduit la dimension de cet espace tout en maintenant le nombre de données d'apprentissage. Or, on sait que les classifieurs entraînés sur des bases de données avec moins de données d'apprentissage qu'il n'y a de caractéristiques pour les décrire présentent une forte instabilité et par conséquent une variance élevée. On parle souvent à ce sujet de la "malédiction de la dimensionnalité". L'utilisation de la méthode RSM permet la plupart du temps de compenser cette instabilité. De plus, lorsque l'espace de description présente de fortes redondances d'information,

apprendre un classifieur sur un sous-ensemble des caractéristiques peut s'avérer plus efficace que de le faire sur l'ensemble. Combiner des classifieurs de ce type permet bien souvent d'améliorer les performances que l'on pourrait obtenir avec un classifieur unique, s'agissant même de classifieurs stables.

Dans [Ho 1998], Ho expérimente cette méthode principalement dans le cadre d'ensembles d'arbres de décision. Chaque arbre de décision est donc entrainé sur un sous-espace de description du problème. L'opérateur de combinaison utilisé dans ces expérimentations est la moyenne des probabilités *a posteriori*. On verra dans la section 2.2 que dans le cadre de forêts de décision, les arbres sont construits jusqu'à atteindre leur taille maximale. Cela implique que l'erreur en apprentissage est toujours nulle et que chaque arbre sur-apprend au maximum les données d'apprentissage. Cependant, la réduction de la dimension de l'espace dans lequel l'arbre travaille peut entrainer des difficultés supplémentaires pour discriminer les données. Il est possible que deux données, parfaitement différenciables dans l'espace de description original, se confondent dans ces sous-espaces aléatoires. Il est donc nécessaire de mettre en place un opérateur de combinaison capable de prendre en charge les données d'apprentissage mal classées.

L'auteur compare sa méthode à 4 autres algorithmes : un arbre de décision seul, avec et sans élagage (*cf.* section 2.2), AdaBoost et Bagging. Elle utilise dans un premier temps 4 bases de données du Projet StatLog [Feng 1993], disponibles sur le site de l'UCI repository [Asuncion 2007]. Ces bases ont été choisies principalement parce qu'elles contiennent beaucoup de caractéristiques et/ou beaucoup de données. Pour chacune d'elles, RSM s'est montré plus performant que les autres méthodes, pour des forêts formées avec 100 arbres. Bien qu'aucun test de significativité n'ait été effectué, les écarts semblent importants, avec les arbres seuls, mais aussi avec AdaBoost et Bagging.

Elle étend ensuite ces comparaisons à des bases contenant un nombre moindre de caractéristiques et/ou de données. Les résultats sont cette fois-ci moins favorables à RSM. Pour seulement 5 de ces 14 nouvelles bases, RSM se montre plus performant que AdaBoost et Bagging. En analysant les propriétés de ces bases et en les comparant aux résultats obtenus, Ho conclut que la méthode RSM est plus avantageuse pour les bases de données qui contiennent un grand nombre de caractéristiques et de données. À l'inverse, elle semble moins performante lorsque que l'on dispose de peu de caractéristiques et de peu de données ou de beaucoup de classes. Pour toutes les autres bases, il semble difficile de déterminer laquelle des trois méthodes est meilleure. Dans tous les cas, les trois méthodes d'induction d'EoC permettent d'obtenir un gain conséquent sur les arbres seuls, élagués ou non.

1.4.3 Error Correcting Output Codes

C'est Dietterich qui introduit les méthodes **Error Correcting Output Code (ECOC)** en 1991 dans [Dietterich 1991]. Nous avons déjà cité cette approche dans la section 1.3.2, où nous avons expliqué qu'il s'agissait de manipuler les étiquettes de

données d'apprentissage. Le principe est précisément de générer des sous-problèmes aléatoires en regroupant de différentes façons les c classes d'un problème multi-classes en deux sous-ensembles aléatoires, notés A_i et B_i. Les données d'apprentissage sont alors ré-étiquetées avec l'une ou l'autre de ces deux super-classes et utilisées pour l'apprentissage d'un classifieur. Cette procédure est ensuite répétée pour tous les classifieurs élémentaires de l'ensemble de sorte qu'ils s'intéressent tous à des problèmes différents, concernant des groupes de classes différents. En phase de prédiction, chaque donnée à prédire est classée par tous les classifieurs, chacun d'entre eux attribuant un vote à l'ensemble des classes appartenant au sous-groupe prédit. Après que tous les classifieurs ont attribué leur vote, ceux-ci sont sommés pour chaque classe, et celle réunissant le plus grand nombre de votes est choisie comme prédiction finale.

Initialement, Dietterich introduisit cette méthode d'induction d'ensembles de classifieurs pour proposer une nouvelle façon de traiter les problèmes multi-classes avec des classifieurs binaires. Le processus est inspiré des travaux de Sejnowski and Rosenberg ([Sejnowski 1987]) sur leur système appelé NETtalk, qui assigne à chaque classe une chaîne de bits unique de longueur n, appelée code de la classe. Il s'agit alors d'apprendre non plus une fonction pour le problème multi-classe, mais n fonctions binaires ; une pour chacun des bits de la chaîne. En phase de prédiction, la classe affecté à une donnée à prédire est déterminée à l'aide d'une mesure de distance — typiquement une distance de Hamming — entre le code obtenu et les différents codes des classes. La classe prédite est celle qui minimise cette distance.

NETtalk réalise cet apprentissage avec des réseaux de neurones, chaque fonction étant modélisée par un des neurones de la couche de sortie. Dietterich a donc étendu ce principe à tout type de classifieur en utilisant des ensembles de classifieurs au lieu de classifieurs uniques, et en randomisant le choix des codes pour chaque classe. Il teste cette méthode dans [Dietterich 1995] sur plusieurs bases de données multi-classes avec des classifieurs élémentaires de types arbres de décision et réseaux de neurones. Il compare les forêts de décision ECOC, pour chaque base de données, à un arbre de décision C4.5 standard naturellement capable de traiter les problèmes multi-classes et à une forêt de décision qui adopte l'approche "un-contre-tous" (*cf.* section 1.2). Sur les 8 bases de données qu'il teste, alors que l'approche un-contre-tous détériore significativement les performances en comparaison avec l'arbre C4.5 sur 4 bases, la forêt ECOC les améliore significativement sur 6 d'entre elles.

Dietterich démontre également dans ce même article que ces résultats sont robustes au nombre de données dans l'ensemble d'apprentissage. Dans [Kong 1995], il étudie avec Kong quelques raisons du succès de ce type de méthodes à travers la décomposition biais-variance de l'erreur et montre que ECOC permet de réduire simultanément la variance et le biais ; la variance par l'utilisation de plusieurs classifieurs élémentaires instables, comme c'est le cas avec la méthode de Bagging par exemple (*cf.* section 1.4.1) ; le biais par la diversité dans les erreurs des classifieurs élémentaires (*cf.* section 1.3.1). Cette diversité est alors créée par le fait que les classifieurs apprennent des frontières de décision différentes. Ils conjecturent que les classifieurs dont les procédures sont "globales", c'est-à-dire s'appuyant sur l'inté-

gralité de l'espace de description (comme les réseaux de neurones et les arbres de décision), sont plus propices à l'utilisation du ECOC que ne le sont les classifieurs qui travaillent plus "localement" dans cet espace (comme les k-plus proches voisins par exemple).

1.4.4 Boosting

Le nom **Boosting** désigne un principe d'apprentissage plus qu'une méthode d'EoC et constitue par conséquent une famille de plusieurs algorithmes. Le principe de base est de spécialiser progressivement les classifieurs de l'ensemble de façon itérative, et de combiner ensuite chacun des classifieurs obtenus à chaque itération. Typiquement, il s'agit à l'itération k de concentrer l'apprentissage du classifieur h_k sur les erreurs des classifieurs h_{k-1}, h_{k-2} ...,h_1, obtenus aux itérations précédentes. Dans le principe de Boosting, cet objectif est réalisé à l'aide d'une pondération des données d'apprentissage. À la première itération, toutes les données de l'ensemble d'apprentissage se voient attribuer un poids identique, tel que la somme de ces poids soit égale à 1. Puis un classifieur dit "faible" est construit sur cet ensemble, pour lequel le taux d'erreur en apprentissage est non nul, mais nécessairement inférieur à 0.50. La pondération est ensuite mise à jour sur la base de ces erreurs de prédiction, de sorte d'augmenter le poids des données d'apprentissage qui ont été mal classées par ce classifieur, tout en diminuant simultanément les poids des données bien classées. Ainsi, on spécialise progressivement les classifieurs pour qu'ils se concentrent sur l'apprentissage des données précédemment mal classées.

Une des conditions pour que ce processus fonctionne est que les classifieurs utilisés comme classifieurs élémentaires soient "faibles". Dans ce cadre, on appelle classifieur faible un classifieur au moins aussi bon qu'une prédiction complètement aléatoire. C'est à dire que les classifieurs produits par ce type d'algorithme doivent en moyenne obtenir un taux de bonnes prédictions supérieur ou égal à 50%. L'apparition des approches de boosting répond alors à la question : peut-on rendre "fort" n'importe quel apprenant "faible" ? peut-on transformer un apprenant faible $\mathcal{L}$ en un apprenant fort ?

La première réponse formelle à ce problème a été apportée par Schapire dans [Schapire 1990] où il prouve le résultat surprenant de l'équivalence des notions d'apprenabilité forte et faible. En d'autres termes, s'il est vrai qu'un apprenant fort est toujours par définition également un apprenant faible, on peut prouver qu'il est toujours possible, théoriquement, d'obtenir un apprenant fort à partir d'un apprenant faible. Il en déduit un premier algorithme de boosting permettant d'obtenir un apprenant fort à partir d'un apprenant faible, c'est-à-dire un algorithme d'apprentissage qui combine plusieurs classifieurs obtenus avec un algorithme d'apprentissage faible. Cet algorithme est récursif et construit un système multi-classifieurs à l'aide de trois niveaux de combinaison. La figure 1.10 schématise l'architecture de cette combinaison de classifieurs. Le point clé de cet algorithme est que l'apprentissage de chaque classifieur faible est basé sur une distribution différente des données d'ap-

prentissage, générée à l'aide des prédictions des classifieurs précédemment induits.

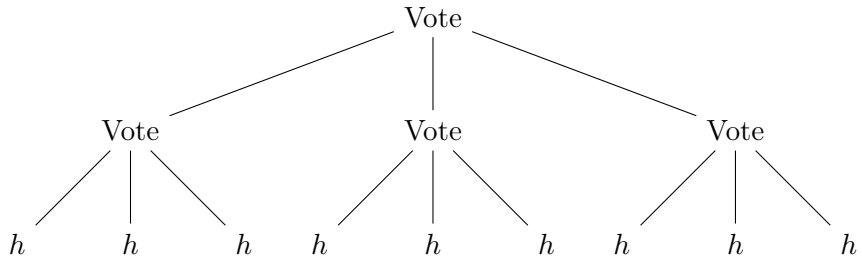

FIG. 1.10: Illustration du premier algorithme de boosting proposé par Schapire dans [Schapire 1990]. Chaque nœud "h" est un classifieur faible, et chaque nœud "Vote" est un opérateur de vote à la pluralité.

Après Schapire, Freund introduit à son tour son algorithme de boosting dans [Freund 1990], reprenant le principe global de l'algorithme de Schapire mais de façon non plus récursive mais itérative. En 1996 ensuite, les deux auteurs présentent l'algorithme **AdaBoost** ([Freund 1996]). La première version de cet algorithme appelé AdaBoost.M1 est décrite par l'algorithme 1.

Algorithme 1 AdaBoost.M1

Entrée : $\mathcal{L}$ un apprenant faible.
Entrée : L le nombre de classifieurs de l'ensemble final.
Entrée : T un ensemble de N données d'apprentissage.

$D_1(x_i) = \frac{1}{N}, i = 1, ..., N$ *Initialisation des poids (équiprobabilité)*
pour $t = 1, ..., L$ **faire**
  $h_t = \mathcal{L}(D_t)$ *Apprentissage de h_t*
  $\widehat{\epsilon}_t = \sum_{i:h(x_i)\neq y_i} D_t(x_i)$ *Calcul de l'erreur pondérée de h_t*
  si $\widehat{\epsilon}_t > \frac{1}{2}$ **alors**
    Stopper la boucle
  $\beta_t = \frac{\widehat{\epsilon}_t}{(1-\widehat{\epsilon}_t)}$ *Calcul du coefficient de pondération de h_t*
  pour $i = 1, ..., N$ **faire**
    si $h_t(x_i) = y_i$ **alors**
      $D_{t+1}(x_i) = \frac{D_t(x_i)}{Z_t} \times \beta_t$
    sinon
      $D_{t+1}(x_i) = \frac{D_t(x_i)}{Z_t}$
$h_c(x) = \arg max_{y\in Y} \sum_{t=1}^{L} \log \frac{1}{\beta_t} \times \mathbb{1}_{h_t(x)=y}$ *Vote pondéré*

La définition de l'apprenabilité faible impose que l'erreur d'un apprenant soit "légèrement meilleure que le hasard", ce qui peut se traduire par $\widehat{\epsilon}_t < \frac{1}{2}$, où $\widehat{\epsilon}_t$ représente l'erreur en apprentissage dans le cas de problème à deux classes. Cependant, dans le cas de problèmes à plus de deux classes, être meilleur que le hasard

ne signifie pas forcément que l'erreur de l'apprenant est tout juste inférieure à $\frac{1}{2}$. Pour un problème à c classes, prédire une classe au hasard fournit une espérance mathématique de l'erreur égale à $1 - \frac{1}{c}$. La contrainte $\widehat{\epsilon}_t < \frac{1}{2}$ peut alors s'avérer particulièrement difficile à respecter avec certains problèmes multi-classes. C'est là le principal inconvénient de ce premier algorithme : il impose $\widehat{\epsilon}_t < \frac{1}{2}$.

Pour remédier à cela, Freund et Schapire proposent dans le même article une deuxième version de l'algorithme qui se comporte de la même manière qu'AdaBoost.M1 dans le cas de problème à deux classes, mais qui corrige ces défauts dans le cas de problèmes à plus de 2 classes. Les principales différences entre cette nouvelle version et la première sont les suivantes :

(i) l'apprenant faible fournit non plus un label en prédiction mais un ensemble de labels plausibles auxquels sont associés des scores de plausibilité.

(ii) la contrainte de performance n'est plus modélisée par une simple condition sur l'erreur de prédiction telle que $\widehat{\epsilon}_t > \frac{1}{2}$. Elle est désormais basée sur une fonction de perte qui prend en compte la distribution sur l'ensemble des couples de données d'apprentissage et de labels incorrects, au lieu simplement de l'erreur basée sur la distribution des données d'apprentissage. Ainsi, l'algorithme ne se focalise pas seulement sur les données difficiles à classer, mais sur les classes à prédire.

Les premières expérimentations de Freund et Schapire dans cet article indiquent clairement que la deuxième version donne de meilleurs résultats que la première sur les problèmes multi-classes[4]. C'est particulièrement le cas lorsque les arbres de décision sont utilisés comme classifieurs élémentaires.

Il est important de noter tout de même que dans le cas de forêts boostées, les arbres de décision doivent impérativement être élagués. En effet le boosting est réalisé sur la base des erreurs de prédiction en apprentissage des classifieurs faibles. Or, sans élagage, les arbres de décision sur-apprennent les données d'apprentissage jusqu'à avoir une erreur en apprentissage nulle dans la plupart des cas.

À la suite de ces travaux de Freund et Schapire, une multitude de variantes d'AdaBoost ont été présentées et étudiées [Friedman 2000, Grandvalet 2001, Freund 2003, Sebban 2003, Torre 2004]. Nous donnons ces quelques références qui nous semblent intéressantes, mais n'abordons pas plus en détail la littérature des méthodes de Boosting. Les méthodes de Boosting sont en fait très fréquemment citées à titre de comparaison dans les travaux qui étudient les forêts aléatoires, puisqu'en plus d'être particulièrement efficaces pour l'induction de forêts de décision, elles s'opposent aux forêts aléatoires dans l'idée principale : le Boosting s'appuie sur un principe déterministe pour la création de la diversité dans les ensembles alors que les forêts aléatoires par définition le font via les principes de randomisation.

[4] Pour les problèmes à deux classes les deux procédures sont par définition équivalentes

1.5 Conclusion

Dans ce chapitre nous avons introduit le contexte de la combinaison de classifieurs et plus particulièrement celui des ensembles de classifieurs. Nous avons détaillé chaque niveau de la catégorisation des systèmes multi-classifieurs, pour décrire, du plus large au plus spécifique, le cadre des méthodes d'induction d'EoC. Après avoir montré l'intérêt de telles méthodes, nous avons tout d'abord présenté les différentes architectures de combinaison que l'on rencontre dans la multitude de travaux sur les systèmes multi-classifieurs. Puis, en nous intéressant plus particulièrement aux combinaisons parallèles, nous avons introduit la catégorie regroupant les méthodes d'ensembles de classifieurs. Cette catégorie, sous-familles des méthodes de combinaison parallèle, est composée des approches de combinaison qui utilisent et agencent en parallèle plusieurs classifieurs de même type.

Ensuite, nous avons présenté les différents paradigmes pour l'induction de ces EoC, dont l'enjeu est de générer un ensemble de classifieurs présentant un maximum de diversité en son sein. Ces méthodes procèdent généralement à des modifications de l'ensemble d'apprentissage à chaque fois qu'un classifieur élémentaire est induit et ajouté à l'ensemble. De cette façon, ces classifieurs présentent des structures et des propriétés différentes des autres classifieurs de l'ensemble, de sorte qu'ils ne donnent pas tous systématiquement les mêmes prédictions.

Enfin, nous avons détaillé les 4 principales méthodes ou familles de méthodes d'induction d'EoC qui adoptent chacune un paradigme qui les caractérise pour générer de la diversité dans un EoC. Avec les familles de méthodes de Bagging et de Boosting, nous avons présenté deux approches pour générer un ensemble de classifieurs divers en rééchantillonnant les données d'apprentissage pour l'induction de chaque classifieur élémentaire. La méthode des Random Subspaces quant à elle modifie l'espace de description en générant des sous-espaces aléatoires différents pour chaque classifieur élémentaire. On génère de cette façon pour chacun d'eux une représentation différente du problème étudié. Enfin avec l'approche Error Correcting Output Codes, on manipule les étiquettes des classes de sorte que chaque classifieur élémentaire traite un sous-problème différent à chaque fois.

Toutes ces approches ont beaucoup été testées et éprouvées dans le cadre d'induction de forêts de décision. Les arbres de décision, utilisés dans ce cas comme classifieurs élémentaires, sont des classifieurs qui se prêtent particulièrement bien à ce type d'utilisation en combinaison parallèle, car leur instabilité permet facilement de créer de la diversité dans un ensemble. Dans le chapitre suivant nous expliquons plus en détails les raisons à cela. Nous abordons également plus particulièrement une catégorie de forêts de décision qui s'appuie sur l'utilisation de l'aléatoire pour générer de la diversité dans les ensembles d'arbres. Ces méthodes, appelées forêts aléatoires, font l'objet depuis quelques années d'un intérêt particulier, puisqu'elles se sont montrées particulièrement efficaces dans les derniers travaux qui les ont expérimentées. Nous présentons en détails dans le chapitre suivant le formalisme et les principaux aspects théoriques et pratiques qui définissent le cadre de ces méthodes.

Chapitre 2

Forêts Aléatoires

Sommaire

2.1 Introduction

Dans ce chapitre, nous nous intéressons plus en détail aux méthodes de forêts de décision. Ces méthodes ont peu à peu fait leur apparition dans le paysage de l'apprentissage automatique avec la popularisation des méthodes de combinaison de classifieurs. La principale raison à cela est que les arbres de décision sont des classifieurs qui se prêtent particulièrement bien à une utilisation dans le cadre de systèmes multi-classifieurs. Ces classifieurs présentent en effet quelques unes des limitations que nous avons présentées en section 1.2 du précédent chapitre, qu'il est intéressant d'exploiter dans des EoC.

Ils sont tout d'abord basés sur un découpage heuristique de l'espace de description. La construction de la structure de décision d'un arbre est réalisée par partitionnements récursifs de cet espace, ce qui rend les décisions finales fortement dépendantes des découpages effectués en amont. L'espace de recherche des structures possibles d'arbres de decision est alors fortement restreint par ces dépendances. En perturbant ces niveaux successifs de partitionnement, il est possible de créer une variété bien plus large de structures, que l'on peut alors exploiter au sein d'un EoC. De plus, bien qu'un grand nombre de techniques existe pour compenser la tendance des arbres de décision à sur-apprendre les données d'apprentissage (*cf.* section 2.2), ils n'en reste pas moins des classifieurs discriminants, qui possèdent de faibles capacités en généralisation. De cela découle une forte instabilité qu'il est alors intéressant d'exploiter dans le cadre d'EoC, comme nous le verrons plus en détail dans ce chapitre.

Pour ces raisons, les méthodes de forêts de décision se sont montrées particulièrement efficaces et compétitives avec les méthodes d'apprentissage les plus performantes. En particulier, les méthodes qui utilisent l'aléatoire pour générer de la diversité dans les ensembles d'arbres se sont multipliées. Récemment, Breiman a introduit un cadre formel pour cette catégorie de méthodes qu'il a nommée *Random Forest* (RF) que nous traduisons en français par forêts aléatoires ([Breiman 2001]). Ce formalisme, s'il n'est pas à l'origine de l'utilisation de ces méthodes, permet tout de même d'en dresser un cadre théorique générique et donne notamment quelques éléments importants pour comprendre et manipuler ces principes de randomisation. Dans ce chapitre, nous allons entrer dans les détails de ce formalisme, expliquer quels sont ces éléments, et dresser un panorama des différentes méthodes d'induction de forêts aléatoires existant dans la littérature.

Dans une première section, nous abordons l'induction automatique d'arbres de décision. Pour bien comprendre les mécanismes de fonctionnement des RF, il est nécessaire d'expliquer ce processus d'induction et de s'attarder en particulier sur la problématique de partitionnement des nœuds. C'est ce que nous faisons en premier lieu. Ensuite, nous dressons un bilan des travaux qui se sont intéressés aux ensembles d'arbres et qui ont mené à ce formalisme. Dans la quatrième section, nous détaillons la définition des RF et expliquons de façon plus précise comment

ces méthodes fonctionnent. Nous en donnons ensuite quelques résultats théoriques importants, qui fournissent les premiers éléments de compréhension de leur mécanismes de fonctionnement. Nous expliquons notamment ce que sont les propriétés de *force* et de *corrélation*, sur lesquelles nous nous appuyons à plusieurs reprises dans ces travaux de thèse. Nous présentons dans la sixième section un grand nombre de méthodes d'induction de forêts aléatoires que l'on peut trouver dans la littérature. Enfin, nous discutons les inconvénients de l'algorithme de référence pour l'induction de forêts aléatoires, appelé Forest-RI, pour introduire notamment les problématiques auxquelles nous nous sommes intéressé dans ces travaux de thèse. Nous expliquons plus particulièrement les problèmes de mise en œuvre et de paramétrisation, ainsi que les raisons pour lesquelles il nous semble important d'y répondre.

2.2 Arbres de décision

L'arbre de décision est un outil largement répandu dans les domaines des statistiques, de l'ingénierie, de la théorie de la décision, ou encore de l'apprentissage automatique ; et cette popularité repose en grande partie sur sa simplicité.

Dans son état de l'art sur les arbres de décision, réalisé dans le cadre de ses travaux de thèse [Murthy 1997], Murthy résume ce que sont les arbres de décisions en expliquant qu'ils constituent un moyen de représenter un ensemble de régles qui sous-tendent des données par une structure hiérarchique, en partitionnant de façon récursive sa population (*cf.* figure 2.1 [Rakotomalala 2005]).

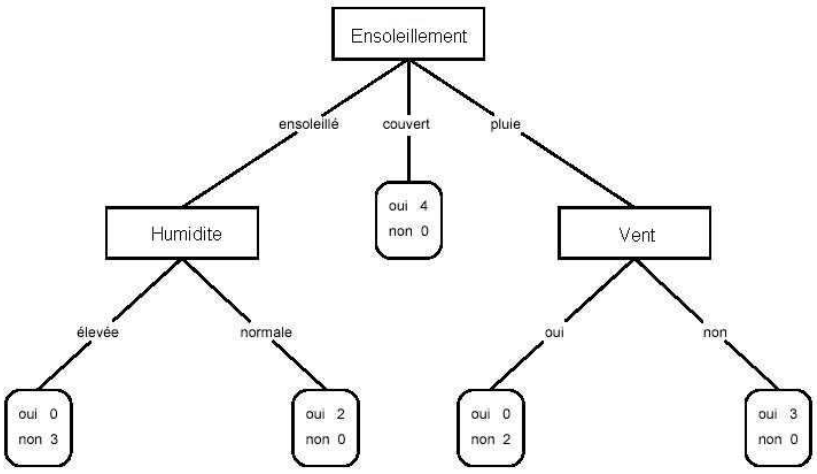

FIG. 2.1: Exemple d'un arbre de décision construit à partir de l'exemple de Quinlan [Quinlan 1993] : il s'agit de prédire "oui" ou "non" en réponse à la question "Le temps permet-il de jouer à l'extérieur ?".

Il existe une très grande variété de travaux liés à l'induction automatique d'arbres de décision à partir d'un jeu de données. Depuis les années 90, quelques états de l'art

ont été publiés qui répertiorient et résument l'ensemble de ces travaux [Murthy 1997, Murthy 1998, Brostaux 2005, Breslow 1997, Loh 1997]. De ces états de l'art, on distingue trois principales problématiques respectivement liées au choix de la règle de partitionnement pour chaque nœud, à la taille des arbres, et aux décisions prises aux nœuds terminaux. Nous faisons dans cette section un rapide panorama de ces trois problématiques et des solutions communément mises en place pour y répondre. Nous nous attardons plus particulièrement sur le choix de la régle de partitionnement qui est un point essentiel pour la compréhension du fonctionnement des forêts aléatoires.

2.2.1 Phase d'apprentissage

La phase d'apprentissage d'un arbre de décision consiste essentiellement à en construire la structure hiérarchique. À l'instar de Rakotomalala [Rakotomalala 2005], nous pouvons appréhender les étapes de cette phase à l'aide d'un exemple extrait de l'ouvrage de Quinlan [Quinlan 1993]. Ce dernier a par ailleurs été très actif dans ce domaine, et a notamment mis au point les méthodes d'induction d'arbres **ID3** et **C4.5** qui sont deux algorithmes de référence en la matière. Son exemple "weather", dont les données sont présentées dans le tableau 2.1, a également été très fréquemment cité dans la littérature pour illustrer les problématiques et les méthodes de construction d'arbres de décision.

Numéro	Ensoleillement	Température (°F)	Humidité (%)	Vent	Jouer
1	soleil	75	70	oui	oui
2	soleil	80	90	oui	non
3	soleil	85	85	non	non
4	soleil	72	95	non	non
5	soleil	69	70	non	oui
6	couvert	72	90	oui	oui
7	couvert	83	78	non	oui
8	couvert	64	65	oui	oui
9	couvert	81	75	non	oui
10	pluie	71	80	oui	non
11	pluie	65	70	oui	non
12	pluie	75	80	non	oui
13	pluie	68	80	non	oui
14	pluie	70	96	non	oui

TAB. 2.1: Description des données de l'exemple "weather" [Quinlan 1993]

Pour cet exemple, on dispose d'une population de 14 observations (individus), pour lesquelles il s'agit d'expliquer le comportement se rapportant à l'action de "jouer en extérieur", à partir d'un ensemble de prévisions

météorologiques (*cf.* tableau 2.1). Plus concrètement, on souhaite par exemple pouvoir prédire "oui" ou "non" en réponse à la question : "Le temps nous permet-il de jouer en extérieur ?".

En considérant que cet ensemble dans son intégralité constitue la population de la "racine" de l'arbre — terme qui désigne le nœud initial — on retrouve effectivement dans la figure 2.1 la répartition du tableau 2.1, à savoir 9 individus "oui" et 5 "non" au total.

À chaque étape de la construction il s'agit d'ajouter de nouveaux nœuds à la structure, et de faire ainsi "grandir"[1] l'arbre de décision. Le rôle de chaque nœud va être de diviser la population d'individus qui le constitue en plusieurs sous-groupes plus homogénes (*cf.* figure 2.2 [Brostaux 2005]). Pour cela on associe à chacun une règle de partitionnement qui permettra de répartir les individus dans l'un ou l'autre des sous-groupes en question.

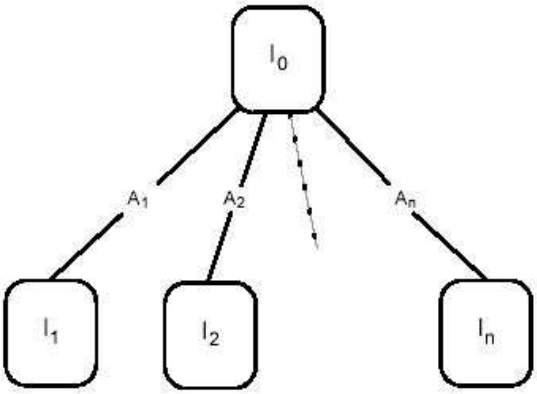

FIG. 2.2: Partition d'un ensemble d'individus au sein d'un arbre de décision. Dans cet exemple I_0 représente la population du nœud courant et $I_1...I_n$ les sous-ensembles résultant de la partition. $A_1...A_n$ désignent quant à eux les évènements qui conditionnent la répartition dans les nœuds fils.

En se positionnant sur un nœud, cette règle est déterminée de la façon suivante :

1. on choisit l'une des variables d'entrée correspondant aux attributs (ou caractéristiques) de chaque individu. Elle servira alors de critère de partitionnement.
2. en fonction des valeurs que prend cette variable au sein de la population du nœud, un ou plusieurs tests sont définis à l'aide desquels on peut réaliser le partitionnement.
3. chaque individu est réparti dans les sous-ensembles, en le soumettant aux tests de partitionnement.

[1] le verbe anglais "to grow" est très souvent utilisé pour désigner l'action de construire un arbre de décision

La façon dont la caractéristique est choisie et le calcul de la valeur de coupure sont détaillés un peu plus loin dans cette section.

Dans notre exemple, le premier partitionnement ainsi réalisé, utilise la variable "Ensoleillement" pour définir la règle de partitionnement. 3 modalités sont alors possibles : "soleil", "couvert" et "pluie". De ce fait, 3 nouveaux nœuds sont créés, chacun correspondant aux sous-groupes en relation avec ces modalités. On parlera de nœud "père" et nœuds "fils" pour représenter la relation entre la racine et ces trois nouveaux nœuds.

Il résulte donc de ce partitionnement intermédiaire la naissance de nouvelles **branches**, partant du nœud courant vers autant de nouveaux nœuds qu'il y a de partitions créées. On reprend alors le même processus avec tous ces nouveaux nœuds jusqu'à ce qu'il ne reste plus que des **feuilles** au bout de chaque **branche**.

Le terme **feuille** désigne ici un nœud terminal. Il s'agit d'un nœud qui ne possède pas de fils et dont la population est considérée comme suffisamment homogène pour décider de ne pas la partitionner à nouveau. On dit d'une feuille qu'elle est **pure** si sa population d'individus n'appartient qu'à une seule classe. La notion de pureté ou d'impureté est souvent le critère principal d'arrêt de la croissance d'un arbre car si une feuille est pure, il n'est pas nécessaire de continuer à partitionner sa population. Cependant on rencontre souvent dans la littérature l'utilisation d'un critère d'arrêt supplémentaire. Il peut être en effet nécessaire d'interrompre la croissance d'un arbre qui deviendrait trop complexe, sans même attendre de n'obtenir que des feuilles pures. De la même manière il existe des techniques **d'élagage**[2] permettant de réduire la taille et la complexité d'un arbre après l'avoir laissé atteindre sa taille maximale ([Breiman 1984, Quinlan 1986b, Mingers 1989a, Quinlan 1993, Malerba 1996, Esposito 1997, Breslow 1997, Sebban 2000]). L'élagage, quel qu'en soit le procédé, consiste à supprimer *a posteriori* des branches de l'arbre jugées inutiles. Ces techniques sont généralement mises en place dans les algorithmes d'induction pour améliorer les capacités en généralisation des arbres de décision. Si aucun élagage n'est réalisé *a posteriori* un arbre sur-apprend inévitablement les données d'apprentissage et est par conséquent très mauvais en généralisation.

Nous le verrons dans la suite de ce chapitre, cette faiblesse des arbres de décisions, *i.e.* leur faible capacité à généraliser, peut être gommée par la combinaison de plusieurs d'entre eux. Pour cela, les EoC tirent généralement profit de l'instabilité des arbres, en manipulant les données d'apprentissage pour chacun d'eux. Si les arbres d'un comité sont construits sur des ensembles d'apprentissage différents les uns des autres, les structures des arbres en seront d'autant plus modifiées qu'ils sur-apprendront ces ensembles d'apprentissage. La conséquence de cela est qu'il est préfèrable de ne pas utiliser de techniques d'élagage dans le cadre d'ensembles d'arbres de décision. Par ailleurs, la grande majorité des algorithmes de forêts aléatoires n'utilise aucune méthode d'élagage, et laisse

[2] en anglais **pruning**

l'induction de l'arbre se poursuivre jusqu'à n'obtenir que des feuilles pures. C'est la raison pour laquelle nous n'entrons pas plus en détail ici dans les différentes méthodes d'élagage existant dans la littérature, mais nous donnons quelques références de travaux qui se sont exclusivement intéressés à cette problématique ([Quinlan 1986b, Mingers 1989a, Malerba 1996, Esposito 1997, Breslow 1997]).

En ce qui concerne notre exemple, aucun test d'arrêt supplémentaire n'a été utilisé ; l'arbre de la figure 2.1 a grandi jusqu'à atteindre sa taille maximale. Ainsi, la racine a donné naissance à trois nœuds fils, parmi lesquels un d'entre eux est une feuille, car c'est un nœud pur du point de vue de la variable à prédire.

Enfin, la dernière étape de la construction d'un arbre va consister à attribuer une conclusion à chacune de ses feuilles. Si aucun test d'arrêt supplémentaire n'a été mis en place, et si aucun élagage n'a été réalisé (*i.e.* si toutes les feuilles sont pures), les conclusions pour chaque feuille sont alors toutes évidentes. Cependant, dans le cas contraire, il faut décider d'une classe à attribuer pour chaque cheminement de l'arbre. Généralement, la prédiction attribuée à une feuille qui n'est pas pure correspond à la classe la plus représentée parmi les données d'apprentissage qui y figurent ([Quinlan 1986b, Mingers 1989a, Esposito 1997]). Cependant, d'autres techniques existent. Elles sont notamment mises en place lorsqu'un critère d'arrêt est utilisé au cours de l'induction de l'arbre ou lorsqu'une procédure d'élagage est mise en œuvre. À nouveau ici, il n'est pas nécessaire, lorsque qu'on se place dans le cadre de forêts, d'implémenter ce type de procédé dans la mesure où, comme nous l'avons mentionné, les arbres sont systématiquement induits jusqu'à leur taille maximale. Pour plus de détails sur ces procédés, nous reportons le lecteur aux états de l'art sur les arbres de décision ou sur les méthodes d'élagage plus particulièrement, que nous avons déjà cités précédemment dans cette section.

En résumé, la construction d'un arbre de décision se compose des étapes suivantes : (à partir de la racine de l'arbre)

1. Choix d'une variable de partitionnement parmi les attributs qui décrivent les données d'apprentissage.
2. Choix d'une ou de plusieurs valeurs de coupure de cette variable pour définir la partition.
3. Recommencer les étapes 1 et 2 avec chacun des nœuds fils qui ne remplissent pas les critères pour devenir des feuilles (Si tous les nœuds fils sont des feuilles pures, la branche courante a atteint sa taille maximale).
4. Elaguer si besoin l'arbre.
5. Affecter à chaque feuille une conclusion.

Si, comme nous venons de l'expliquer, les étapes d'élagage et d'attribution des conclusions aux feuilles de l'arbre ne sont pas cruciales pour l'induction d'arbres

dans les forêts aléatoires, le choix de la règle de partitionnement à chaque nœud en revanche, joue un rôle important notamment pour mieux comprendre les enjeux de l'utilisation de principes de randomisation pour l'induction de forêts aléatoires (*cf.* section 2.6.1). C'est pourquoi nous entrons maintenant un peu plus dans les détails de cette étape.

2.2.2 Évaluation des règles de partitionnement

La plupart des méthodes d'induction d'arbres s'appuient sur le même procédé quant à la sélection des caractéristiques de partitionnement à chaque nœud : pour chaque caractéristique candidate, on réalise le partitionnement de la population et on calcule ensuite un critère de qualité. La caractéristique retenue est alors celle qui optimise ce critère. Les méthodes vont donc se distinguer les unes des autres principalement par ce critère d'évaluation utilisé pour juger de la qualité d'une partition.

Choisir le bon critère pour sélectionner la bonne règle de partitionnement est une problématique sur laquelle se sont penchés de nombreux travaux depuis l'introduction des premiers algorithmes d'induction d'arbres par Breiman *et al.* en 1984 [Breiman 1984] et par Quinlan en 1986 [Quinlan 1986a]. Murthy, dans son état de l'art sur l'induction d'arbres de décision, liste un grand nombre de références se rapportant à ce sujet spécifique de la sélection de caractéristiques pour le partitionnement [Murthy 1998].

Mingers, en 1989 [Mingers 1989b], répertorie et compare 11 des principaux critères utilisés pour l'évaluation des règles de partitionnement dans des algorithmes d'induction d'arbres. Il conclut que le choix aléatoire d'une caractéristique de partitionnement semble étonnamment aussi efficace que la mise en place de n'importe quel autre critère.

Buntine et Niblett en 1992 [Buntine 1992], argumentant le manque de rigueur du protocole d'expérimentation de Mingers, proposent une autre comparaison empirique de 4 de ces 11 mesures, sur des tests plus hétéroclites et plus complets et suivant leur propre protocole. Ils concluent différemment de Mingers, en présentant des résultats pour lesquels la sélection aléatoire de l'attribut présente des résultats significativement moins bons que les autres méthodes.

De notre point de vue, si l'on peut raisonnablement conclure de ces travaux que le choix d'un critère d'évaluation est nécessaire pour un algorithme d'induction d'arbres de décision, ils fournissent cependant assez peu de précisions sur les avantages et inconvénients d'utiliser l'un ou l'autre de ces critères. Murthy explique dans [Murthy 1998] que jusqu'à maintenant, on rencontre la même conclusion dans la quasi-totalité des publications sur le sujet, à savoir qu'aucun critère ne se distingue des autres par de meilleures performances. Si certains d'entre eux se comportent mieux que d'autres pour des contextes d'application particuliers, de façon générale, aucun ne l'emporte réellement dans des expérimentations plus poussées. Breiman *et al.* statuent dans [Breiman 1984] que le choix du critère d'évaluation n'est pas décisif

vis à vis des performances de l'arbre de décision, et que tout se joue davantage dans la façon d'élaguer.

Les travaux de Mingers [Mingers 1989b] permettent cependant de mettre en évidence que certains critères engendrent des structures beaucoup plus complexes que d'autres et que le nombre de niveaux et de feuilles de l'arbre varie fortement selon les cas.

Dans la majorité des cas, ce critère d'évalutation des règles de partitionnement est basé sur une mesure **d'impureté**. Si l'on parlera de l'impureté d'un nœud, ou d'une population, il s'agit en fait de l'impureté d'une distribution $P = (P(y_1), P(y_2), ..., P(y_c))$ des labels de classe dans une population d'individus. Elle peut être mesurée de différentes manières mais doit toujours respecter la propriété suivante : elle doit être minimale lorsque tous les individus de la population d'un nœud appartiennent à une même classe, *i.e.* possèdent le même label, et maximale lorsque chaque classe est représentée par le même nombre d'individus dans la population.

Murthy mentionne dans [Murthy 1998] trois catégories non exclusives de critères : les critères issus de la théorie de l'information, tels que l'Entropie de Shannon, les critères basés sur des distances, parmi lesquels il range par exemple l'indice de Gini, et les critères basés sur les dépendances, comme par exemple les mesures statistiques du type test du χ^2.

Dans les paragraphes suivants, nous détaillons les trois principaux critères qui sont utilisés dans le cadre des forêts aléatoires. Il s'agit en fait des trois critères utilisés dans les algorithmes d'induction d'arbres de décision ID3, C4.5 et CART, qui sont les trois principaux algorithmes implémentés dans les divers logiciels d'apprentissage automatique.

Gain d'information

Le gain d'information est le critère utilisé par Quinlan pour son algorithme d'induction d'arbres appelé ID3 (pour Induction Decision Tree) [Quinlan 1986a]. D'après sa définition, l'impureté peut se calculer différemment d'un critère à l'autre. Elle est, dans le cas du **Gain d'Information**, mesurée par la quantité d'information nécessaire pour déterminer la classe d'un individu d'une population T, estimée par la formule de l'Entropie de Shannon, où les probabilités de classes sont estimées par les fréquences observées correspondantes :

$$I(T) = \sum_{j=1}^{c} -\frac{n_{j.}}{n_{..}} \log_2 \frac{n_{j.}}{n_{..}} \tag{2.1}$$

où n_{ji} représente l'effectif de la classe j au nœud i ; le point symbolise alors la sommation selon l'indice correspondant.

Par exemple, pour des populations T qui présentent les distributions P suivantes, on a :

- $P = (0.5, 0.5)$ alors $I(T) = 1$
- $P = (0.67, 0.33)$ alors $I(T) = 0.92$
- $P = (1.0, 0.0)$ alors $I(T) = 0$

Et pour le cas de notre exemple "weather", si l'on considère T comme étant la population initiale de notre base d'apprentissage on obtient :

- $P = \left(\frac{9}{14}, \frac{5}{14}\right)$ qui nous est donc donné par les effectifs des individus de chaque classe au sein de la population.

On a alors $I(T) = 0.94$

Considérons maintenant que l'on partitionne la population T en m_i sous-groupes d'individus notés $T_1, T_2, ..., T_{m_i}$, selon les valeurs de l'attribut A_i (m_i étant le nombre de modalités de l'attribut A_i). L'information nécessaire pour identifier la classe d'un élément quelconque de T devient la moyenne pondérée des informations nécessaires pour identifier la classe d'un élément de chaque T_k :

$$I(T, A_i) = \sum_{k=1}^{m_i} \frac{n_{.k}}{n_{..}} I(T_k) \tag{2.2}$$

L'impureté ainsi calculée, le **Gain d'information**, noté $Gain(T, A_i)$ est défini par :

$$Gain(T, A_i) = \Delta I(T, A_i) = I(T) - I(T, A_i) \tag{2.3}$$

Si l'on considère toujours la population constituée des 14 individus de notre exemple "weather" présentés dans le tableau 2.1, en choisissant la caractéristique "Ensoleillement" pour la règle de partitionnement, on obtient :

$$I(T, "Ensoleillement") = \frac{5}{14} \times I\left(\frac{3}{5}, \frac{2}{5}\right) + \frac{4}{14} \times I\left(\frac{4}{4}, 0\right) + \frac{5}{14} \times I\left(\frac{3}{5}, \frac{2}{5}\right)$$

$$I(T, "Ensoleillement") = 0.694$$

Et donc

$$Gain(T, "Ensoleillement") = I(T) - I(T, "Ensoleillement")$$

$$Gain(T, "Ensoleillement") = 0.94 - 0.694 = 0.246$$

En revanche si on souhaite utiliser l'attribut "Vent" pour la règle de partitionnement, on obtient :

$$I(T, "Vent") = 0.892$$

$$Gain(T, "Vent") = 0.048$$

On constate donc que selon le critère de Gain d'information, la caractéristique "Ensoleillement" permet d'obtenir une meilleure partition que la caractéristique "Vent" pour le partitionnement à la racine de l'arbre.

Gain Ratio

Une correction a été apportée par la suite au Gain d'information pour compenser la tendance de ce critère à favoriser les caractéristiques présentant de nombreuses modalités. On parlera de **Gain d'Information Relatif** ou de **Gain Ratio** :

$$IV(T, A_i) = -\sum_{k=1}^{m_i} \frac{n_{.k}}{n_{..}} \log_2 \frac{n_{.k}}{n_{..}} \tag{2.4}$$

$$Gain\ Ratio(T, A_i) = \frac{Gain(T, A_i)}{IV(T, A_i)} \tag{2.5}$$

où IV représente la fonction qui fournit l'information intrinsèque de l'attribut A_i et *Gain Ratio* son Gain d'Information Relatif.

Reprenons à nouveau notre exemple :

$$IV(T, "Ensoleillement") = -\frac{5}{14} \times \log_2 \frac{5}{14} + \frac{4}{14} \times \log_2 \frac{4}{14} + \frac{5}{14} \times \log_2 \frac{5}{14}$$

$$IV(T, "Ensoleillement") = 1.577$$

Et donc

$$Gain\ Ratio(T, "Ensoleillement") = \frac{0.246}{1.577} = 0.156$$

Puis

$$IV(T, "Vent") = 0.985$$

$$Gain\ Ratio(T, "Vent") = \frac{0.048}{0.985} = 0.049$$

Indice de Gini

Ce critère est celui qui est introduit et utilisé par Breiman *et al.* dans la méthode d'induction d'arbres **Classification And Regression Tree (CART ou C&RT)** [Breiman 1984]. C'est également le critère que Breiman a décidé d'utiliser dans ses algorithmes d'induction de forêts aléatoires.

L'impureté d'une population est alors mesurée par la formule suivante :

$$I(T) = \sum_{j=1}^{c} \frac{n_{j.}}{n_{..}} \left(1 - \frac{n_{j.}}{n_{..}}\right) \tag{2.6}$$

que l'on rencontre également dans la littérature sous la forme :

$$I(T) = 1 - \sum_{j=1}^{c} \left(\frac{n_{j.}}{n_{..}}\right)^2 \tag{2.7}$$

Cette mesure peut être vue comme l'erreur de classification attendue si la classe prédite était choisie aléatoirement en suivant la distribution des probabilités de classes approximées par les effectifs du nœud T courant.

De la même façon que pour le Gain d'Information, elle permet alors de calculer le gain d'impureté engendré par le partitionnement avec une certaine caractéristique :

$$\Delta I(T, A_i) = I(T) - \sum_{k=1}^{m_i} \frac{n_{.k}}{n_{..}} I(T_k) \tag{2.8}$$

avec également de la même manière T_k représentant la population du k-ième nœud fils, selon la valeur de la caractéristique $A_i = k$.

Reprenons l'exemple précédent pour une règle de partitionnement utilisant la caractéristique "Ensoleillement" :

$$I(T) = \frac{5}{14} \times \left(1 - \frac{5}{14}\right) + \frac{9}{14} \times \left(1 - \frac{9}{14}\right) = 0.23 + 0.23 = 0.46$$

$$\Delta I(T, "Ensoleillement") = 0.46 - \left(\frac{5}{14} \times I\left(\frac{3}{5}, \frac{2}{5}\right) + \frac{4}{14} \times I\left(\frac{4}{4}, 0\right) + \frac{5}{14} \times I\left(\frac{3}{5}, \frac{2}{5}\right)\right)$$

$$\Delta I(T, "Ensoleillement") = 0.46 - \left(\frac{5}{14} \times 0.48 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.48\right) = 0.117$$

Puis

$$\Delta I(T, "Vent") = 0.46 - \left(\frac{6}{14} \times I\left(\frac{3}{6}, \frac{3}{6}\right) + \frac{8}{14} \times I\left(\frac{2}{8}, \frac{6}{8}\right)\right)$$

$$\Delta I(T, "Vent") = 0.46 - \left(\frac{6}{14} \times 0.5 + \frac{8}{14} \times 0.375\right) = 0.031$$

Là encore la valeur de l'indice de Gini pour la caractéristique "Vent" est bien plus faible que celle pour la caractéristique "Ensoleillement".

On constate ici que quel que soit le critère utilisé, la règle de partitionnement qui maximise ce critère est toujours la même. Dans notre exemple, c'est à chaque fois la caractéristique "Ensoleillement" qui est sélectionnée en premier. Si on poursuit ces calculs pour l'induction complète de l'arbre, on constatera que c'est le cas pour une grande majorité des règles de partitionnement, particulièrement aux premiers niveaux de l'arbre, et qu'en définitive tous les critères tendront à fournir plus ou moins les mêmes structures, ou tout du moins les mêmes feuilles pures. C'est la raison pour laquelle aucun des critères existants ne se distinguent réellement des autres, comme l'ont conclu la plupart des travaux sur le sujet ([Mingers 1989b, Buntine 1992, Liu 1994]). Les faibles différences que l'on pourrait constater dans les arbres d'un critère à un autre ne sont pas suffisantes pour obtenir des différences significatives dans les résultats. Ce qui conditionne réellement les structures des arbres ne sont pas les critères d'évaluation des règles de partitionnement, mais les données d'apprentissage.

On voit alors se dessiner ici un des principaux inconvénients des arbres de décision : leur instabilité. Ces classifieurs sont en effet des classifieurs qui sur-apprennent énormément les données d'apprentissage, et qui par conséquent dépendent inévita-

blement de la représentativité de ces données vis à vis du domaine de définition du problème. Un changement, même faible, dans l'ensemble d'apprentissage, impliquera d'importants changements dans la structure de l'arbre et aura forcément des conséquences sur ses capacités en généralisation.

Cette instabilité est une des principales raisons qui ont motivé les chercheurs à s'intéresser aux ensembles d'arbres de décision. Elle est une faiblesse évidente pour un classifieur unique, mais peut *a contrario* devenir une force dans le cadre des ensembles de classifieurs, puisqu'elle permet de créer de la diversité via une manipulation des données d'apprentissage. Modifier sensiblement l'ensemble d'apprentissage pour chaque arbre de décision d'un ensemble, résultera en des différences dans leurs structures — et donc dans leurs prédictions — et se traduira par une grande diversité dans l'ensemble.

Dans la section suivante, nous dressons un bilan bibliographique des travaux sur les ensembles d'arbres de décision, pour illustrer comment les chercheurs du domaine en sont venus peu à peu à définir et utiliser ce que nous appellerons par la suite des **forêts de décision**.

2.3 De l'arbre à la forêt

Les premiers algorithmes d'induction d'arbres de décision dans le cadre de la classification ont été introduits dans les années 80, avec l'apparition notamment de l'algorithme **CHAID pour Chi-square AID** ([Kass 1980]). C'est à cette époque que l'étude de l'induction automatique des arbres de décision a pris son essor avec les importantes contributions de Breiman et Quinlan, respectivement en 1984 et 1986, qui vont chacun introduire leurs algorithmes CART ([Breiman 1984]) et ID3 ([Quinlan 1986a]). Quinlan en 1993 ([Quinlan 1993]) étendra les principes de fonctionnement d'ID3 dans son algorithme C4.5 aujourd'hui considéré avec CART comme l'un des algorithmes de référence pour l'induction automatique d'arbres de décision.

C'est à cette même période du début des années 90 que vont apparaitre les premiers travaux sur les combinaisons d'arbres de décision. Shlien en 1990 ([Shlien 1990]) propose d'utiliser une combinaison d'arbres de décision pour la reconnaissance de caractères manuscrits et d'établir un consensus entre les arbres en utilisant la théorie de Dempster-Shafer.

En 1991, Dietterich introduit la méthode des **Error Correcting Output Codes (ECOC)**, un principe d'apprentissage d'ensemble de classifieurs qui fonctionne avec tout type de classifieur binaire, mais que Dietterich expérimente dans un premier temps avec des arbres de décision ([Dietterich 1991]). Il démontre que cette méthode permet, en plus de fournir un outil intéressant pour résoudre des problèmes multi-classes avec des classifieurs binaires, d'en améliorer considérablement les performances.

En 1992, Ho met dans un premier temps en évidence le gain de performance que permettent d'obtenir des classifieurs qui utilisent plusieurs arbres de décision en combinaison parallèle ([Ho 1992]). Puis en 1995, elle est la première dans [Ho 1995] à utiliser le terme de **Decision Forest** pour désigner de façon générale les ensembles d'arbres de décision, et propose par la même occasion les prémisses des méthodes de **Random Subspaces**.

Ces premiers travaux sur les ensembles d'arbres randomisés vont amorcer par la suite une série de nouveaux principes de randomisation dans le cadre de forêts de décision. C'est le cas par exemple un an après du fameux principe de **Bagging** (pour **B**ootstrap **Agg**regat**ing**), introduit par Breiman dans [Breiman 1996]. Là aussi, il s'agit d'un principe d'apprentissage d'ensemble de classifieurs qui peut être utilisé avec tout type de classifieur élémentaire, mais dont l'efficacité est démontrée par Breiman principalement dans le cadre de combinaison d'arbres de décision.

La même année, Freund et Schapire proposent leur algorithme de Boosting **Adaboost** ([Freund 1996]), qui est depuis longtemps considéré comme le principe d'apprentissage d'ensemble de classifieurs le plus efficace. Les travaux sur le Boosting, et plus particulièrement sur cet algorithme, vont ensuite se multiplier, pour enrichir la littérature sur les ensembles d'arbres boostés.

Amit et Geman présentent également en 1996 le principe **Random Feature Selection** dans [Amit 1996], même si comme nous le verrons dans la section 2.6.1 son nom lui a été donné bien après par Breiman dans le cadre des forêts aléatoires. Il s'agit là d'un des premiers principes de randomisation des arbres de décision.

Un an plus tard, Ho publie dans [Ho 1998] une vue plus précise des méthodes Random Subspaces dont elle démontre la supériorité sur le Bagging et le Boosting dans le cadre de combinaisons d'arbres de décision sur des problèmes à grande dimension.

La même année Dietterich présente également un principe de randomisation d'arbres de décision, très proche dans le fonctionnement du Random Feature Selection d'Amit et Geman, qu'il nomme pour sa part **Random Split Selection** ([Dietterich 1998b]). Dans cet article il compare sa méthode au Bagging et au Boosting d'arbres, et montre alors que pour des données peu bruitées, Adaboost reste le principe d'apprentissage le plus performant pour les forêts de décision.

À la suite de l'introduction de tous ces principes d'induction de forêts, plusieurs travaux expérimentaux les ont confrontés les uns aux autres, pour dégager les forces et les faiblesses de chacun d'eux. On peut citer par exemple les travaux d'Amit *et al.* en 2000 ([Amit 2000]) qui comparent et analysent 7 principes d'induction d'arbres — aléatoires ou non — et introduisent par la même occasion l'acronyme **MRCL** pour **Multiple Randomized CLassifiers**, qui désigne de façon générale les principes de randomisation pour l'induction d'ensemble de classifieurs — qu'ils soient des arbres de décision ou non. Bauer et Kohavi dans [Bauer 1999] fournissent une étude conséquente sur les familles de méthodes de Bagging et de Boosting, en comparant pour chacune d'elles différentes variantes des algorithmes de référence. Ils donnent avec ces études expérimentales de nombreux éléments de compréhension

du gain de performance obtenu pour chacune de ces deux familles de méthodes et prouvent une fois de plus que le Boosting est bien plus efficace que le Bagging dans le cadre des arbres de décision.

Il s'agit pour tous ces principes d'induction d'arbres de décision de travaux antérieurs à l'article de Breiman, dans lequel il introduit le terme **Random Forest** et le formalisme qui y est associé. On peut donc constater que Breiman n'a pas "inventé" les forêts aléatoires tout seul. Il a cependant défini un cadre formel pour ce type d'approches. Dans la section suivante nous détaillons ce cadre et donnons la définition générique des méthodes de forêts aléatoires. Nous discutons cette définition pour expliquer plus précisément comment ces méthodes fonctionnent.

2.4 Définition

A ce stade, nous avons déjà introduit en grande partie les principes sous-jacents des méthodes de forêt aléatoire. Mais pour s'en faire une idée plus précise il est nécessaire d'expliquer plus en détail le formalisme qui les définit. C'est ce que nous faisons dans cette section.

Pour bien appréhender les forêts aléatoires il est intéressant de commencer par donner la définition de Leo Breiman dans son article publié en 2001 dans la revue internationale *Machine Learning*. En voici une traduction littérale :

Définition 1. *Une **Forêt Aléatoire** est un classifieur constitué d'un ensemble de classifieurs élémentaires de type **arbres de décision**, notés*

$$\{\ h(x, \Theta_k), \quad k = 1, ...L\ \}$$

où $\{\Theta_k\}$ est une famille de vecteurs aléatoires indépendants et identiquement distribués, et au sein duquel chaque arbre participe au vote de la classe la plus populaire pour une donnée d'entrée x.

Ces quelques lignes donnent les deux éléments clés qui définissent les forêts aléatoires :

1. Les forêts aléatoires sont basées sur des ensembles d'arbres de décision.
2. Une certaine "quantité" d'aléatoire est introduite dans le processus d'induction.

C'est à partir de ces deux principales idées que se dessine le spectre des méthodes d'induction de forêts aléatoires. On peut d'ailleurs aisément imaginer qu'en pratique les possibilités pour l'induction des forêts aléatoires sont nombreuses, puisque le ou les principes de randomisation utilisés pour l'induction des arbres ne sont pas contraints par cette définition. Cependant il y a quelques autres points qu'il est intéressant de commenter puisqu'ils apparaissent plus ou moins implicitement dans cet énoncé :

(i) Le premier d'entre eux concerne **l'opérateur de combinaison** : la définition semble contraindre cet opérateur à l'utilisation du vote à la pluralité — *chaque arbre participe au vote de la classe la plus populaire pour une donnée d'entrée* x. Cependant en pratique, plusieurs travaux ont élargi cette définition à l'utilisation d'autres opérateurs de combinaison qui permettent alors de mieux s'adapter à certaines spécificités de problèmes particuliers ([Robnik-Sikonja 2004, Tsymbal 2006, Breitenbach 2002]).

(ii) Le deuxième point concerne la **formalisation de la randomisation** des algorithmes d'induction de forêts aléatoires : elle se traduit par l'introduction d'un vecteur aléatoire dans la définition des classifieurs élémentaires $h(x, \Theta_k)$. L'ensemble des $\{\Theta_k\}$ forme alors une famille de vecteurs aléatoires indépendants et identiquement distribués. En pratique, cela signifie dans un premier temps que durant le processus d'induction des forêts aléatoires, chaque arbre est induit indépendamment des autres arbres de décision déjà ajoutés à la forêt (*les* $\{\Theta_k\}$ *sont indépendants*). Ensuite, cela signifie que le ou les principes de randomisation utilisés doivent être identiques pour l'induction de tous les arbres de la forêt (*les* $\{\Theta_k\}$ *sont identiquement distribués*). Les résultats théoriques en grande partie démontrés par Breiman sur la convergence [Breiman 2001] et la consistance des forêts [Breiman 2004], sont fortement basés sur cette hypothèse d'indépendance et de distribution homogène. Il n'existe d'ailleurs à notre connaissance aucun algorithme d'induction de forêts aléatoires qui sort de ce cadre.

Ces deux points en particulier sont importants car Breiman s'appuie ensuite sur ces deux hypothèses pour donner des résultats théoriques sur cette famille d'algorithmes. Il démontre notamment la convergence de l'erreur en généralisation pour un nombre croissant d'arbres dans une forêt. Il propose également une borne supérieure pour cette erreur en généralisation et introduit notamment les propriétés de *force* et de *corrélation*. Ces propriétés sont importantes pour la compréhension des mécanismes de fonctionnement des forêts. C'est pourquoi nous en parlons plus en détail dans la section suivante.

2.5 *Force* et *corrélation*

Dans [Breiman 2001], Breiman démontre la propriété importante de convergence des forêts aléatoires. Plus précisément il démontre que le taux d'erreur en généralisation, qu'il note PE^*, d'une forêt aléatoire converge nécessairement vers une valeur limite, quand le nombre d'arbres qui la composent augmente. Concrètement, cela signifie une chose importante : avec un nombre suffisamment élevé d'arbres, une forêt aléatoire évite le sur-apprentissage. C'est un résultat que l'on connaissait déjà (*cf.* sections 1.2 et 2.2), mais Breiman en donne la preuve mathématique dans le cadre générique des forêts aléatoires.

On peut alors déterminer une borne supérieure de cette valeur limite, en fonction

de deux propriétés intrinsèques des forêts :

- La première mesurant la fiabilité de la forêt, est appelée *force*[3].
- La deuxième mesurant le taux de *corrélation* des classifieurs élémentaires entre eux.

Considérons un ensemble de classifieurs $h_1(x), h_2(x), ..., h_L(x)$. Nous rappelons que X et Y représentent les domaines respectifs des vecteurs d'entrée x et de la variable de sortie y. La marge d'un tel ensemble de classifieurs est définie par :

$$mg(X,Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j) \tag{2.9}$$

où $I(.)$ est la fonction indicatrice définie par :

$$I(A) = \begin{cases} 1, & \text{si l'événement } A \text{ est vrai} \\ 0, & \text{sinon} \end{cases} \tag{2.10}$$

et où av_k est la fonction moyenne par rapport à l'indice k.

Dans le cadre d'un problème de classification, si un vote est mis en place pour combiner les décisions des classifieurs élémentaires, la valeur de la marge du classifieur permet de quantifier à quel point le nombre de votes pour la bonne classe dépasse le nombre de votes pour la "plus populaire" des classes erronées. C'est-à-dire qu'en considérant les votes de tous les classifieurs de l'ensemble, cette formule calcule la différence entre le nombre de votes pour la bonne classe, et le nombre de votes pour celle, parmi les autres classes, qui en a obtenu le plus. Donc plus cette valeur est grande et plus les prédictions sont fiables. En revanche, si la marge est négative, cela signifie que l'ensemble a attribué plus de votes à une mauvaise classe qu'à la vraie classe, et qu'il s'est donc trompé dans sa prédiction.

On en déduit ensuite la définition de l'erreur en généralisation suivante :

$$PE^* = P_{X,Y}(mg(X,Y) < 0) \tag{2.11}$$

Il s'agit d'exprimer ici la probabilité que la marge d'un ensemble de classifieurs soit négative, *i.e.* la probabilité que le classifieur combinant se trompe dans sa prédiction, quelque soit le couple (x, y).

Dans le cadre particulier d'une forêt aléatoire, on peut noter $h_k(X) = h(X, \Theta_k)$. Breiman établit alors dans un premier temps que, pour un grand nombre d'arbres de décision, suivant la loi forte des grand nombres, on obtient le théorème suivant :

Théorème 1. *Pour un nombre croissant d'arbres de décision, et pour presque sû-*

[3]Breiman désigne cette propriété par le terme Strength dans [Breiman 2001] ; nous le traduisons ici littéralement.

rement toutes les séquences $\Theta_1, ...\Theta_L$*,* PE^* *converge vers la valeur*

$$P_{X,Y}(P_\Theta(h(X,\Theta) = Y) - max_{j \neq Y} P_\Theta(h(X,\Theta) = j) < 0)$$

Cette formule exprime la probabilité globale d'une forêt aléatoire d'avoir une marge négative, quelles que soient les valeurs (x, y) de $X \times Y$, et donc la probabilité que celle-ci se trompe dans sa prédiction pour un individu quelconque de cette population.

Cependant, cette valeur étant théorique, il est nécessaire d'en trouver une estimation plus significative et calculable. Dans un second temps donc, Breiman démontre comment l'on peut établir une borne supérieure de ce terme en fonction de la *force* et de la corrélation.

Si on pose la fonction qui définit la marge d'une forêt aléatoire comme ceci :

$$mr(X,Y) = P_\Theta(h(X,\Theta) = Y) - max_{j \neq Y} P_\Theta(h(X,\Theta) = j) \tag{2.12}$$

qui représente le degré de confiance du classement établi par les arbres de cette forêt sur la population (X, Y), mesuré par la différence entre la probabilité de prédiction de la classe correcte y et la probabilité de prédiction de la classe erronée $j \neq y$ la plus populaire, on peut définir la valeur de prédiction d'un jeu d'arbres $\{h(X,\Theta)\}$ par l'espérance mathématique de cette fonction :

$$s = E_{X,Y}[mr(X,Y)] \tag{2.13}$$

Cette valeur caractérise ce que Breiman appelle la *force* d'une forêt aléatoire.

Or, en supposant que $s \geq 0$, l'inégalité de Chebychev nous donne une première estimation d'une borne supérieure de PE^* :

$$PE^* \leq \frac{var(mr)}{s^2} \tag{2.14}$$

Il s'agit donc maintenant de trouver une expression de la variance de la marge d'une forêt aléatoire plus facile à interpréter, ce que Breiman fait ensuite dans [Breiman 2001].

Il définit d'abord la fonction de marge brute d'une forêt aléatoire par :

$$rmg(\Theta, X, Y) = I(h(X,\Theta) = Y) - I(h(X,\Theta) = \hat{\jmath}(X,Y)) \tag{2.15}$$

où $\hat{\jmath}(X,Y)$ représente l'indice de la classe erronée la plus populaire, que l'on peut définir par :

$$\hat{\jmath}(X,Y) = arg\ max_{j \neq Y} P_\Theta(h(X,\Theta) = j) \tag{2.16}$$

Considérant maintenant Θ et Θ' indépendants et avec la même distribution on peut

alors noter

$$mr(X,Y)^2 = E_{\Theta,\Theta'}[rmg(\Theta,X,Y)rmg(\Theta',X,Y)] \tag{2.17}$$

On en déduit donc facilement

$$\begin{aligned} var(mr) &= E_{\Theta,\Theta'}[cov_{X,Y}rmg(\Theta,X,Y)rmg(\Theta',X,Y)] &(2.18)\\ &= E_{\Theta,\Theta'}[\rho(\Theta,\Theta')\sigma(\Theta)\sigma(\Theta')] &(2.19)\end{aligned}$$

où $\rho(\Theta,\Theta')$ est la corrélation entre $rmg(\Theta,X,Y)$ et $rmg(\Theta',X,Y)$, pour Θ et Θ' fixés, et où $\sigma(\Theta)$ est l'écart-type (ou déviation standard) de $rmg(\Theta,X,Y)$, pour Θ fixé.

De cette façon, en notant $\overline{\rho}$ comme étant la valeur moyenne de la corrélation $\rho(\Theta,\Theta')$, Breiman obtient alors le premier résultat suivant :

$$\begin{aligned} var(mr) &= \overline{\rho}(E_\Theta\sigma(\Theta))^2 &(2.20)\\ &\leq \overline{\rho}E_\Theta var(\Theta) &(2.21)\end{aligned}$$

Enfin en écrivant :

$$\begin{aligned} E_\Theta var(\Theta) &\leq E_\Theta(E_{X,Y}rmg(\Theta,X,Y))^2 - s^2 &(2.22)\\ &\leq 1 - s^2. &(2.23)\end{aligned}$$

Il conclut qu'une borne supérieure pour l'erreur en généralisation est donnée par :

$$PE^* \leq \frac{\overline{\rho}(1-s^2)}{s^2} \tag{2.24}$$

Cette propriété permet alors d'identifier deux éléments importants qui semblent conditionner la "qualité" d'une forêt aléatoire. D'abord le PE^* est d'autant plus faible que l'ensemble des classifieurs élémentaires est fiable, ce qui n'est pas surprenant intuitivement. Mais il est également d'autant plus faible que ces classifieurs élémentaires sont décorrélés en termes de prédictions. Tout l'enjeu de l'induction de forêts aléatoires réside donc dans la production d'un jeu d'arbres, les plus performants possibles individuellement, et les moins corrélés possibles.

En conclusion, bien que cette borne de l'erreur en généralisation semble assez peu précise, elle permet de fournir des éléments intéressants pour la compréhension du fonctionnement des forêts aléatoires. On comprend avec ce résultat que l'enjeu des forêts aléatoires est de construire un ensemble d'arbres de décision présentant la meilleure marge possible, mais également capable de fournir des prédictions les plus décorrélées possibles. Ce résultat n'est certes pas nouveau, mais cette démonstration le prouve et le traduit formellement dans le cadre des forêts aléatoires. Elle définit également ces deux propriétés importantes pour l'étude des forêts.

En conclusion tous ces aspects définissent un cadre formel dans lequel peuvent s'inscrire une multitude de méthodes d'induction de forêts aléatoires, qui se différencient alors les unes des autres par le principe de randomisation mis en œuvre. Dans la section suivante nous détaillons la plupart de ces algorithmes que l'on peut trouver dans la littérature.

2.6 Algorithmes d'induction de forêts aléatoires

2.6.1 Random Feature Selection (ou Random Tree)

Le principe de randomisation **Random Feature Selection** a été introduit en grande partie par Amit et Geman dans [Amit 1996] où ils proposent une méthode d'ensembles d'arbres aléatoires pour la reconnaissance d'écriture manuscrite. Breiman a ensuite repris cette idée et lui a donné son nom de Random Feature Selection. L'idée est d'introduire l'aléatoire dans le choix des règles de partitionnement à chaque nœud des arbres, de sorte que chaque règle ne soit plus choisie à partir de l'ensemble des caractéristiques disponibles, mais à partir d'un sous-ensemble de ces caractéristiques. Il s'agit plus précisément de sélectionner par tirage aléatoire sans remise un nombre K de caractéristiques et de choisir la meilleure des règles possibles utilisant ces K caractéristiques uniquement. L'algorithme d'induction d'un arbre de décision qui intègre ce procédé est généralement appelé *Random Tree*. La figure 2.3 illustre ce procédé.

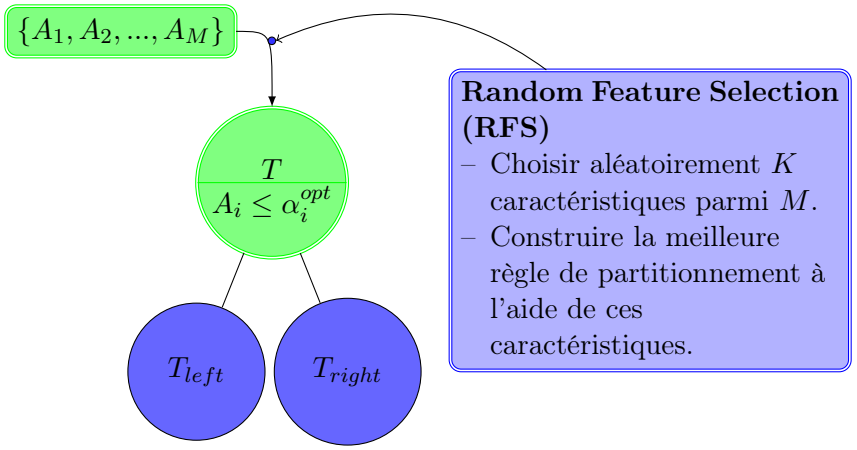

FIG. 2.3: Illustration du principe de Random Feature Selection

Cette méthode de randomisation des arbres de décision a initialement été introduite dans le cadre de problèmes de reconnaissance d'écriture manuscrite. La représentation des données utilisées dans ces travaux implique que le nombre de règles de partitionnement possibles est trop important pour qu'elles soient toutes testées à chaque nœud. L'idée d'Amit et Geman a donc été de choisir un sous-

ensemble aléatoire de ces règles et de choisir parmi elle la meilleure, afin de réduire les calculs. L'introduction de la technique de Random Feature Selection répond donc principalement à un besoin de réduction de la complexité de l'induction des arbres de décision.

C'est cependant l'utilisation de ce principe dans le cadre de forêts aléatoires qui l'a popularisé, avec notamment les travaux de Breiman et l'introduction de son algorithme Forest-RI, que nous détaillons dans la section suivante.

2.6.2 Forest-RI (ou Random Forests - Random Input)

L'algorithme d'induction de forêts aléatoires appelé Forest-RI — pour Random Forest - Random Input — est le premier de ces algorithmes. Premier en terme de chronologie puisqu'il a été introduit dans le même article ([Breiman 2001]) dans lequel Breiman a proposé le formalisme que nous venons de présenter, et premier en terme d'utilisation dans les travaux de recherche qui ont suivi cet article. Il est traditionnellement reconnu comme l'algorithme de référence pour l'induction de forêts aléatoires pour deux principales raisons : (i) Breiman, avec la collaboration d'Adèle Cutler a développé une implémentation de cet algorithme en langage Fortran 77, qu'ils ont déposé sous le nom de **Random Forest**[4] ; (ii) la plupart des logiciels d'apprentissage automatique qui implémentent une méthode de forêt aléatoire, implémentent en réalité cet algorithme qu'il propose sous le nom de "Random Forest". De sorte que la plupart des travaux de recherche qui s'intéressent à ces méthodes présentent des résultats ou des algorithmes sous le nom de Random Forest, s'agissant en réalité de classifieurs de type forêts aléatoires obtenus à l'aide de l'algorithme d'induction Forest-RI.

Cet algorithme s'appuie sur l'utilisation de deux principes de randomisation que nous avons déjà présentés précédemment : le principe de Bagging que nous avons présenté dans la section 1.4.1, et le principe de Random Feature Selection, que nous venons d'expliquer dans la section 2.6.1.

Nous détaillons avec les algorithmes 2 et 3 la procédure d'induction de forêts aléatoires avec Forest-RI.

Les expérimentations de Breiman, présentées dans [Breiman 2001], ont tout d'abord consisté à comparer les performances de Forest-RI avec celles de l'algorithme AdaBoost, jusqu'ici le plus efficace dans le cadre de l'induction de forêts de décision. Il présente les résultats obtenus sur 20 bases de données, dont 13 sont issues de l'UCI Machine Learning Repository ([Asuncion 2007]). Le protocole d'expérimentation est légèrement différent selon la nature et les caractéristiques des bases. Pour les 13 premières bases de l'UCI, 10% des données sont d'abord aléatoirement mis à l'écart pour servir ensuite de données de test, alors que les 90% restant ont servi à l'apprentissage. Deux forêts ont à chaque fois été construites à partir de ces ensembles d'apprentissage, combinant toutes les deux 100 arbres, mais pour des valeurs de K différentes. La première forêt avec $K = 1$, c'est-à-dire

[4] http ://www.stat.berkeley.edu/ breiman/RandomForests/cc_home.htm

Algorithme 2 ForestRI

Entrée : T l'ensemble d'apprentissage
Entrée : L le nombre d'arbres dans la forêt
Entrée : K le nombre de caractéristiques à sélectionner aléatoirement à chaque nœud
Sortie : $foret$ l'ensemble des arbres qui composent la forêt construite

1: **pour** l de 1 à L **faire**
2: $T_l \leftarrow$ ensemble bootstrap, dont les données sont tirées aléatoirement (avec remise) de T
3: $arbre \leftarrow$ un arbre vide, *i.e.* composé de sa racine uniquement
4: $arbre.racine \leftarrow RndTree(arbre.racine, T_l, K)$
5: $foret \leftarrow foret \bigcup arbre$
6: **retour** $foret$;

Algorithme 3 RndTree

Entrée : n le nœud courant
Entrée : T l'ensemble des données associées au nœud n
Entrée : K le nombre de caractéristiques à sélectionner aléatoirement à chaque nœud
Sortie : n le même nœud, modifié par la procédure

1: **si** n n'est pas une feuille **alors**
2: $C \leftarrow K$ caractéristiques choisies aléatoirement
3: **pour tout** $A \in C$ **faire**
4: Procédure CART pour la création et l'évaluation (critère de Gini) du partitionnement produit par A, en fonction de T
5: $partition \leftarrow$ partition qui optimise le critère Gini
6: $n.ajouterFils(partition)$
7: **pour tout** $fils \in n.noeudFils$ **faire**
8: $RndTree(fils, fils.donnes, K)$
9: **retour** n

pour un choix entièrement aléatoire des caractéristiques utilisées dans les règles de partitionnement ; la deuxième forêt avec $K = \lfloor \log_2(M) + 1) \rfloor$ (avec M le nombre de caractéristiques). L'auteur ne justifie pas ce deuxième choix, qui semble plutôt arbitraire. Concernant les 7 autres bases de données, celles-ci étant toutes divisées en deux ensembles, pour l'apprentissage et pour le test, le protocole est identique à l'exception de la séparation $90\% - 10\%$ des données.

Les résultats donnés dans [Breiman 2001] sont les taux d'erreur en généralisation obtenus sur les bases de tests. Dans le tableau récapitulatif 2.2, la première colonne présente les taux d'erreur en généralisation moyen du système Adaboost, tandis que les trois autres concernent les Forest-RI. Pour cette première colonne, 50 arbres de décision sont combinés avec la procédure Adaboost, à partir des ensembles d'apprentissage. Les taux d'erreur en généralisation sont alors calculés sur la base de 100 estimations de taux d'erreur en test, dont on considère ensuite la moyenne.

La colonne intitulée "Selection" contient des estimations du taux d'erreur en généralisation des Forest-RI, calculées en prenant le meilleur parti des deux forêts : celles avec $K = 1$ et celles avec $K = \lfloor \log_2(M) + 1) \rfloor$. Sur chaque base, Breiman calcule 100 estimations du taux d'erreur en généralisation, avec à chaque fois un nouveau sous-ensemble d'apprentissage sélectionné aléatoirement. Chacune de ces 100 estimations correspond au taux d'erreur en test de la forêt ayant présenté le meilleur des taux d'erreur out-of-bag parmi les deux. Ce sont les moyennes de ces taux d'erreur mélangés qui sont donc présentées dans cette colonne. La troisième colonne présente les moyennes des taux d'erreur en test pour les forêts construites avec $K = 1$. La dernière colonne "One tree" enfin contient la moyenne des taux d'erreur out-of-bag des arbres qui composent les forêts retenues pour la colonne "sélection".

Data set	Adaboost	Selection	Forest-RI single input	One tree
Glass	22.0	20.6	21.2	36.9
Breast cancer	3.2	2.9	2.7	6.3
Diabetes	26.6	24.2	24.3	33.1
Sonar	15.6	15.9	18.0	31.7
Vowel	4.1	3.4	3.3	30.4
Ionosphere	6.4	7.1	7.5	12.7
Vehicle	23.2	25.8	26.4	33.1
German credit	23.5	24.4	26.2	33.3
Image	1.6	2.1	2.7	6.4
Ecoli	14.8	12.8	13.0	24.5
Votes	4.8	4.1	4.6	7.4
Liver	30.7	25.1	24.7	40.6
Letters	3.4	3.5	4.7	19.8
Sat-images	8.8	8.6	10.5	17.2
Zip-code	6.2	6.3	7.8	20.6
Waveform	17.8	17.2	17.3	34.0
Twonorm	4.9	3.9	3.9	24.7
Threenorm	18.8	17.5	17.5	38.4
Ringnorm	6.9	4.9	4.9	25.7

TAB. 2.2: Taux d'erreur en test pour les Forest-RI (%) [Breiman 2001]

Le premier constat est que les forêts aléatoires présentent des résultats compétitifs avec AdaBoost pour une majorité des bases testées ; pour 12 bases sur 20, Forest-RI présente des taux d'erreur en test au moins aussi bon qu'AdaBoost et

parfois beaucoup plus faibles, et pour 4 des 8 autres bases les écarts de valeurs entre les deux méthodes ne nous semblent pas significatifs. De notre point de vue sur la base de ce tableau uniquement, Forest-RI nous semble donner de meilleurs résultats qu'AdaBoost. Cependant sans test de significativité il est difficile de se rendre compte des véritables différences de performances entre les deux méthodes. Si les taux d'erreurs de Forest-RI semblent être majoritairement plus faibles que ceux d'AdaBoost, ces écarts sont parfois très petits et ne nous permettent pas dans tous les cas de conclure de façon certaine.

Concernant la valeur de K, Breiman affirme que les performances n'y semblent pas sensibles dans la mesure où les écarts moyens des taux d'erreur obtenus avec $K = 1$ et $K = \lfloor \log_2(M) + 1 \rfloor$ ne dépassent pas les 1%. Cela nous parait pour notre part être une conclusion un peu hâtive puisque seulement deux valeurs ont été testées, et que ces valeurs sont qui plus est très proches l'une de l'autre au regard de l'ensemble des valeurs possibles pour K, à savoir tout entier entre 1 et M, M étant le nombre de caractéristiques disponibles. Pour la base Zip-code par exemple, $\lfloor \log_2(M)+1 \rfloor = 9$ alors que K peut prendre ses valeurs entre 1 et 256. Or, seules 1 et 9 sont testées alors qu'une valeur plus élevée pourrait peut-être permettre d'obtenir de meilleurs résultats.

Comme nous l'abordons en détail dans le chapitre suivant, la question de la sensibilité des performances à la valeur de ce paramètre K n'est pas aussi évidente que l'affirme Breiman. Ses conclusions sur ce point nous paraissent hâtives et nous verrons avec des expérimentations plus rigoureuses que l'influence de ce paramètre sur les performances des forêts n'est pas négligeable.

En résumé, l'algorithme Forest-RI montre dans ces expérimentations un potentiel très intéressant puisqu'il est au moins aussi bon qu'AdaBoost et parfois même meilleur, en terme de taux d'erreur. Ces résultats sont d'ailleurs confirmés par la plupart des études qui les comparent ([Banfield 2004, Boinee 2005a, Banfield 2006, Geurts 2006]). Cependant, il est difficile d'affirmer de façon catégorique que l'un ou l'autre de ces deux algorithmes est plus efficace de façon générale, car aucun test de significativité n'est utilisé pour les confronter l'un à l'autre, et les résultats sont bien souvent beaucoup trop proches pour conclure.

2.6.3 Forest-RC (ou Random Forests - Random Combinations)

La méthode **Forest-RC** est une variante de **Forest-RI**, et est également définie par Breiman dans [Breiman 2001]. Il y explique qu'elle permet de pallier certains problèmes que pourrait engendrer l'utilisation des Forest-RI, principalement lorsque la base de données d'apprentissage utilisée n'est décrite qu'à l'aide d'un nombre réduit de caractéristiques. Car dans ce cas, choisir pour K une fraction relativement importante de l'ensemble de ces caractéristiques peut entrainer une diminution de la diversité dans l'ensemble.

Pour remédier à cela donc, au lieu de sélectionner la meilleure caractéristique

parmi les K choisies au hasard, Breiman propose de calculer K combinaisons linéaires de F caractéristiques, F n'étant pas nécessairement égal à K. Plus précisément il s'agit dans un premier temps de sélectionner aléatoirement F caractéristiques, puis dans un second temps de les combiner à l'aide de K jeux de coefficients également déterminés aléatoirement dans l'intervalle $[-1, 1]$. Une fois les K combinaisons linéaires générées, on détermine celle qui propose le meilleur partitionnement. De cette manière, le phénomène évoqué plus haut est évité.

Ce principe d'utiliser des combinaisons linéaires pour les règles de partitionnement est souvent mis en œuvre dans les algorithmes d'induction automatique d'arbres de décision, indépendamment du contexte de forêt, car dans ce cas les séparations qui définissent les partitionnements à chaque nœud ne sont plus parallèles aux axes de l'espace de description, mais obliques. La figure 2.4 illustre ces deux types de situation dans le cas d'un espace de description à deux dimensions.

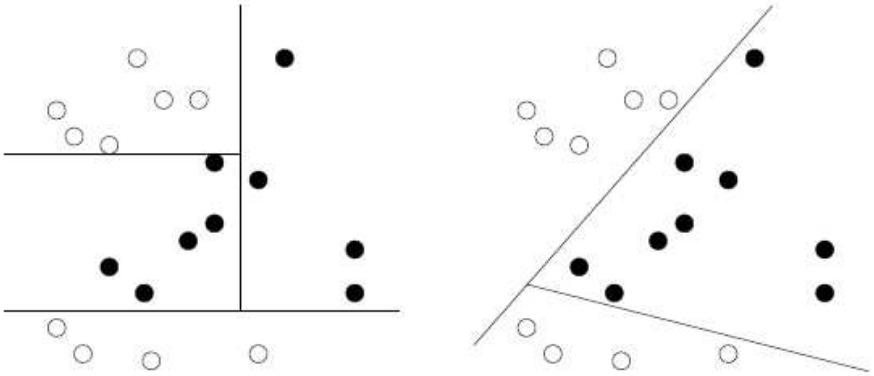

FIG. 2.4: Exemple de partitionnement d'un espace réalisé à l'aide d'un arbre de décision. Á gauche : séparations linaires parallèles aux axes. Á droite : séparations linéaires obliques. [Ho 1998]

Breiman adapte donc ce principe de séparations obliques aux forêts aléatoires en utilisant des coefficients choisis aléatoirement. Les expérimentations menées ensuite sur les Forest-RC, dont les résultats sont présentés dans le tableau 2.3, suivent le même protocole que pour les Forest-RI et sont appliquées sur les mêmes bases de données, afin de comparer les resultats des deux méthodes. Le choix des paramètres F et K semble également arbitraire pour ces tests, bien que justifié par la complexité algorithmique plus élevée. F est donc fixé à 3, et deux valeurs de K sont testées : 2 et 8.

Dans la majorité des cas (15 cas sur 19 exactement), Forest-RC présente des taux d'erreur en test inférieurs à ceux de Forest-RI. Au vu des faibles écarts dans les valeurs, on pourrait cependant se poser la question de la significativité de ce gain de performances, qui semble assez faible pour beaucoup des bases. Breiman précise qu'excepté sur les bases de données plus grandes, les Forest-RC se comportent mieux pour une valeur de K égale à 2, plutôt que 8, et qu'il est alors superflu la plupart

		Forest-RC		
Data set	Adaboost	Selection	Two features	One tree
Glass	22.0	24.4	23.5	42.4
Breast cancer	3.2	3.1	2.9	5.8
Diabetes	26.6	23.0	23.1	32.1
Sonar	15.6	13.6	13.8	31.7
Vowel	4.1	3.3	3.3	30.4
Ionosphere	6.4	5.5	5.7	14.2
Vehicle	23.2	23.1	22.8	39.1
German credit	23.5	22.8	23.8	32.6
Image	1.6	1.6	1.8	6.0
Ecoli	14.8	12.9	12.4	25.3
Votes	4.8	4.1	4.0	8.6
Liver	30.7	27.3	27.2	40.3
Letters	3.4	3.4	4.1	23.8
Sat-images	8.8	9.1	10.2	17.3
Zip-code	6.2	6.2	7.2	22.7
Waveform	17.8	16.0	16.1	33.2
Twonorm	4.9	3.8	3.9	20.9
Threenorm	18.8	16.8	16.9	34.8
Ringnorm	6.9	4.8	4.6	24.6

TAB. 2.3: Taux d'erreur en test pour les Forest-RC (%) [Breiman 2001]

du temps de choisir une valeur supérieure.

Parmi les bases de données testées dans ces expérimentations, celles pour lesquelles Forest-RC est moins performant qu'Adaboost sont les trois plus grandes. Breiman approfondit ces expérimentations avec ces trois bases en effectuant quelques tests pour une valeur de K égale à 100. Sur deux d'entre elles, il arrive à améliorer ces résultats significativement. Concernant la troisième base, il explique qu'il réussit à obtenir un taux d'erreur minimal avec l'algorithme Forest-RI pour une valeur de K égale à 25. Cela prouve selon nous que la valeur de K est cruciale pour obtenir de bonnes performances, mais qu'il est difficile de la fixer *a priori*. De plus, Breiman ne discute pas ici la valeur de F, dont on ne sait finalement pas si elle est tout aussi influente sur les performantes que K. La conclusion de cela est que la paramétrisation de cet algorithme semble

importante pour obtenir de bonnes performances, mais est également très complexe.

En définitive, l'intérêt de cet algorithme réside principalement dans le fait qu'il semble plus adapté que Forest-RI aux problèmes dont l'espace de description est de dimension plus faible. Bien souvent cependant, le gain de performances ne semble pas très significatif. De plus, sur les bases de données de dimensionnalité élevée, Forest-RI se montre dans la majorité des cas plus performant. Et enfin, cet algorithme introduit un nouveau paramètre qui le rend encore plus difficile à maîtriser que Forest-RI.

2.6.4 Extremely Randomized Trees (ou Extra-Trees)

L'aléatoire peut intervenir à différents niveaux du processus de construction de forêts, à l'image de l'algorithme Forest-RI au sein duquel il intervient dans un premier temps en amont de l'induction des arbres de décision (Bagging), puis dans un second temps à l'intérieur même de ce processus d'induction (Random Feature Selection). Avec leurs **Ensembles d'Arbres Extrémement Aléatoires** [Geurts 2006], Geurts *et al.* accentuent le facteur aléatoire à l'intérieur du processus d'induction d'arbres et suppriment en contrepartie l'utilisation du principe de Bagging en amont. Ils nomment cette méthode **Extra-Trees (ET)** pour **Extremely Randomized Trees**.

S'appuyant sur plusieurs de leurs précédents travaux, ils justifient cette décision en adoptant un point de vue "biais/variance". Ce paradigme est souvent au centre des problématiques d'induction d'arbres de décision, dont on sait qu'ils ont une variance élevée, et plus encore de celle des combinaisons de classifieurs, dont on sait qu'ils permettent de pallier cela (*cf.* principes de Bagging et Boosting, sections 1.4.1 et 1.4.4).

Dans ce cas précis, le renforcement de l'aléatoire dans le processus d'induction d'arbres vise à en réduire la variance, tandis que l'utilisation de l'intégralité de l'ensemble d'apprentissage pour chaque classifieur de base devrait permettre la réduction du biais. Cette idée est motivée par le constat que la variance de l'erreur d'un arbre de décision est principalement induite par le choix d'un test de partitionnement optimal, et plus précisément d'une valeur de coupure optimale. Ils proposent donc dans leur algorithme de sélectionner ce point de coupure de façon totalement aléatoire, c'est-à-dire indépendamment de la variable à prédire.

Les auteurs ont testé leur algorithme sur 24 bases de données différentes, la moitié concernant des problèmes de classification, et l'autre moitié des problèmes de régression. Ils l'ont ainsi comparée à quatres autres classifieurs : un unique arbre de décision obtenu avec l'algorithme CART (désigné par l'acronyme PST pour Pruned Single Tree) ; une forêt obtenue avec une méthode de Tree Bagging (TB) ; une forêt de Random Trees (qu'ils désignent par l'acronyme RS pour Random Subspace, car ils considèrent que le Random Feature Selection est une variante des Random Subspaces) ; ainsi qu'une forêt aléatoire construite avec l'algorithme Forest-RI, *i.e.* la combinaison des deux précédents types de forêts (RF pour Random Forest). Ces

comparaisons sont intéressantes car elles permettent d'étudier dans le détail à quel point l'aléatoire est influent, au sein du processus complet d'induction de forêts. La figure 2.5 présente les résultats obtenus pour ces expérimetations ([Geurts 2006]). Toutes les forêts ont ici été construites avec 100 arbres. Les astérisques à côté de chaque acronyme dans la figure indiquent que les résultats ont été obtenus pour une valeur du paramètre K optimale, K étant le nombre de caractéristiques aléatoirement pré-sélectionnées à chaque nœud, tandis que la lettre d signifie que la valeur utilisée est la valeur par défaut, à savoir $K = \sqrt{M}$, où M désigne la dimension de l'espace des caractéristiques.

Sans surprise, on constate immédiatement que l'arbre CART est incontestablement supplanté par les méthodes de combinaisons d'arbres, très largement même pour la grande majorité des bases. On constate ensuite que le Tree Bagging donne très souvent de moins bons résultats que les trois autres forêts : dans 11 cas sur 12 pour les problèmes de classification, et dans 6 cas sur 12 pour les problèmes de régression. Ce premier constat semble indiquer que l'introduction de l'aléatoire dans le processus de sélection du test de partitionnement à chaque nœud donne de meilleurs résultats qu'un choix essentiellement déterministe dans le cadre d'une forêt. Rappelons également que Breiman lui-même avait reconnu que les méthodes de Bagging sont moins performantes que des méthodes de combinaison telle que le Boosting. Il avait d'ailleurs justifié l'implémentation du Bagging dans l'algorithme Forest-RI par son utilisation en tandem avec le Random Feature Selection uniquement.

Il est intéressant à présent d'examiner plus particulièrement les résultats des trois dernières méthodes. Les taux d'erreurs moyens sur tous les tests sont globalement très proches sur l'ensemble des bases de données, et il est difficile dans un premier temps de conclure de façon certaine que l'une ou l'autre des méthodes se montre plus performante. Les auteurs précisent cependant que les Extras-Trees sont au moins aussi performants que les autres méthodes, et même souvent sensiblement meilleurs, notamment sur les problématiques de classification.

Pour mieux analyser les effets de l'accentuation de l'aléatoire dans un tel système, Geurts *et al.* étudient à nouveau le dilemne "biais/variance", d'un point de vue plus comparatif cette fois. Les conclusions qu'ils en tirent sont intéressantes car elles permettent de mieux interpréter les effets de la randomisation du processus de sélection du test de partitionnement. La figure 2.6 présente des histogrammes des résultats obtenus. La première observation qui en ressort est que les méthodes de forêts permettent de considérablement diminuer la variance en comparaison avec l'arbre CART, mais également augmentent significativement le biais. C'est particulièrement le cas de la méthode Extra-Trees qui maximise le biais dans ces expérimentations pour une valeur de $K = 1$.

Sur cette figure, deux variantes d'Extra-Trees sont analysées : la première utilise la valeur optimale du paramètre K (ET^*), la seconde utilise $K = 1$, c'est-à-dire sélectionne de façon entièrement aléatoire la caractéristique de la règle de partitionnement de chaque nœud (ET^1). Cette comparaison permet de mettre en évidence que la sélection totalement aléatoire de cette caractéristique engendre un biais très fortement supérieur à celui de ET^*. En contrepartie, le gain de performance et

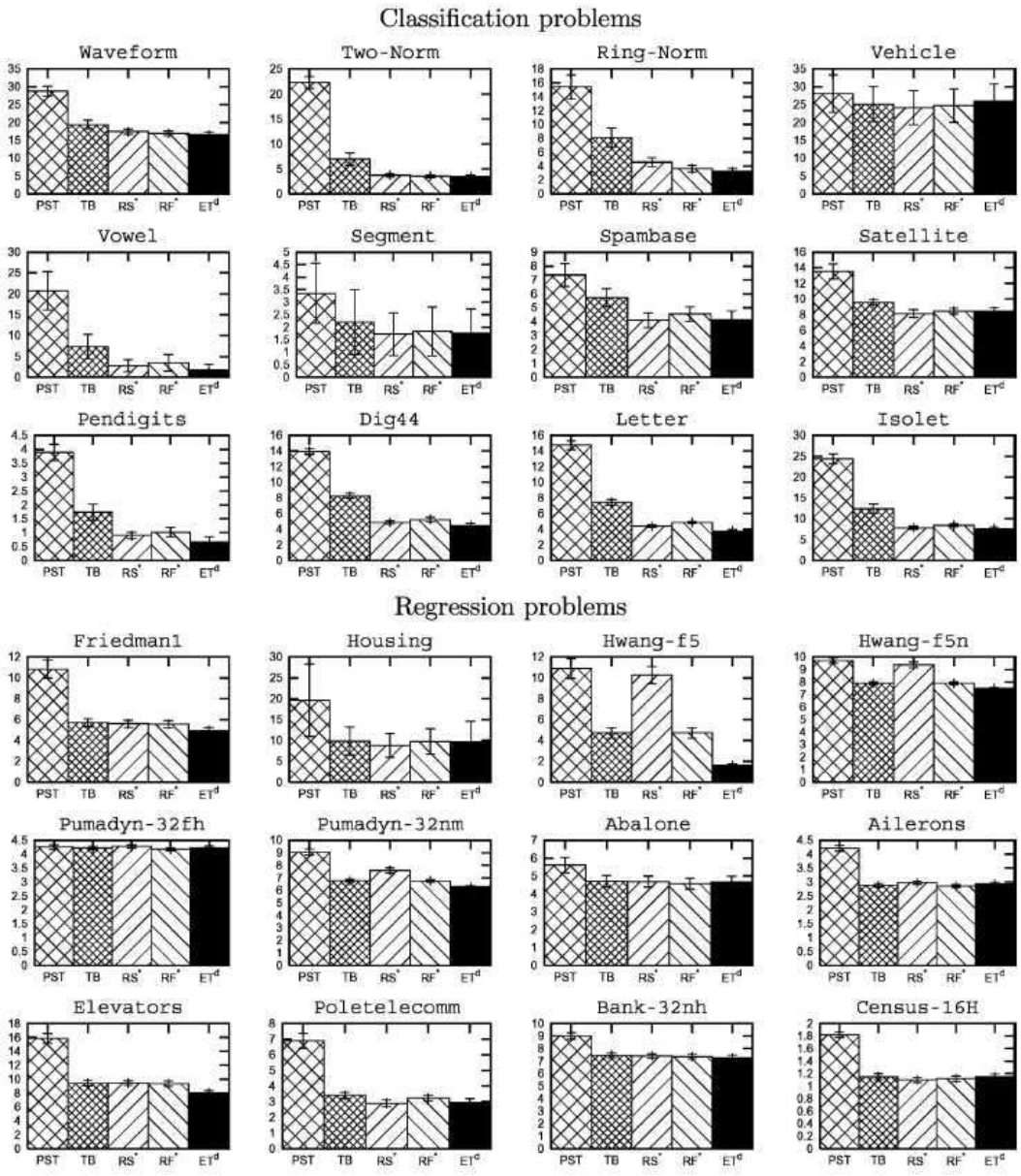

FIG. 2.5: Comparaison des erreurs moyennes et des écart-types sur 12 problèmes de classification et 12 problèmes de régression [Geurts 2006]

la diminution de la variance ne sont pas aussi significatifs. Cela tend à démontrer qu'un processus d'induction d'arbres "totalement aléatoire" (ici non seulement l'attribut est sélectionné aléatoirement, mais également le point de coupure) n'est pas nécessairement plus efficace.

Comme nous l'avons mentionné dans un paragraphe précédent, les auteurs ont justifié l'utilisation de l'intégralité de l'ensemble d'apprentissage par la réduction du biais. On peut avec ces estimations de la variance et du biais vérifier cette affirmation,

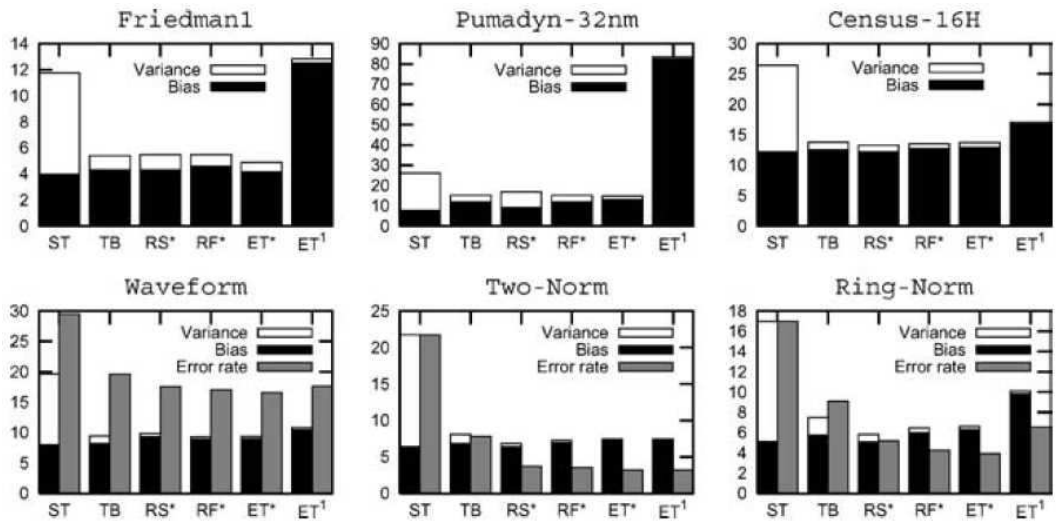

FIG. 2.6: Comparaison du biais et de la variance sur 3 problèmes de classification et 3 problèmes de régression [Geurts 2006]

notamment en comparant les résultats obtenus avec les méthodes RS et RF, puisque ce qui sépare ces deux méthodes est l'utilisation ou non du Bagging. On constate effectivement que le biais pour les RF est sensiblement supérieur à celui des RS.

En définitive, on remarque que l'on peut extraire de ces histogrammes des tendances globales de l'évolution du biais et de la variance par rapport au degré d'aléatoire injecté dans le processus de sélection des tests de partitionnement. Du plus déterministe, pour les TB, vers le plus aléatoire pour les ET^1. On constate donc que plus l'aléatoire est présent et plus la variance diminue, mais que dans le même temps, le biais augmente. Les auteurs insistent cependant sur le fait que la diminution de la variance n'est pas équivalente à l'augmentation du biais, et que les ET^* notamment permettent d'obtenir le meilleur compromis biais/variance parmi toutes ces méthodes (95% de diminution de la variance contre 21% d'augmentation du biais).

2.6.5 Perfect Random Tree Ensembles (ou PERT)

Cutler et Zhao introduisent dans [Cutler 2001] une méthode appelée **Perfect Random Trees (PERT)** qui renforce également le facteur aléatoire dans l'induction des arbres de décision. Adèle Cutler, qui avait collaboré avec Breiman à la conception de l'application Fortran "Random Forest" (*cf.* section 2.6.2), présente cette nouvelle méthode d'induction de forêts aléatoires qui reprend l'algorithme Forest-RI, mais en accentuant la randomisation dans l'induction des arbres de décision.

Le procédé utilisé pour ce faire est le suivant :

- à chaque nœud de l'arbre, la partition est réalisée en choisissant aléatoirement

deux de ses individus, de deux classes différentes ; appelons-les x et z. (Si ce n'est pas possible, il s'agit donc d'un nœud pur)

- La variable de partition est elle aussi choisie au hasard, et notée j.
- on génére une valeur α à partir d'une loi uniforme centrée réduite.
- on réalise la coupure de partition à la valeur $\alpha x_j + (1 - \alpha) z_j$.

Au sein d'un système multi-classifieurs, le classifieur de base PERT peut ensuite être combiné en utilisant pour chaque arbre l'intégralité de l'ensemble d'apprentissage, ou en mettant en œuvre les principes de Bagging, comme c'est le cas pour les Forest-RI.

Les propriétés de cette méthode mises en valeurs par les auteurs concernent en premier lieu la rapidité d'exécution, et l'adaptation parfaite de la structure aux données d'apprentissage. En contre-partie, on constate que les arbres produits sont considérablement plus grands et plus complexes en termes de hiérarchie qu'un arbre produit par la méthode CART par exemple, ce qui augmente le coût de complexité en mémoire et les temps d'exécution en prédiction.

Ensuite, Cutler et Zhao présentent une comparaison des résultats d'un système de combinaison d'arbres PERT (Forêt de PERT ou PERT-Forest) avec ceux obtenus par Breiman avec l'algorithme Forest-RI. Les expérimentations menées reprennent le protocole mis en œuvre dans [Breiman 2001]. Les résultats obtenus avec cette méthode sont relativement proches de ceux obtenus avec Forest-RI, et parfois même meilleurs, alors que nous le rappelons leur facilité de mise en œuvre est bien plus intéressante. Ces résultats nécessiteraient cependant de notre point de vue d'être confirmés avec un protocole expérimental plus solide. A noter qu'il n'est pas précisé dans l'article si le Bagging est utilisé pour ces expérimentations.

Dans un second temps, les auteurs examinent plus précisément le mécanisme des forêts de PERT du point de vue "*force*/*corrélation*", tout comme Breiman le fait dans [Breiman 2001] avec Forest-RI. Le tableau 2.4, qui présente les estimations de *force* et de *corrélation* pour des forêts induites avec Forest-RI et pour des forêts de PERT, suggère que l'accentuation de l'aléatoire dans le processus d'induction d'arbres tend logiquement à diminuer la *force* des forêts, mais également à considérablement en diminuer la *corrélation*. Les auteurs en concluent, qu'en réussissant à "renforcer" individuellement les arbres de décision PERT, tout en maintenant la *corrélation* à ce bas niveau, on pourrait présumer obtenir des résultats nettement meilleurs que les forêts aléatoires Forest-RI. Ils ne vont cependant pas jusque là et à notre connaissance, il n'existe pas d'extension de ces travaux dans la littérature.

En conclusion, l'intérêt principal de ces travaux réside de notre point de vue dans son étude de l'effet de l'accentuation de l'aléatoire sur le fontionnement des forêts, et plus précisément sur les propriétés de *force* et de *corrélation*. Cette étude ne va cependant pas aussi loin qu'on pourrait l'espérer puisque ces conclusions ne sont pas exploitées pour obtenir une méthode plus performante que Forest-RI par exemple.

	Strength, s		Correlation, $\bar{\rho}$	
Data Set	CART	PERT	CART	PERT
waveform	.37	.20	.22	.06
twonorm	.54	.50	.14	.07
threenorm	.31	.20	.13	.04
ringnorm	.48	.27	.14	.05

TAB. 2.4: Comparaison de la *force* et de la *corrélation* des Forest-RI et des PERT-Forest [Cutler 2001]

2.6.6 Balanced Random Forests et Weighted Random Forests

Dans leur article [Chen 2004], Chen et Liaw, en collaboration avec Breiman, présentent deux variantes de la méthode **Forest-RI**. Il s'agit de proposer une méthode de forêts aléatoires plus adaptée à des problèmes de données inégalement réparties. Ces travaux partent du constat que les forêts aléatoires "classiques" ont tendance à négliger les classes qui pourraient être sous-représentées dans la population de données traitées. Cela peut être problématique dans certains cas où la classification des individus d'une classe sous-représentée est particulièrement importante ou simplement lorsqu'elle devrait engendrer un taux d'erreur relatif aussi faible que pour les autres classes. C'est par exemple souvent le cas pour les problèmes d'aide à la décision liés aux diagnostics médicaux.

Les auteurs proposent deux méthodes de forêts aléatoires adaptées à cette spécificité : les **Balanced Random Forest (BRF)** et les **Weighted Random Forest (WRF)**. Le principe pour chacune d'elles est de compenser cette sous-représentation qui en temps normal engendrerait un taux d'erreur relatif de la classe sous-représentée beaucoup plus élevé que pour les autres classes. Ces deux méthodes consistent (i) à sur-échantillonner ou sous-échantillonner les données de certaines classes pour disposer d'une base d'apprentissage mieux répartie (ii) à affecter un coût plus important aux erreurs de classification des données de la ou des classes minoritaires.

Balanced Random Forest (BRF)

Il s'agit pour cette première méthode d'une approche plutôt naïve. Dans le cas de données inégalement réparties il existe une forte probabilité pour que la ou les classes minoritaires soient également sous-représentées, voire non représentées, dans les ensembles bootstrap. Les auteurs proposent alors d'utiliser des ensembles d'apprentissage bootstrap **stratifiés**[5]. En d'autre termes les ensembles bootstrap sont constitués de plusieurs échantillons, chacun issu des sous-ensembles de données regroupées par classe. C'est-à-dire qu'il s'agit de générer aléatoirement un échantillon de cas issus de la classe minoritaire, puis un deuxième échantillon contenant autant de cas, pour une deuxième classe plus représentée et ainsi de suite. Les échantillons

[5] en anglais Stratified bootstrap

sont ensuite réunis pour former l'ensemble d'apprentissage pour un classifieur de base.

Weighted Random Forest (WRF)

Cette deuxième méthode suit le schéma de l'apprentissage pondéré[6]. Puisque les forêts aléatoires semblent favoriser les classes majoritaires, il va s'agir ici de pénaliser plus fortement les erreurs de classification des données appartenant aux classes minoritaires. Chaque classe se voit ainsi attribuer un poids, inversement proportionnel au nombre de données qui la représentent dans l'ensemble d'apprentissage. Ces poids sont alors pris en compte à deux reprises dans l'algorithme. Dans un premier temps, ils sont incorporés à l'évaluation des règles de partitionnement : ici le calcul du critère de Gini. Puis, dans un second temps, dans le vote d'aggrégation des prédictions : le vote à la majorité devient un vote à la majorité pondérée.

Les expérimentations ont été réalisées sur 6 bases de données dédiées à ce type de problèmes. Les deux méthodes présentées ci-dessus sont alors comparées entre elles, ainsi qu'à plusieurs autres méthodes spécialisées dans les problématiques de données inégalement réparties. Différentes métriques souvent utilisées dans ce contexte sont calculées pour ces comparaisons, telles que le taux "True Positive" ou "True Negative", la Précision, le Rappel, la F-Mesure, *etc.*

De ces différents résultats, détaillés dans [Chen 2004], on constate globalement que les deux méthodes supplantent les systèmes auxquels elles sont comparées sur l'ensemble de ces 6 bases. Concernant la comparaison des deux méthodes entre elles, les résultats sont difficilement interprétables. Elles semblent très proches l'une de l'autre, à l'image des courbes ROC présentées dans [Chen 2004]. Cependant, on peut noter que la construction des BRF est algorithmiquement plus intéressante sur les grandes bases d'apprentissage notamment, car elle utilise des ensembles bootstrap contrairement aux WRF. De plus, les auteurs précisent en conclusion de ces travaux que cette deuxième méthode pourrait être sensible aux bruits, mais qu'une étude plus poussée serait nécessaire pour le confirmer.

En définitive, si ces méthodes de forêts aléatoires sont convaincantes dans le cadre de problèmes présentant une population d'individus dont la répartition est hétérogène vis à vis de la classe à prédire, les conclusions de ces travaux ne sauraient sortir de ce cadre. De plus, ces travaux ne permettent pas d'estimer le gain de performances, si gain il y a, que ces méthodes présentent en comparaison à la méthode Forest-RI. Et bien qu'intuitivement cela ne semble pas être le cas, rien ne garantit que les performances sur des bases plus régulières ne soient pas altérées.

2.6.7 Weighted Voting Random Forests

.

Nous l'avons vu dans la section 1.3, la diversité dans un système multi-classifieurs tel qu'une forêt aléatoire joue un rôle important. De façon plus intuitive, on peut

[6] en anglais *cost sensitive learning*

comprendre cela par le fait que la diversité dans les réponses des classifieurs de base laisse plus de place aux compromis. Certes, plusieurs avis valent mieux qu'un, mais à quoi bon si ces avis sont tous les mêmes, surtout lorsqu'ils sont érronés. Dans une forêt aléatoire, on espère donc que les erreurs de certains arbres de décision seront compensées par une majorité de bonnes prédictions parmi les autres. Et on comprend alors qu'en cas de mauvaise prédiction, les arbres de décision ne sont certainement pas tous responsables de cette erreur au même degré.

Partant de cette idée, Robnik-Sikonja propose dans [Robnik-Sikonja 2004] d'étudier une amélioration du système de vote des forêts aléatoires, dans le but d'évincer du vote final les arbres dont on sait qu'ils risquent de se tromper sur la donnée traitée. Son idée est de baser la sélection des arbres de décision que l'on fait participer au vote final, sur leurs performances individuelles sur des données "similaires". Pour chaque individu dont on souhaite prédire la classe, on détermine quelles sont, parmi les données d'apprentissage, celles qui lui ressemblent le plus. On décide alors de ne faire confiance qu'aux arbres de décision qui savent le mieux classer ces données dites "similaires".

Pour ce faire il s'appuie sur la procédure suivante, utilisée par Breiman dans [Breiman 2001] pour mesurer la similarité :

- en phase de prédiction, lorsque l'on souhaite prédire la classe d'un individu $\boldsymbol{x}$ quelconque avec un arbre de décision de la forêt aléatoire, on identifie les données d'apprentissage associées à la feuille avec laquelle $\boldsymbol{x}$ a été classé.
- toutes ces données d'apprentissage voient un indice de similarité qui leur est associé incrémenter.
- cette procédure est repétée pour chaque arbre de décision, en mettant à jour les indices de similarité de toutes les données d'apprentissage rencontrées par l'instance $\boldsymbol{x}$.

De cette façon, après que tous les arbres de la forêt ont classé $\boldsymbol{x}$, ces indices représentent des estimations du degré de similarité de chaque donnée d'apprentissage à laquelle ils sont associés. Si l'indice est nul, la donnée n'a jamais été rencontrée par $\boldsymbol{x}$ et ne lui est alors pas du tout similaire. Si par contre cet indice est maximal la donnée est une de celles qui lui ressemblent le plus.

L'objectif est ensuite d'estimer les performances des arbres de décision sur ces données similaires uniquement :

- on sélectionne les t données les plus similaires, *i.e.* dont l'indice de similarité, calculé comme expliqué ci-dessus, est le plus élevé.
- on classe ensuite chacune de ces données avec les arbres de décision de la forêt pour lesquels elles figurent dans l'ensemble out-of-bag.
- on mesure enfin la marge de chaque arbre de décision, sur la base des résultats ainsi obtenus (*cf.* équation 2.12 de la section 2.5).

De cette façon, il est possible dans un premier temps d'évincer du vote final pour la classe de $\boldsymbol{x}$, les arbres de décision pour lesquels la marge moyenne est strictement

négative, puis dans un second temps de pondérer le vote final, en utilisant comme poids associé aux votes de chaque arbre leur marge moyenne.

L'auteur effectue un certain nombre d'expérimentations dans le but d'étudier les évolutions de performances des forêts aléatoires lorsqu'elles implémentent ou non le vote pondéré. Il reprend pour ce faire les bases de données et le protocole expérimental utilisé par Breiman ([Breiman 2001]). Ayant également utilisé ces mêmes bases de données durant le développement de son système, Robnik-Sikonja décide d'effectuer une seconde batterie de tests sur des bases supplémentaires, afin d'en vérifier la robustesse. Ensuite, pour évaluer la différence de performances entre les forêts aléatoires utilisant un vote à la majorité classique, et celles utilisant un vote pondéré, il utilise le test statistique de Wilcoxon[7].

Pour analyser ces performances, l'auteur dresse un tableau récapitulatif des taux de reconnaissances sur chaque base testée, ainsi que de leurs aires sous la courbe ROC. Le constat immédiat de l'auteur, en comparant les résultats obtenus, est que le vote pondéré fournit la plupart du temps de meilleurs résultats que le vote ordinaire (11 cas sur 17), et dans tous les cas ne fournit jamais de résultats moins bons. Sur les 17 nouvelles bases de données testées par souci de robustesse, le constat est identique, et le vote pondéré intégré au processus de prédiction des forêts aléatoires engendre une forte amélioration des taux de reconnaissance (dans également 11 cas sur 17) et de l'aire sous la courbe ROC.

Il est important de noter que les complexités en temps et en mémoire sont considérablement augmentées avec l'intégration du vote pondéré. D'abord parce qu'il est nécessaire de mémoriser les données d'apprentissage associées à chaque feuille de chaque arbre de décision pour le calcul des mesures de similarité ($N \times L$ données, pour une base de N données d'apprentissage et pour L classifieurs de base). Puis parce que l'évaluation des marges relatives pour chaque prédiction alourdit également considérablement les temps de traitements en phase de décision.

2.6.8 Rotation Forests

Rodriguez *et al.* présentent dans [Rodriguez 2006] une nouvelle méthode d'induction de forêts aléatoires qu'ils nomment **Rotation Forest**. La motivation derrière cette méthode d'induction de forêts est de favoriser simultanément l'augmentation des performances individuelles des arbres et l'augmentation de la diversité.

Cette méthode s'inspire de trois principes d'induction d'EoC dont nous avons déjà parlé en section 1.4 : Random Subspaces, ECOC et Bagging. L'idée est de générer des sous-problèmes aléatoires pour chaque classifieurs élémentaires, en sélectionnant aléatoirement des sous-ensembles de caractéristiques, des sous-ensembles de classes, ainsi qu'en générant des ensembles bootstrap. Plusieurs de ces sous-problèmes sont générés pour chaque classifieur élémentaire. Ces sous-problèmes ne servent cependant pas directement à l'apprentissage de ces derniers,

[7] il s'agit d'un test statistique permettant de déterminer si deux échantillons sont issus de la même loi. Il permet ici d'évaluer les effets de l'implémentation du vote pondéré dans la méthode "classique" de forêts aléatoires, *i.e.* Forest-RI

mais sont en fait "reconstruits" à l'aide d'Analyses en Composantes Principales (ACP). Nous détaillons la procédure complète de cette méthode dans l'algorithme 4.

Algorithme 4 Rotation Forest

Entrée : $\mathcal{X}$ l'ensemble des N données d'apprentissage, chacune décrite par M caractéristiques.
Entrée : $\mathcal{Y}$ les étiquetes des données d'apprentissage.
Entrée : L le nombre de classifieurs élémentaires dans l'ensemble.

pour i de 1 à L **faire**
  $F_{i,j}(j = 1..K) \leftarrow$ Séparer l'ensemble des M caractéristiques en K sous-ensembles.
  pour j de 1 à K **faire**
    $X_{i,j} \leftarrow$ les données d'apprentissage projetées dans $F_{i,j}$.
    Enlever de $X_{i,j}$ un sous-ensemble aléatoire de classes.
    $X'_{i,j} \leftarrow$ Générer un ensemble bootstrap à partir de $X_{i,j}$ et de taille 75% de la taille de $X_{i,j}$.
    Appliquer une ACP sur $X'_{i,j}$.
  $R_i \leftarrow$ réarranger les vecteurs propres des ACP dans une matrice de rotation.
  $h_i \leftarrow$ apprendre le classifieur sur l'ensemble $\mathcal{X}R_i$.

Les auteurs présentent ensuite une étude comparative de leur méthode avec les principes de Boosting, de Bagging, et de forêts aléatoires, en utilisant donc à chaque fois des arbres de décision en tant que classifieurs élémentaires. L'algorithme mis en œuvre pour le Boosting est AdaBoost.M1, dont nous parlons dans la section 1.4.4, et celui pour les forêts aléatoires est Forest-RI.

Dans un premier temps, les auteurs montrent que la méthode des Rotation Forests est meilleure que les trois autres principes d'induction d'ensembles sur une majorité des bases de données testées, et ce pour un nombre L d'arbres de décision dans l'ensemble allant de 1 à 100. Les résultats obtenus pour ces expérimentations sont présentés dans le tableau 2.5. Les victoires de l'un ou l'autre des algorithmes sont décidées sur la base des taux de reconnaissance. La mise en place d'un test de significativité précise cependant qu'elles sont significativement meilleures que les trois autres principes pour 5 des 33 bases seulement. A noter également que les paramètres utilisés pour chacun des algorithmes auxquels sont comparées les Rotation Forests ont été fixés aux valeurs par défaut proposées dans le logiciel d'apprentissage automatique WEKA. Nous verrons dans le chapitre 3 que, concernant les forêts aléatoires, ces valeurs sont quelque fois très discutables.

De façon générale, ces expérimentations montrent que la méthode des Rotation Forests surpassent dans une majorité des cas les trois autres principes auxquels elles sont comparées. Les auteurs l'expliquent à l'aide de diagrammes de $\kappa - Error$

Data Set	Rotations J48	J48	Bagging J48	Boosting J48
anneal	98.93±0.95	98.61±1.06	98.89±0.92	99.58±0.71
audiology	79.80±6.92	77.24±7.04	81.03±7.36	84.90±7.07 ○
autos	82.50±8.66	82.34±9.22	82.69±8.60	85.31±6.99
balance-scale	90.33±2.52	77.82±3.69 •	81.85±3.74 •	78.46±4.07 •
breast-cancer	72.66±6.71	74.19±6.05	72.65±6.12	66.88±7.37 •
cleveland-14-heart	82.85±6.26	76.71±6.84 •	79.21±6.74	79.38±6.99
credit-rating	86.13±3.88	85.63±4.12	85.78±4.02	83.86±4.35
german-credit	74.10±3.93	71.09±3.53 •	73.75±3.62	71.01±3.93 •
glass	74.27±8.11	67.55±9.33 •	73.97±9.41	75.20±8.26
heart-statlog	82.25±6.43	78.22±7.20	80.74±6.66	78.27±7.20
hepatitis	82.80±8.91	79.58±9.28	81.24±8.22	82.46±8.00
horse-colic	84.73±5.44	85.16±5.70	85.41±5.70	81.63±6.11
hungarian-14-heart	80.28±6.33	80.08±7.65	79.62±6.70	78.75±6.65
hypothyroid	99.56±0.35	99.53±0.35	99.58±0.32	99.64±0.30
ionosphere	93.88±3.68	89.91±4.57 •	92.25±3.80	93.18±4.02
iris	95.73±5.20	94.89±5.03	94.67±5.12	94.27±5.18
labor	91.56±11.91	79.56±15.78•	83.13±15.20	87.31±13.36
letter	95.48±0.47	88.04±0.73 •	92.72±0.63 •	95.53±0.47
lymphography	83.99±8.33	76.37±11.09•	77.97±10.22•	81.73±8.61
pendigits	99.20±0.26	96.46±0.56 •	97.93±0.47 •	99.02±0.30
pima-diabetes	76.48±4.44	74.38±4.91	75.65±4.45	71.96±4.53 •
primary-tumor	45.06±6.40	41.71±6.83	43.74±6.76	41.87±6.53
segment	98.05±0.95	96.79±1.28 •	97.49±1.07	98.14±0.89
sonar	83.56±7.84	73.98±8.67 •	78.31±9.11	79.79±8.63
soybean	94.77±2.36	91.90±3.11 •	92.73±2.87 •	92.74±2.82 •
splice	95.47±1.15	94.17±1.22 •	94.43±1.26 •	94.60±1.15 •
vehicle	78.05±3.64	72.33±4.42 •	74.45±4.18 •	75.78±4.19
vote	96.26±2.79	96.49±2.65	96.37±2.54	95.34±3.11
vowel-c	96.89±1.74	79.62±4.17 •	90.20±3.16 •	92.77±2.77 •
vowel-n	95.68±1.95	79.16±4.58 •	89.45±3.22 •	92.13±2.84 •
waveform	83.93±1.69	75.27±2.00 •	81.75±1.70 •	81.34±1.88 •
wisconsin-breast-cancer	97.04±1.94	94.87±2.69 •	95.99±2.44	96.06±2.27
zoo	92.15±8.22	92.56±7.04	93.30±7.07	96.38±5.75
(Win/Tie/Loss)		(0/16/18)	(0/24/10)	(1/24/9)

○ Rotation Forest is significantly worse, • Rotation Forest is significantly better, level of significance 0.05

TAB. 2.5: Comparaison des taux de reconnaissance pour les méthodes Rotation Forest, C4.5 (noté J48 dans ce tableau), Bagging et Adaboost.M1 [Rodriguez 2006]

([Margineantu 1997]). Ces diagrammes permettent de représenter le compromis "performance/diversité" pour chaque paire d'arbres dans une forêt. Ils mettent en évidence que la méthode des Rotation Forest permet, en comparaison avec les autres principes d'induction d'EoC, d'améliorer les performances individuelles plus qu'elle n'augmente la diversité dans les ensembles. Les auteurs en concluent, au regard des résultats du tableau 2.5, que les performances individuelles prennent plus d'importance que la diversité dans les performances globales des forêts.

2.6.9 Les Probabilistic Random Forests

Dans [Breitenbach 2002], Breitenbach *et al.* définissent une méthode de forêt aléatoire, appelée **Probabilistic Random Forest (PRF)**, permettant de fournir

en plus d'une prédiction de classe, une probabilité de bonne classification, spécifique à l'individu à classer. Les auteurs évoquent la nécessité pour certaines applications de fournir en plus d'une "simple" prédiction, un indice de confiance de cette prédiction relative à l'individu.

Cet algorithme s'appuie sur la méthode **Minimax Probability Machine Classification (MPMC)** dont le principe est de déterminer un hyperplan qui minimise la probabilité maximale d'erreur de classification ([Lanckriet 2002]). Cet hyperplan est calculé en résolvant un problème d'optimisation formulé à l'aide des moyennes et des matrices de covariances des données de chaque classe du problème. Il s'agit d'ailleurs exclusivement de problèmes à deux classes.

Les Probabilistic Random Forests implémentent donc les deux mécanismes suivants :

- les règles de partitionnement de chaque nœud des arbres sont définies par :

$$a_1 K(f_{x_1}, f) + a_2 K(f_{x_2}, f) - b \begin{cases} \geq 0 & \rightarrow \text{ nœud fils de gauche} \\ < 0 & \rightarrow \text{ nœud fils de droite} \end{cases} \tag{2.25}$$

  où f est un vecteur de K caractéristiques sélectionnées aléatoirement par tirage sans remise (comme dans le principe de Random Feature Selection) pour la donnée à classer $\boldsymbol{x}$; où f_{x_1} et f_{x_2} sont les valeurs correspondantes pour deux données x_1 et x_2 appartenant respectivement à l'une et l'autre des deux classes du problème, et choisies aléatoirement parmi les données d'apprentissage ; et où $K(f_x, f) = f_x^T f$ est un noyau linéaire. Les valeurs a_1, a_2 et b, sont quant à elle obtenues à l'aide de la méthode MPMC, en utilisant les données du nœud courant pour estimer la covariance entre les classes, ainsi que leur centre.

- chaque donnée à évaluer par l'arbre h_i se voit associée une valeur calculée par l'équation :

$$\beta_i = a_1 K(f_x, f) + a_2 K(f_y, f) - b \tag{2.26}$$

  Les valeurs $\beta_1, ..., \beta_L$ sont ensuite utilisées à la fois pour la classification et pour le calcul d'un score de confiance associé à la prédiction. Ce score est obtenu en estimant une probabilité d'erreur de prédiction sur la base des distributions des points out-of-bag. Cette méthode nécessite donc de mettre en place un principe de pré-élagage qui assure que chaque feuille contienne bien un nombre suffisant de données out-of-bag de chaque classe pour l'estimation de ces distributions.

Les auteurs proposent quelques expérimentations pour éprouver les performances de leur système et le comparer à plusieurs autres algorithmes d'apprentissage, tel que les MPMC "classiques", les SVM et les forêts aléatoires Forest-RI. Ces expérimentations sont assez brèves et permettent difficilement de conclure sur les performances réelles de cette méthode. Les résultats obtenus sur les 5 bases de données testées sont peu convaincants puisque pour pratiquement chacune d'elle les Proba-RF donnent

des taux de reconnaissances inférieurs aux SVM ou à Forest-RI.

2.6.10 "Meta" Random Forests

Dans cette section nous présentons quelques travaux dont l'approche a été d'approfondir le paradigme "combiner pour mieux régner", en encapsulant deux niveaux de combinaison de classifieurs, c'est-à-dire en faisant de la combinaison de combinaison de classifieurs. Ce que nous appelons ici les **"Meta" forêts aléatoires** appliquent cette idée en utilisant des forêts aléatoires comme classifieurs de base au sein d'un second système de combinaison.

Dans un article publié en 2005 [Boinee 2005b], Boinee *et al.* présentent une étude sur deux systèmes de Meta forêts aléatoires. Il s'agit d'appliquer alternativement les principes de Boosting, via l'algorithme AdaBoost, et de Bagging, en utilisant des forêts aléatoires en tant que classifieurs de base. Les auteurs décrivent le processus de construction de forêts aléatoires comme celui des Forest-RI pour lesquels cependant il mettent en place le calcul des indices de similarité de Breiman, dont nous parlons dans la section 2.6.7.

Bagged Random Forest (BgRF)

Le principe de cette première méthode est plutôt simple et intuitif. L'application du Bagging à la combinaison de forêts aléatoires consiste essentiellement en la constitution d'un nombre J fixé d'ensembles boostrap à partir desquels sont construits J forêts aléatoires. Nous le rappelons l'algorithme Forest-RI reprend lui même le principe de Bagging pour l'apprentissage des arbres de décision qui composent la forêt. De cette façon chacun des J ensembles boostrap formés en premier lieu fait alors lui-même l'objet de "tirages avec remise" au sein du processus de construction de la forêt dans un deuxième temps. Chaque arbre de décision est ainsi construit sur la base d'un sous-ensemble d'apprentissage obtenu par deux échantillonnages successifs de l'ensemble d'origine. De la même manière les résultats sont combinés par deux processus de vote à la majorité — ou de moyennage dans le cas de problèmes de régression. La prédiction finale est donc le résultat de deux tours d'élection.

AdaBoosted Random Forest (ABRF)

Ce système s'appuie sur l'utilisation de l'algorithme AdaBoost que nous présentons dans la section 1.4.4. Les auteurs introduisent deux approches pour la construction de forêts aléatoires "AdaBoostées", mais ne les testent pas toutes les deux. La première consisterait à construire un ensemble de classifieurs combinants, chacun obtenu à l'aide du processus AdaBoost appliqué aux arbres de décision. Ces arbres de décision "AdaBoostés" seraient ensuite combinés selon un principe de forêts aléatoires, c'est-à-dire sur la base d'une famille de vecteurs aléatoires indépendants et identiquement distribués (*cf.* définition 1 dans la section 2.4). La seconde, et celle que les auteurs ont choisie, consiste à construire une forêt aléatoire à partir de l'ensemble d'apprentissage associé à une matrice de poids, puis à la booster avec l'algorithme AdaBoost. Et donc cette fois-ci, les arbres de décision

sont dans un premier temps combinés dans une forêt aléatoire, cette dernière étant dans un second temps utilisée comme classifieur faible pour l'algorithme AdaBoost.

Dans les expérimentations présentées ensuite dans le même article, les auteurs utilisent quelques-unes des bases de données de l'UCI Machine Learning Repository [Asuncion 2007]. Il ne s'agit cependant pas exactement des mêmes bases de données que celles utilisées par Breiman dans ses travaux sur les forêts aléatoires ([Breiman 2001]). L'objectif de ces expérimentations est essentiellement comparatif. Les auteurs mettent en oeuvre les algorithmes qu'ils décrivent, à savoir les Forest-RI, les BgRF et les ABRF, dont ils comparent les performances.

Dans ces expérimentations les BRF se montrent significativement plus performantes que les deux autres méthodes. La figure 2.7 détaille les résultats obtenus. On constate également que si les ABRF permettent aussi d'obtenir un gain de performances en comparaison avec Forest-RI, celui-ci n'est pas aussi important que pour les BRF.

Cependant il faut noter que pour l'algorithme Forest-RI, les auteurs ont choisi de fixer le nombre d'arbres de décision à 20. De notre point de vue ce nombre est bien trop faible pour obtenir des performances raisonnablement bonnes avec cet algorithme, comme nous le démontrons dans le chapitre 4, dans lequel nous avons mené une étude expérimentale plus complète sur cette problématique. Dès lors, il est difficile de conclure de ces résultats que les méthodes BRF et ABRF permettent réellement d'obtenir de meilleurs résultats que Forest-RI de façon générale. Il faudrait pour cela confirmer ces résultats pour une paramétrisation de Forest-RI plus adaptée ce qui à ce jour n'a pas été fait à notre connaissance.

2.6.11 Synthèse

Dans cette section nous avons listé et détaillé les principaux travaux de la littérature sur les méthodes d'induction de forêts aléatoires. Ces travaux explorent beaucoup de pistes intéressantes pour l'induction de forêts alétoires et proposent ainsi un grand nombre de méthodes. En particulier, on remarque qu'un certain nombre d'entre eux s'intéressent plus en profondeur à l'introduction de l'aléatoire dans le processus d'induction d'arbres de décision, et au processus de randomisation Random Feature Selection. Cela montre de notre point de vue que beaucoup des chercheurs s'intérrogent sur les mécanismes de fonctionnement de ce principe et sur son impact sur l'erreur en généralisation des forêts aléatoires.

Cependant, on constate également qu'assez peu de ces travaux ont été approndfondis dans ce sens. À l'exception des travaux de Geurts *et al.* sur les méthodes d'Extras-Trees qui ont notamment été utilisés pour des applications liées à l'analyse et au traitement des images ([Marée 2007, Dumont 2009]), très peu de ces méthodes ont fait l'objet de publications postérieures aux articles que sont mentionnés dans cette section. Ce manque de maturité, ainsi que parfois peut-être l'éfficacité toute relative de ces méthodes d'induction font qu'aujourd'hui rares sont les travaux qui mettent en œuvre un algorithme autre que Forest-RI.

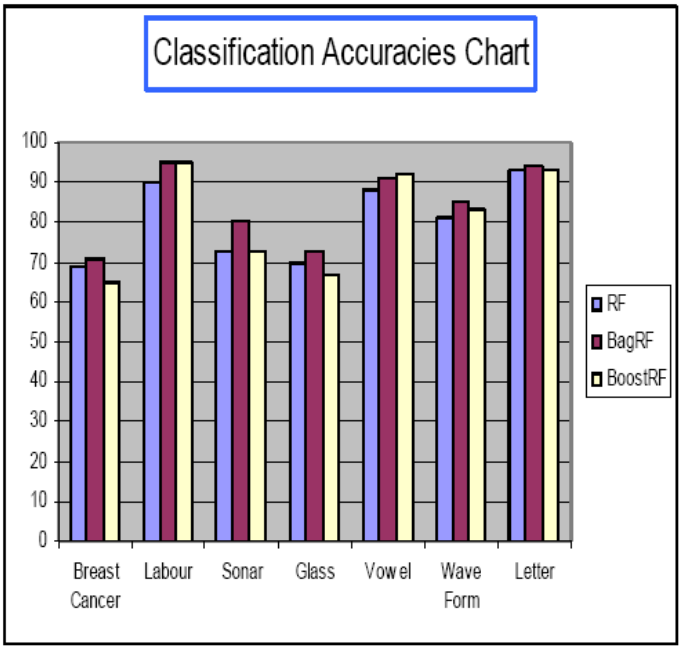

FIG. 2.7: Comparaison des taux de reconnaissance pour les méthodes Forest-RI, agged-RF et Boosted-RF [Boinee 2005b]

Dans la section suivante, nous discutons par conséquent la mise en œuvre de l'algorithme Forest-RI en particulier, ainsi que les diffultés que cette mise en œuvre pose encore aujourd'hui.

2.7 Discussions sur l'algorithme Forest-RI

Forest-RI est l'algorithme principalement responsable de la popularité grandissante des forêts aléatoires. Avec son implémentation en Fortran par Breiman et Cutler et les autres implémentations qui suivirent dans les logiciels d'apprentissage automatique et dans des langages plus populaire, de nombreux chercheurs l'ont par la suite confronté aux principes d'apprentissage d'ensemble de classifieurs les plus performants, tels que le Boosting, les Random Subspaces, le Bagging, *etc.* [Breiman 2001, Banfield 2004, Boinee 2005a, Geurts 2006, Banfield 2006]. C'est en constatant que les résultats sont souvent favorables aux forêts que les études sur ces méthodes se sont multipliées. Nous discutons dans cette section de quelques résultats expérimentaux obtenus avec les forêts aléatoires et introduisons les problématiques qui ont motivé ces travaux de thèse.

Comme nous l'avons vu dans la section 2.6.2, Breiman est le premier à comparer sa méthode à Adaboost, connu pour être l'un des principes les plus efficaces dans le cadre de l'induction de forêts de décision (*cf.* section 1.4.4). Une autre étude

expérimentale intéressante étend les tests de Breiman en comparant les forêts aléatoires induites avec Forest-RI aux techniques de Bagging, de Boosting et de Random Subspaces utilisées pour l'induction de forêts de décision ; il s'agit des travaux de Banfield *et al.* présentés dans [Banfield 2006]. La démarche ici est de concevoir un protocole expérimental le plus rigoureux possible, en reprenant l'ensemble des expériences menées par les auteurs des différents principes testés, et en utilisant plusieurs tests de significativité pour établir de la façon la plus fiable possible les écarts de performances d'une méthode à l'autre. Banfield *et al.* dressent un tableau très intéressant qui récapitule les différentes expérimentations antérieures à la leur (*cf.* tableau 2.6).

Reference	Number of Datasets	Ensemble Size	Experiments	Results
Ho [5], random subspaces	18	100	half-half split, repeat 10 times, average of middle 8	RS better than bagging, boosting in 5 or 6 of 14; no statistical test
Breiman [6], random forests	20	RF: 100, boosting: 50	90-10 split, 100 trials	RF better than Adaboost in 11 of 19 data sets, no statistical test
Dietterich [7], random trees	33	RT: 200 bagging: 200 boosting: 100	10-fold cross-val select pruned or unpruned trees	statistically significantly better in 6 of 33 data sets
Freund [3], boosting (Adaboost)	27	100	10-fold cross-val, 10 trials	boosting better than bagging in 12 of 27 data sets, no statistical test
this work	57	1,000	10-fold cross-val, 5x2-fold cross-val, Friedman-Holm test	no stat. sig. improvement over bagging in 38 of 57; boosting_1000, RF_lgN best; bagging, random subspaces equiv.

TAB. 2.6: Tableau récapitulatif des protocoles expérimentaux et des résultats de plusieurs travaux sur des principes d'induction de forêts de décision [Banfield 2006]

Ils mènent ensuite leurs expérimentations sur 57 bases de données et en utilisant deux tests statistiques de significativité différents. Le bagging est utilisé ici comme méthode de référence et les méthodes ne sont pas statistiquement comparées toutes entre elles, mais principalement au Bagging. Les résultats obtenus montrent dans un premier temps que pour 37 des bases testées, aucune différence significative n'a été observée entre chacun des principes d'induction d'ensembles et le Bagging. La conclusion est globalement que, si le Bagging se montre dans une majorité des cas le moins performant de ces principes d'induction d'ensembles, le gain de performances n'est pas toujours significatif.

Concernant l'algorithme Forest-RI plus particulièrement, les auteurs ont testé trois variantes, *i.e.* pour trois valeurs de K différentes. Pour reprendre les expérimentations de Breiman, les valeurs choisies ici sont 1, 2 et $int(\log_2(M)+1)$. Les résultats obtenus (*cf.* tableau 2.7) avec les tests de significativité montrent dans un premier temps que c'est avec $K = \lfloor \log_2(M)+1 \rfloor$ que Forest-RI bat le plus souvent le Bagging. Dans un second temps en comparant les valeurs des taux

d'erreur moyens et en appliquant une méthode de comparaison de la littérature, basée sur un classement des différents algorithmes testés, ils montrent qu'en moyenne $K = 2$ est le plus souvent le meilleur choix. Avec l'une ou l'autre de ces deux valeurs, les résultats montrent indiscutablement que les performances des forêts sont au moins aussi intéressantes que celles obtenues avec AdaBoost ou n'importe quel autre principe d'apprentissage de forêts ; ce qui n'est pas forcément le cas pour $K = 1$ pour lequel les performances sont sensiblement inférieures.

	Boosting 1000	Boosting 50	Random Subspaces	Random Trees B	Random Forests-lg	Random Forests-1	Random Forests-2
10 Fold Wins	10	8	6	10	8	1	8
10 Fold Losses	0	2	10	7	0	8	2
5x2 Fold Wins	8	6	5	2	6	0	5
5x2 Fold Losses	0	0	9	4	0	6	2
10 Fold average rank[1]	3.96	4.30	5.42	4.53	3.77	4.59	3.67
5x2 Fold average rank[2]	3.34	5.15	5.39	4.52	3.70	4.53	3.32

[1] Average rank for 10 Fold Bagging is 5.76
[2] Average rank for 5x2 Fold Bagging is 6.06

TAB. 2.7: Tableau récapitulatif des résultats des expérimentations de [Banfield 2006]

Ces derniers résultats, à l'instar des quelques conclusions discutables de Breiman à ce sujet, soulèvent la question de la paramétrisation de l'algorithme Forest-RI. Dans les expérimentations qui ont été menées avec cet algorithme, la valeur du paramètre K notamment est toujours fixée avec une des valeurs par défaut données dans la littérature. Cependant, aucune de ces valeurs n'a été justifiée, ni théoriquement, ni expérimentalement. On constate pourtant, avec des expérimentations comme celles de Banfield *et al.* que nous venons de présenter, que ces valeurs peuvent avoir une influence importante sur les performances des forêts. On peut déplorer alors qu'un test plus systèmatique des différentes valeurs possibles de K n'ait jamais été réalisé. Il n'existe d'ailleurs à ce jour aucune étude dans la littérature qui s'intéresse plus en profondeur à l'influence de la valeur de K sur les performances de Forest-RI, ni qui compare plus rigoureusement ces valeurs entre elles. Lorsque quelque l'on souhaite utiliser Forest-RI, on se heurte en définitive au réglage de ce paramètre car on ne dispose à ce jour d'aucun élément théorique ni d'aucune règle heuristique permettant d'en fixer la valeur.

Ce paramètre n'est par ailleurs pas le seul problème que soulève l'algorithme Forest-RI concernant l'induction des forêts. Plus classiquement le deuxième paramètre L, qui fixe le nombre d'arbres à induire dans la forêt, n'est pas plus facile à règler. Ce paramètre est commun à la plupart des algorithmes d'induction d'EoC, mais il n'a jamais été étudié en particulier dans le cadre des forêts aléatoires. Pourtant, les questions de son réglage ainsi que de son influence sur les performances sont encore des problèmes ouverts.

Par exemple, Breiman prouve mathématiquement dans [Breiman 2001] que pour un nombre croissant d'arbres dans une forêt, ses performances en généralisation convergent. Nous avons discuté ce résultat dans la section 2.5. Si cela nous indique qu'à partir d'une certaine limite, ajouter plus d'arbres ne permet pas d'obtenir de gain de performances significatif, cela ne nous renseigne pas sur la valeur de cette limite. Nous montrons d'ailleurs, dans le chapitre 4, que cette valeur est souvent différente d'une base à une autre. Ce résultat soulève de notre point de vue plusieurs autres questions : tous les arbres contribuent-ils à l'amélioration des performances de la forêt ? N'y-a-t'il pas des arbres aléatoires qu'il serait préférable de ne pas ajouter à l'ensemble parce qu'ils apportent plus d'erreurs en prédiction qu'ils n'améliorent les performances ? De ce fait le processus d'induction statique tel qu'il est mis en œuvre dans l'algorithme Forest-RI est-il adapté à ce type d'EoC ? Ne serait-il pas profitable de guider l'induction des arbres pour mieux en contrôler la contribution au sein d'une forêt ? Il nous semble important en définitive d'apporter des réponses à toutes ces questions, afin dans un premier temps de permettre une meilleure utilisation de l'algorithme Forest-RI, puis dans un second temps d'améliorer le processus d'induction des forêts pour mieux en maîtriser les performances.

2.8 Conclusion

Dans ce chapitre nous avons présenté de façon détaillée la famille des forêts aléatoires. Nous en avons tout d'abord donné une définition précise et nous en avons expliqué les principes de fonctionnement. Puis, nous avons dressé un panorama des travaux qui se sont intéressés à ces méthodes depuis qu'elles ont été introduites en 2001. Dans ces travaux, beaucoup de processus d'induction de forêts aléatoires ont été proposés, qui explorent un grand nombre de pistes intéressantes. Cependant, l'algorithme le plus souvent mis en œuvre pour l'induction de forêts aléatoires reste l'algorithme Forest-RI.

Nous avons vu dans ce chapitre que dans les différentes expérimentations sur cet algorithme, celui-ci se montre particulièrement compétitif avec les autres méthodes d'EoC, et notamment avec l'algorithme de Boosting AdaBoost, réputé pour être l'un des principes d'apprentissage d'EoC les plus performants, particulièrement dans le cadre de forêts de décision. Cependant, la mise en œuvre de cet algorithme pose de notre point de vue plusieurs problèmes.

Le premier de ces problèmes concerne la paramétrisation du processus de randomisation Random Feature Selection qu'il met en œuvre. Ce paramètre, noté K, correspond au nombre de caractéristiques sélectionnées aléatoirement à chaque nœud des arbres, au cours de leur induction. À ce jour, les seules solutions de paramétrisation dont disposent les utilisateurs de cette méthode pour fixer ce paramètre, sont des valeurs traditionnellement utilisées dans la littérature mais qui sont toutes arbitraires. Pourtant, nous avons vu avec quelques études comparatives que nous présentons dans ce chapitre que la valeur de ce paramètre est importante

pour obtenir de bonnes performances avec cet algorithme. On peut regretter alors que des études plus approfondies des différentes valeurs possibles de ce paramètre n'aient pas été réalisées pour en étudier l'influence sur les performances. C'est une première problématique que nous proposons d'aborder dans ces travaux de thèse et ce, à la fois afin de fournir des solutions de paramétrisation plus solides que celles qui sont proposées dans la littérature, et aussi et surtout afin de mieux comprendre l'influence du processus de Random Feature Selection sur le comportement des forêts.

Un deuxième problème qu'il nous semble important d'étudier concerne le nombre d'arbres à utiliser et l'apport de chacun d'eux aux performances des forêts. En effet, l'algorithme Forest-RI, comme la plupart des algorithmes d'induction de forêts aléatoires que nous avons évoqués dans ce chapitre, est un processus d'induction statique, qui construit et ajoute les arbres de façon indépendante les uns des autres. Cela implique qu'aucune garantie n'est donnée sur la complémentarité de ces arbres au sein d'un même comité et sur le fait que chacun d'eux participe à l'amélioration des performances. On peut se demander s'il n'y a pas parmi les arbres d'une forêt une proportion d'entre eux qui apportent plus d'erreurs en prédiction qu'ils ne permettent d'améliorer les performances de l'ensemble. Par conséquent, une des questions que nous nous posons avec ces travaux de thèse est de savoir s'il ne serait pas préférable d'induire les arbres de façon dépendante, de sorte de mieux contrôler leur contribution à l'ensemble, et si oui, de trouver un moyen de le faire de façon efficace.

Chapitre 3

Influence du *Random Feature Selection* sur les Performances des Forêts Aléatoires

Sommaire

3.1 Introduction

La force de l'algorithme Forest-RI pour l'induction de forêts aléatoires repose principalement sur l'utilisation du mécanisme de Random Feature Selection que nous avons présenté dans le précédent chapitre (*cf.* section 2.6.1). La construction d'un arbre de décision qui intègre ce principe est guidée par un paramètre important : le nombre K de caractéristiques choisies aléatoirement à chaque nœud pour déterminer sa règle de partitionnement. Ce nombre permet d'introduire plus ou moins d'aléatoire dans l'induction d'un arbre. Or, comme nous venons de le voir en conclusion de ce précédent chapitre, il existe dans la littérature très peu d'études sur l'influence de ce paramètre sur le comportement des forêts induites avec Forest-RI. Il semble même assez difficile d'appréhender le comportement de ce type de forêts en fonction de ce paramètre. Pourtant, ce paramètre a une influence importante sur les performances, comme le montre la figure 3.1 qui présente 6 courbes de résultats que nous avons obtenus au cours de cette première étude. Sur ces 6 diagrammes, nous présentons l'évolution du taux d'erreur en test en fonction des valeurs du paramètre K. Nous expliquerons plus en détails dans la suite de ce chapitre comment ces courbes ont été obtenues, mais nous montrons déjà avec cette figure que pour des valeurs différentes de K, on observe la plupart du temps des écarts significatifs de taux d'erreur, parfois très importants. Ce paramètre est donc crucial dans le fonctionnement de l'algorithme Forest-RI, et pourtant très peu de travaux de recherche s'y sont intéressés.

Pour expliquer le rôle de ce paramètre, Breiman tout d'abord amorce dans [Breiman 2001] une étude expérimentale de l'évolution des performances, de la force et de la corrélation d'une forêt en fonction de plusieurs valeurs de K. Il propose d'utiliser une base de données relativement petite en termes de nombre de données (*i.e.* 208 données au total), mais avec un nombre relativement grand de caractéristiques disponibles (*i.e.* 60 caractéristiques). Le test est simple : lancer l'apprentissage de plusieurs forêts pour des valeurs de K allant de 1 et 50, et calculer pour chacune d'elles l'erreur en test et l'erreur out-of-bag, ainsi que la force et la corrélation telles qu'elles sont définies dans la section 2.5. Il semble évident que, bien que le principe de ces expérimentations soit intéressant, les conclusions sont très immatures puisqu'une seule base est testée ici et qu'en plus cette base est de notre point de vue particulièrement inadaptée à ce type d'étude en raison du très faible nombre de données d'apprentissage et de test. On peut noter cependant que Breiman dresse dans le cadre de cette étude quelques pistes intéressantes pour mieux comprendre le comportement des forêts et plus particulièrement du processus de Random Feature Selection. Il met notamment en évidence qu'au delà d'une certaine valeur de K le rapport $\frac{\overline{\rho}}{s^2}$ (*cf.* section 3.5) est détérioré par une augmentation de la corrélation, tandis que la force elle devient rapidement constante.

À cause également des faiblesses du protocole expérimental (une seule base avec peu de données), les conclusions quant à l'évolution des erreurs out-of-bag et de test ne nous semblent pas fiables tant qu'elle ne sont pas confirmées sur plusieurs autres

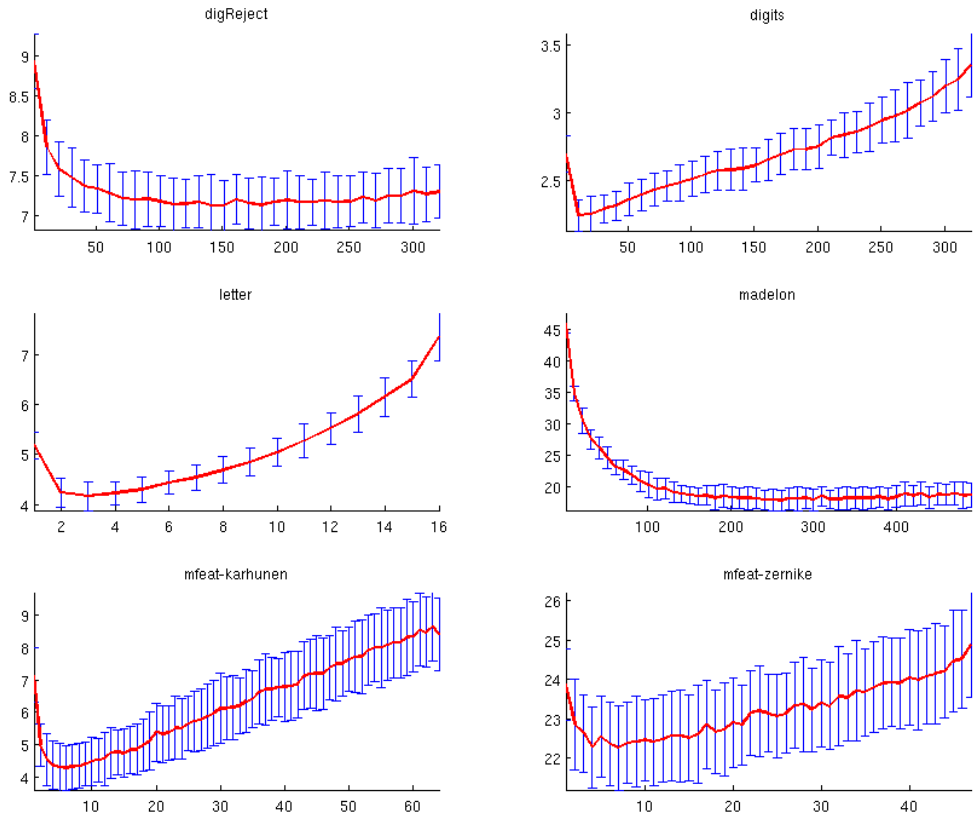

FIG. 3.1: Evolution du taux d'erreur en test en fonction de K pour 6 bases de données

bases de données différentes. L'évolution de l'erreur en généralisation en fonction de la valeur de K présente une forte instabilité que nous pensons en grande partie due à la petite taille de la base.

D'autres chercheurs ont reproduit le même type d'expérimentations sur un plus grand nombre de bases de données, mais avec un algorithme légèrement différent ; il s'agit de Geurts *et al.* avec leurs Extra-Trees dont nous parlons dans la section 2.6.4. Ils présentent dans [Geurts 2006] des courbes représentant l'évolution du taux d'erreur en test en fonction de la valeur de K. Ils proposent de cette façon de mieux comprendre l'influence de K sur les performances en classification et de justifier par la même occasion l'utilisation de la valeur par défaut $K = \sqrt{M}$, où M est le nombre de caractéristiques disponibles. Les résultats obtenus pour les problèmes de classification sont montrés dans la figure 3.2.

On y identifie trois grandes tendances d'évolution de l'erreur en fonction de K : (i) une décroissance monotone des taux d'erreurs (bases Vehicle, Sattelite et Pendigits), (ii) une croissance monotone (base Twonorm) (iii) une décroissance suivie d'une croissance de ces taux (les 8 autres bases). Pour les deux premières catégories, on constate par ailleurs que la valeur par défaut $K = \sqrt{M}$ est clairement sous-

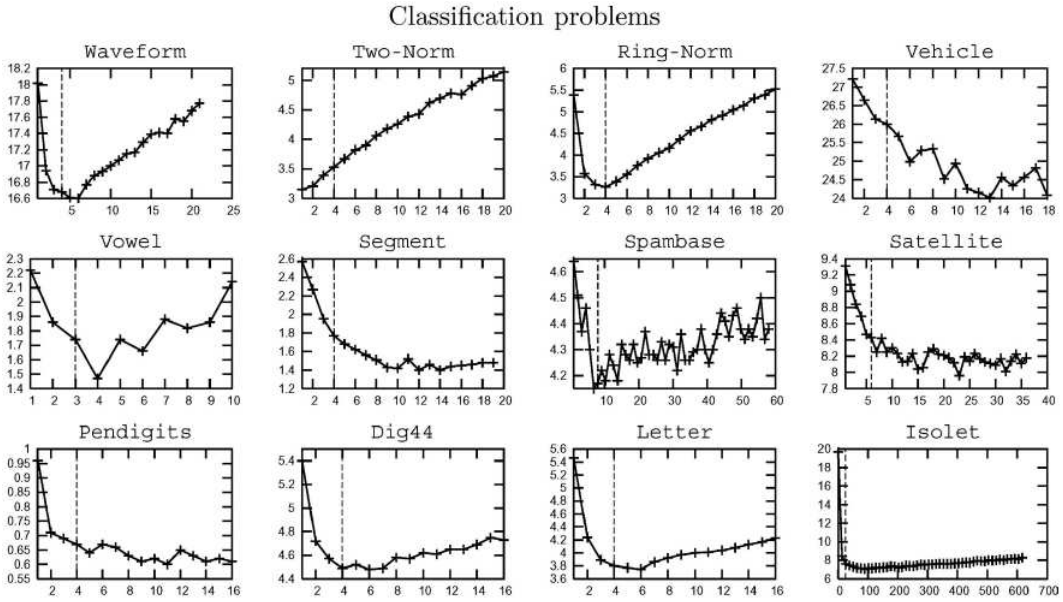

FIG. 3.2: Évolution de l'erreur en test des Extra-Trees en fonction de K [Geurts 2006]

optimale. Ces travaux ont beaucoup guidé notre approche pour cette première étude, notamment s'agissant de l'analyse du rôle de la qualité des caractéristiques sur le comportement des forêts vis à vis de K. Geurts *et al.* donnent quelques éléments à ce sujet sur lesquels nous reviendrons plus tard dans ce chapitre.

Pour conclure sur ces travaux, bien que ces résultats soient particulièrement intéressants dans la mesure où ils dressent des schémas de comportement des forêts en fonction de K, ils concernent un algorithme d'induction de forêt aléatoire qui n'utilise pas le Bagging, et qui n'est pas celui qui nous intéresse ici, à savoir Forest-RI. Et même si ces deux algorithmes sont proches dans leur fonctionnement, il n'est pas possible de deviner dans quelle mesure les résultats et le comportement des forêts sont affectés par leurs différences.

En définitive, lorsque l'on souhaite utiliser l'algorithme Forest-RI pour un problème de classification quelconque, on se retrouve rapidement confronté au problème du choix de la valeur de K. Si les travaux que nous venons de mentionner donnent quelques pistes pour la compréhension du rôle de ce paramètre dans les performances en classification, les conclusions sont encore insuffisantes pour pouvoir fixer *a priori* sa valeur pour un problème donné, afin de tirer le meilleur parti des méthodes de forêts aléatoires. Les seuls éléments que fournit la littérature pour cela sont quelques valeurs arbitraires, dont il est difficile de se satisfaire. Nous listons les principales ci-dessous :

- $K = 1$
- $K = \lfloor \log_2(M) + 1 \rfloor$

- $K = \sqrt{M}$

Nous utiliserons par la suite la notation Forest-RI/$K_{\sqrt{M}}$, Forest-RI/$K_{\log_2}$ et Forest-RI/K_1 pour désigner les forêts aléatoires induites avec Forest-RI et chacune de ces trois solutions de paramétrisation.

Dès lors plusieurs questions se posent : tout d'abord parmi les principales valeurs utilisées dans la littérature, y en a-t-il une qui permet d'obtenir de meilleurs résultats en moyenne que les autres ? Ensuite existe-t-il d'autres valeurs plus pertinentes ? Enfin est-il possible de trouver une règle de paramétrisation pour K qui en fixerait la valeur en fonction du problème de classification à résoudre ? Par exemple, peut-on trouver une fonction $K = g(N, M, c)$, où N représente le nombre de données, M le nombre de caractéristiques, et c le nombre de classes ? L'idée principale de cette première étude est d'apporter des éléments de réponse à toutes ces questions. Dans un premier temps, nous reproduisons le même schéma expérimental que Geurts *et al.* , à savoir tester de façon systématique toutes les valeurs possibles de K, et étudier l'évolution des performances en fonction de ces valeurs. De cette façon, nous pouvons comparer entre elles les valeurs proposées dans la littérature, et déterminer s'il existe une ou plusieurs autres valeurs qui leur sont préférables. Nous montrons que l'évolution des performances en fonction de K est importante et que les valeurs de la littérature ne conviennent pas à tous les problèmes d'apprentissage automatique. Pour certaines de ces bases, nous trouvons une valeur de K très différente de ces valeurs par défaut, et qui permet d'obtenir un gain de performances important.

Nous proposons ensuite d'étudier plus en profondeur les raisons de cette évolution de performances en fonction de K et donnons une explication des différences de comportement que l'on observe d'une base de données à une autre. Nous établissons notamment un lien entre la "qualité" des caractéristiques qui forment l'espace de description et cette évolution des performances.

Enfin nous présentons dans une dernière section un algorithme original, appelé Forest-RK, qui exploite toutes ces conclusions pour proposer une alternative aux solutions de paramétrisation qui existent dans la littérature, et qui corrige notamment les défauts de ces paramétrisations que nous mettons en évidence dans ce chapitre.

3.2 Recherche exhaustive de K^*

Le paramètre K fixe le nombre de caractéristiques sélectionnées aléatoirement à chaque nœud au cours de la procédure d'induction d'un arbre. Sa valeur est donc choisie dans l'intervalle $[1..M]$, où M représente la dimension de l'espace de description. Ce nombre contrôle la quantité d'aléatoire introduite dans le processus de sélection de caractéristiques, de telle sorte que plus la valeur de K est petite et plus on introduit d'aléatoire. Dans le cas où $K = 1$ par exemple, la caractéristique de chaque règle de partitionnement de l'arbre est choisie entièrement aléatoirement à partir des caractéristiques disponibles. À l'inverse, lorsque $K = M$, l'aléatoire n'in-

tervient pas dans la sélection de la règle de partitionnement, et chaque arbre est alors construit à l'aide d'une procédure d'induction classique. Dans ce cas particulier, l'aléatoire est introduit à l'aide de la méthode de Bagging uniquement.

L'idée première de cette étude est de comparer les différentes valeurs de K traditionnellement utilisées dans la littérature, pour déterminer si l'une de ces valeurs est préférable aux autres, et si elles sont proches de l'optimalité en termes d'erreur en généralisation. Nous montrons notamment dans cette section qu'il existe des problèmes d'apprentissage pour lesquels aucune de ces valeurs par défaut ne permet d'approcher les performances optimales. Dans la sous-section suivante, nous détaillons les données et le protocole expérimental utilisé pour cela.

3.2.1 Bases de Données

Pour cette première expérience, nous avons sélectionné 20 bases de données sur une large variété de problèmes d'apprentissage automatique en termes de nombres de données, de dimensions de l'espace de description et de nombres de classes. Le tableau 3.1 donne une description de ces caractéristiques pour chacune d'elles. 17 de ces bases sont issues de l'**UC Irvine Machine Learning Repository** [Asuncion 2007]. Elle sont repérées dans le tableau 3.1 à l'aide d'une astérisque. Trois bases de données ont été ajoutées à ces 17 premières, concernant exclusivement des problématiques de reconnaissance de chiffres manuscrits :

- la fameuse base de données **MNIST** ([LeCun 1998]), composée d'imagettes de chiffres manuscrits segmentés desquelles ont été extraites pour nos travaux 85 caractéristiques issues d'une pyramide multi-résolutions comme dans [Prudent 2005]. Pour chaque image de chiffre en niveau de gris, 85 valeurs de niveaux de gris moyens ont été extraites pour 4 niveaux de résolution différents, comme illustré sur la figure 3.5.
- la base de données **Digit** a également été constituée à partir d'imagettes de chiffres isolés ([Chatelain 2007]) desquelles ont été extraits trois vecteurs de caractéristiques différents : (i) un premier vecteur de 117 caractéristiques statistiques et structurelles comme décrit dans [Heutte 1998] (ii) un vecteur de 128 caractéristiques extraites des directions du contour comme expliqué dans [Kimura 1994] et (iii) le même vecteur que pour la base MNIST, composé des 85 niveaux de gris de la pyramide multi-résolution. Ces trois vecteurs de caractéristiques ont ensuite été concaténés pour former un seul espace de description de dimension 330. Nous donnons un exemple de quelques unes de ces imagettes dans la figure 3.3.
- la base de données **DigReject** est similaire à Digit, si ce n'est qu'aux imagettes de chiffres manuscrits sont ajoutées des imagettes de caractères manuscrits isolés qui ne sont pas des chiffres ([Chatelain 2007]). Ces images ne sont ensuite pas étiquetées à l'aide des caractères qu'elles représentent (comme par exemple un chiffre compris entre 0 et 9) mais avec une première classe s'il s'agit d'un chiffre ou avec une seconde s'il s'agit d'autre chose (*i.e.* une

FIG. 3.3: Exemple de chiffres manuscrits desquels ont été extraites les données de la base Digits ([Chatelain 2006])

lettre, un signe de ponctuation, ou du bruit). La problématique n'est plus de reconnaître les caractères manuscrits mais de différencier les chiffres des autres formes représentant des données atypiques. L'espace de description de ces données est le même que pour la base Digit (330 caractéristiques). Nous donnons dans la figure 3.4 quelques exemples des imagettes à partir desquelles a été constituée cette base. Dans la ligne d'imagettes du haut sont donnés les formes à rejeter, et dans la ligne du bas les chiffres manuscrits.

La base MNIST a été choisie pour ces expérimentations car elle est composée d'un nombre de données très supérieur aux bases disponibles dans l'UCI repository, *i.e.* 60000 données au total. La base Digit a été sélectionnée car son espace de description est formé à partir d'un sous-ensemble de caractéristiques dont on sait qu'elles sont très discriminantes pour ce type de problèmes (à savoir le vecteur de 117 caractéristiques décrit dans [Heutte 1998]) et d'autres à l'inverse qui le sont peu (comme par exemple les 85 caractéristiques issues de la pyramide multi-résolutions). De cette façon nous disposons d'une base contenant des caractéristiques dont on connait *a priori* la "qualité". La base DigReject quant à elle a été ajoutée aux autres bases pour les mêmes raisons que la base Digit dans un premier temps, mais

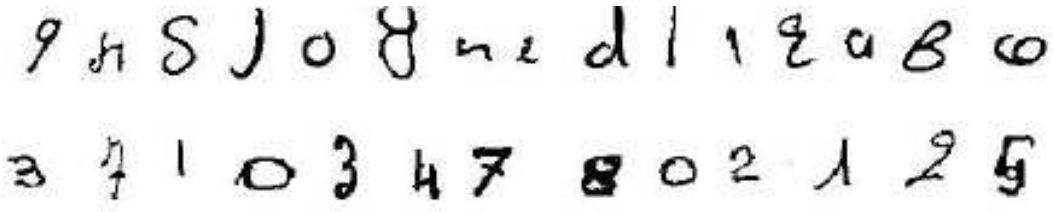

FIG. 3.4: Exemple de chiffres manuscrits desquels ont été extraites les données de la base Digreject ([Chatelain 2006])

aussi parce qu'il s'agit d'une problématique à deux classes intéressante, composée de formes à reconnaitre (les chiffres) et de formes à rejeter (toutes les autres formes) ([Chatelain 2007]).

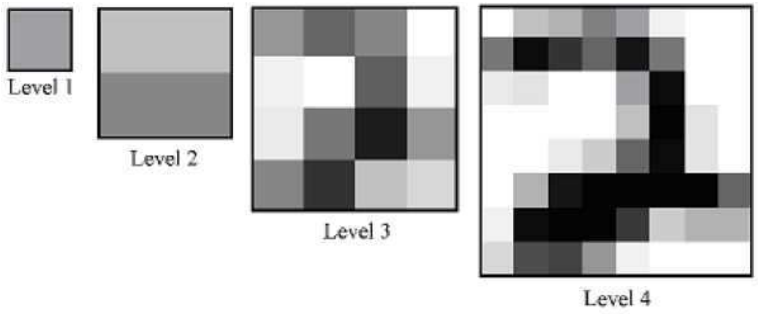

FIG. 3.5: Exemple d'une pyramide multi-résolution pour l'extraction des 85 niveaux de gris moyens sur 4 résolutions différentes.

3.2.2 Protocole Expérimental

Pour ces expérimentations, chacune des bases a tout d'abord été séparée aléatoirement de sorte que deux tiers des données soient destinées à l'apprentissage et le tiers restant utilisé pour la phase de test. Pour fiabiliser les résultats obtenus, cette procédure a été répétée 50 fois, constituant ainsi 50 découpages différents de la même base. Nous notons ces découpages $T_i = (T_{r_i}, T_{s_i})$, pour $i \in [1..50]$ et où T_{r_i} et T_{s_i} représentent respectivement les sous-ensembles d'apprentissage et de test. Ensuite, pour chaque T_i, l'algorithme Forest-RI a été lancé pour plusieurs valeurs de K dans l'intervalle $[1..M]$. Pour chaque base il y a en tout M différentes valeurs possibles pour K. Or, certaines des bases utilisées présentent des espaces de description de dimension relativement grande, à l'image des bases Madelon de dimension 500, Isolet de dimension 616 ou même encore Digit et DigReject de dimension 330. Pour des raisons essentiellement computationnelles, il n'a pas été possible de lancer un apprentissage de forêts pour toutes les M valeurs de K avec toutes les bases utilisées. Pour certaines d'entre elles, l'intervalle $[1..M]$ a

Bases de données	# données	# caractéristiques	# classes
Diabetes*	768	8	2
Digits	38142	330	10
DigReject	14733	330	2
Gamma*	19020	10	2
Isolet*	7797	616	26
Letter*	20000	16	26
Madelon*	2600	500	2
Mfeat-factors*	2000	216	10
Mfeat-fourier*	2000	76	10
Mfeat-karhunen*	2000	64	10
Mfeat-zernike*	2000	47	10
Mnist	60000	85	10
Musk*	6597	166	2
OptDigits*	5620	64	10
Page-blocks*	5473	10	5
Pendigits*	10992	16	10
Segment*	2310	19	7
Spambase*	4610	57	2
Vehicle*	946	18	4
Waveform*	5000	40	3

TAB. 3.1: Description des bases de données. Les bases de données dont le nom est suivi d'une astérisque sont issues de l'UCI Repository.

été échantillonné et les valeurs de K testées ont été prises à intervalles réguliers entre 1 et M. De plus, lorsque l'échantillonnage de cet intervalle ne comprenait pas l'ensemble des quatres valeurs $K = 1$, $K = \lfloor \log_2(M) + 1 \rfloor$, $K = \sqrt{M}$ ainsi que $K = M$, les forêts concernées ont été apprises en plus de celles obtenues avec l'échantillonnage. Dans la suite de ce chapitre, nous notons M' le nombre de valeurs de K testées pour chaque base. Le tableau 3.2 récapitule le nombre de forêts apprises pour chaque base, ainsi que pour chaque découpage de ces bases. A noter que pour l'ensemble des forêts aléatoires apprises au cours des expérimentations présentées dans ce chapitre, le nombre d'arbres qui composent la forêt a été fixé à 100. Nous avons choisi cette valeur sur la base de plusieurs expérimentations complémentaires — dont nous détaillons une partie dans le prochain chapitre — qui ont montré qu'il s'agissait d'une valeur raisonnable en termes de performances

et de temps de traitement.

Dataset	M' pour chaque T_i	# total de forêts apprises
Diabetes	8	$8 \times 50 = 400$
Digits	$\frac{M}{10} = 33$	$33 \times 50 = 1650$
DigReject	$\frac{M}{10} = 33$	$33 \times 50 = 1650$
Gamma	10	$10 \times 50 = 500$
Isolet	$\frac{M}{20} + 1 = 31$	$31 \times 50 = 1550$
Letter	16	$16 \times 50 = 800$
Madelon	$\frac{M}{10} = 50$	$50 \times 50 = 2500$
Mfeat-factors	$\frac{M}{3} = 72$	$72 \times 50 = 3600$
Mfeat-fourier	76	$76 \times 50 = 3800$
Mfeat-karhunen	64	$64 \times 50 = 3200$
Mfeat-zernike	47	$47 \times 50 = 2350$
MNIST	$\frac{M}{2} + 1 = 43$	$43 \times 50 = 2150$
Musk	$\frac{M}{3} + 1 = 56$	$56 \times 50 = 2800$
OptDigits	$\frac{M}{2} = 32$	$32 \times 50 = 1600$
Page-blocks	10	$10 \times 50 = 500$
Pendigits	16	$16 \times 50 = 800$
Ringnorm	20	$20 \times 50 = 1000$
Segment	19	$19 \times 50 = 950$
Spambase	57	$57 \times 50 = 2850$
Twonorm	20	$20 \times 50 = 1000$
Vehicle	18	$18 \times 50 = 900$
Waveform	40	$40 \times 50 = 2000$
Total	771	38550

TAB. 3.2: Nombre de forêts aléatoires induites pour chaque base de données

Notre problème est ensuite de déterminer lequel des algorithmes, parmi les Forest-RI paramétrés avec les différentes valeurs de K, permet d'obtenir les forêts les plus performantes lorsqu'elles sont entrainées et testées sur les mêmes bases de données. Pour répondre à cette problématique, nous nous appuyons sur une étude comparative de cinq tests statistiques présentée dans [Dietterich 1998a]. Dans cet article, Dietterich montre que le test qui correspond le mieux aux protocoles expérimentaux comme le nôtre est le test statistique de McNemar ([Everitt 1977]). Ce

N_{00} = # données mal classées par h_A et h_B	N_{01} = # données mal classées par h_A mais bien classées par h_B
N_{10} = # données mal classées par h_B mais bien classées par h_A	N_{11} = # données bien classées par h_A et h_B

TAB. 3.3: Table de Contingence

test est utilisé ici pour comparer deux algorithmes sur la base de leurs prédictions sur un ensemble de données de test.

S'agissant de confronter deux algorithmes d'apprentissage A et B, induisant respectivement les classifieurs h_A et h_B, pour appliquer le test de McNemar, la table de contingence est tout d'abord construite, comme illustré par le tableau 3.3. Les éléments de cette table de contingence vérifient alors $N = N_{00}+N_{10}+N_{01}+N_{11}$, où N est le nombre total de données dans l'ensemble de test.

Le test de McNemar est ensuite basé sur un test du χ^2 pour l'adéquation de deux distributions de probabilité, qui compare en fait les effectifs attendus sous l'hypothèse nulle, à un ensemble d'effectifs observés. Le principe est de considérer que la statistique X^2 définie par l'équation 3.1 suit une loi de χ^2 avec un certain degré de liberté.

$$X^2 = \frac{(|N_{01} - N_{10}| - 1)^2}{N_{01} + N_{10}} \sim \chi^2_{1,0.05} = 3.841459 \tag{3.1}$$

De cette façon, l'hypothèse nulle H_0, sous laquelle on considère que les deux algorithmes sont équivalents en termes de performance, est rejetée si X^2 est supérieur à $\chi^2_{1,0.05} = 3.841459$. En définitive, ce test nous permet d'obtenir une des trois réponses suivantes :

- H_0 est rejetée et $N_{01} > N_{10}$: l'algorithme B induit des classifieurs significativement plus performants que l'algorithme A.
- H_0 est rejetée et $N_{01} < N_{10}$: l'algorithme A induit des classifieurs significativement plus performants que l'algorithme B.
- H_0 est acceptée : les deux algorithmes comparés induisent des classifieurs statistiquement équivalents en termes de performances.

Avec une telle procédure, il nous est possible alors d'établir laquelle des valeurs testées de K nous permet de produire les forêts les plus performantes en moyenne sur l'ensemble des bases de données testées. De plus, hormis les conclusions des travaux de Dietterich sur la comparaison des tests statistiques, qui comme nous l'avons dit conseille l'utilisation de ce test pour les protocoles expérimentaux comme le nôtre, une autre raison nous a poussé à choisir ce test pour nos comparaisons d'algorithmes : l'utilisation de ce test sur les 50 découpages aléatoires différents pour chaque base va nous permettre de percevoir la variabilité des performances pour une même valeur

de K. C'est un point important étant donné que l'aléatoire intervient énormément dans l'algorithme d'apprentissage testé.

L'algorithme 5 résume le protocole expérimental de ces premiers tests qui visent à comparer les différentes valeurs de K proposées dans la littérature, ainsi qu'à analyser l'évolution des performances de forêts en fonction de ce paramètre. Nous obtenons ainsi deux tableaux de résultats regroupant les taux d'erreurs obtenus en test pour chacune des forêts induites, ainsi que les résultats du test de McNemar pour la comparaison de chaque paire de ces forêts. Dans la section, suivante nous détaillons et discutons ces résultats.

Algorithme 5 Protocole Expérimental 1

Entrée : N le nombre de données disponibles.
Entrée : M le nombre de caractéristiques disponibles.
Entrée : M' le nombre de valeurs testées pour K.
Entrée : m le pas d'échantillonnage pour les valeurs de K.
Sortie : $\epsilon_{RI}[50][M']$ un tableau 2D pour conserver les taux d'erreur obtenus avec Forest-RI.
Sortie : $\mathcal{M}[50][M'][M']$ un tableau 3D pour conserver les réponses au test de McNemar.

1: **pour** $i \in [1..50]$ **faire**
2: Tirer aléatoirement sans remise $\frac{2}{3}$ des données disponibles pour former l'ensemble d'apprentissage T_{r_i}. Les données restantes forment alors l'ensemble de test T_{s_i}, le couple (T_{r_i}, T_{s_i}) est noté T_i.
3: **pour** k de 1 à M par pas de m **faire**
4: $H_{RI}(j) \leftarrow$ induire une RF avec l'algorithme Forest-RI, avec $L = 100$ et $K = k$, sur T_{r_i}.
5: $\epsilon_{RI}(i)(j) \leftarrow$ erreur de $H_{RI}(j)$ sur T_{s_i}.
6: **pour** $(h_m, h_n) \in H_{RI} \times H_{RI}$ **faire**
7: $\mathcal{M}(i)(h_m)(h_n) \leftarrow$ résultat du test de McNemar avec les classifieurs (h_m, h_n) sur T_{s_i}.

3.2.3 Résultats et Discussion

Nous présentons dans un premier temps deux tableaux dans lesquels nous synthétisons les résultats obtenus en appliquant le protocole que nous venons de présenter, qui est résumé par l'algorithme 5. Ce protocole nous donne un ensemble de $M' \times 50$ taux d'erreur par base de données, ainsi que $\frac{M' \times (M'-1)}{2} \times 50$ résultats aux tests de McNemar — *i.e.* un pour chaque paire de forêts induites avec chaque T_i. Dans un premier temps nous ne présentons que les résultats obtenus pour les trois valeurs proposées dans la littérature et présentées en introduction de ce chapitre, ainsi que les résultats obtenus pour $K = M$. Nous présentons également les meilleurs résultats obtenus avec K, dont nous notons K^* la valeur correspondante : pour chaque base, la moyenne des taux d'erreur

sur l'ensemble des T_i a été calculée, et la valeur de K qui a minimisé l'erreur moyenne a été retenue comme valeur optimale de K, notée K^*. Ces tableaux présentent ainsi les résultats de 5 paramétrisations différentes de l'algorithme Forest-RI.

Une première analyse du tableau 3.4 consiste à chercher une correspondance entre certaines propriétés des bases de données et les résultats obtenus, et plus particulièrement la valeur de K^*. On peut penser dans un premier temps que cette valeur optimale de K dépende directement du nombre de données N, du nombre de caractéristiques M et/ou du nombre de classes du problème traité c, c'est-à-dire qu'il existe une fonction $K^* = g(N, M, c)$. Pour nous en faire une idée, nous regroupons toutes ces valeurs dans le tableau 3.6. On constate alors assez rapidement qu'il n'est pas évident d'établir ce lien. Intuitivement, il paraîtrait logique que cette valeur de K^* depende même simplement du nombre de caractéristiques dans la mesure où K désigne la taille des sous-ensembles de caractéristiques sélectionnées à chaque nœud de l'arbre au cours de son induction. Pourtant ce lien, s'il existe, semble trop complexe pour que l'on puisse le déterminer simplement en regardant ces deux tableaux. S'il existe une fonction $K^* = g(M, N, c)$, ou même $K^* = g(M)$, elle n'est pas suffisamment intuitive pour que l'on puisse la déterminer de ce simple "coup d'œil".

Une deuxième constation faite à partir du tableau 3.4 est qu'aucune des trois valeurs de K qui nous sont données dans la littérature ne correspond à K^* dans une majorité des cas. C'est même rarement le cas pour chacune de ces trois valeurs. K^* est égal à $\sqrt{M}$ pour 3 des 20 bases de données (Gamma, MFeat-Zernike et Page-blocks), à $\lfloor \log_2(M) + 1 \rfloor$ pour également 3 d'entre elles (MFeat-Karhunen, Pendigits et Vehicle) et n'est jamais égal à 1. Ces 3 valeurs de K sont même parfois très éloignées de la valeur de K^*, à l'image des bases DigRejects (pour laquelle $K^* = 150$ alors que $\sqrt{M} = 18$ et $\lfloor \log_2(M)+1 \rfloor = 9$), Madelon ($K^* = 261$, $\sqrt{M} = 22$ et $\lfloor \log_2(M)+1 \rfloor = 10$) ou encore Musk ($K^* = 88$, $\sqrt{M} = 13$ et $\lfloor \log_2(M)+1 \rfloor = 18$). Dans un premier temps, on pourrait donc conclure de ce tableau que les trois valeurs proposées dans la littérature ne sont pas satisfaisantes pour tirer le meilleur parti des méthodes de forêts aléatoires. Cependant en examinant un peu plus en détail les valeurs des écart-types ainsi que les écarts de performances obtenus pour deux valeurs immédiatement consécutives de K, on peut nuancer cette conclusion. Ces différences dans les taux d'erreur moyens sont en effet relativement faibles pour deux valeurs de K très proches. C'est la raison pour laquelle il est judicieux d'examiner les résultats des tests de McNemar qui nous donnent une meilleure idée de la significativité statistique de ces résultats. Les résultats à ce test sont reportés dans le tableau 3.5, à l'aide des nombres de cas pour lesquels chacune des trois réponses possibles a été obtenue. Pour chaque base de données et pour chaque paire d'algorithmes — *i.e.* chaque paire de valeurs de K — nous disposons de 50 résultats au test réparties en 3 cas possibles. Nous reportons dans les cellules de ce tableau les effectifs pour chacun de ces 3 cas. Dans l'immédiat, les résultats qui nous intéressent figurent dans les trois premières colonnes de valeurs, puisque ce sont

Dataset	K^*	$K_{\sqrt{M}}$	$K_{log_2(M)+1}$	K_1	K_M	Forest-RK
Diabetes	23.51 ± 1.76 (2)	23.74 ± 1.96 (3)	23.74 ± 1.96 (3)	24.56 ± 1.95	24.61 ± 1.80	23.71 ± 2.34
Digits	2.24 ± 0.12 (10)	2.25 ± 0.13 (18)	2.24 ± 0.12 (9)	2.70 ± 0.13	3.36 ± 0.23	2.24 ± 0.13
DigReject	7.12 ± 0.34 (150)	7.58 ± 0.34 (18)	7.85 ± 0.35 (9)	8.93 ± 0.35	7.30 ± 0.34	7.16 ± 0.33
Gamma	12.17 ± 0.33 (3)	12.17 ± 0.33 (3)	12.22 ± 0.33 (4)	12.38 ± 0.32	12.34 ± 0.33	12.30 ± 0.38
Isolet	5.72 ± 0.53 (40)	5.85 ± 0.44 (25)	6.38 ± 0.52 (9)	13.05 ± 0.58	8.86 ± 0.63	6.87 ± 0.45
Letter	4.16 ± 0.28 (3)	4.23 ± 0.23 (4)	4.30 ± 0.26 (5)	5.19 ± 0.27	7.36 ± 0.48	4.28 ± 0.25
Madelon	17.73 ± 1.60 (261)	30.50 ± 1.94 (22)	34.79 ± 1.26 (10)	46.12 ± 1.56	18.60 ± 1.92	18.34 ± 1.52
Mfeat-factors	3.56 ± 0.71 (10)	3.57 ± 0.58 (15)	3.58 ± 0.73 (9)	4.27 ± 0.72	4.62 ± 0.76	3.48 ± 0.61
Mfeat-fourier	16.81 ± 1 (19)	17.11 ± 1.05 (9)	17.25 ± 1.02 (7)	22.18 ± 1.19	18.66 ± 1.32	17.00 ± 0.98
Mfeat-karhunen	4.30 ± 0.68 (6)	4.33 ± 0.69 (7)	4.30 ± 0.68 (6)	7.14 ± 0.83	8.38 ± 1.11	4.45 ± 0.69
Mfeat-zernike	22.26 ± 1.06 (7)	22.26 ± 1.06 (7)	22.56 ± 1.02 (5)	23.86 ± 0.90	24.87 ± 1.33	22.27 ± 1.14
MNIST	5.03 ± 0.14 (17)	5.19 ± 0.16 (10)	5.32 ± 0.17 (8)	6.53 ± 0.17	6.52 ± 0.24	5.15 ± 0.14
Musk	2.34 ± 0.30 (88)	2.40 ± 0.29 (13)	2.50 ± 0.26 (8)	4.03 ± 0.32	2.45 ± 0.34	2.34 ± 0.31
OptDigits	1.97 ± 2.28 (11)	2.03 ± 0.34 (8)	2.10 ± 0.33 (6)	2.81 ± 0.42	4.05 ± 0.43	1.95 ± 0.29
Page-blocks	2.60 ± 0.28 (3)	2.60 ± 0.28 (3)	2.62 ± 0.29 (4)	2.84 ± 0.28	2.76 ± 0.30	2.62 ± 0.33
Pendigits	1.00 ± 0.17 (5)	1.05 ± 0.16 (4)	1.00 ± 0.17 (5)	1.23 ± 0.18	1.50 ± 0.23	1.01 ± 0.16
Segment	2.29 ± 0.53 (7)	2.30 ± 0.44 (4)	2.38 ± 0.52 (5)	3.05 ± 0.50	2.64 ± 0.57	2.34 ± 0.44
Spambase	$4,83 \pm 0.53$ (6)	$4,98 \pm 0.49$ (8)	$4,93 \pm 0.47$ (7)	5.00 ± 0.45	5.41 ± 0.45	5.11 ± 0.59
Vehicle	24.75 ± 2.02 (5)	25.15 ± 2.14 (4)	24.75 ± 2.02 (5)	26.10 ± 1.95	25.47 ± 2.11	24.80 ± 2.09
Waveform	14.92 ± 0.81 (4)	14.97 ± 0.78 (6)	14.97 ± 0.78 (6)	16.46 ± 0.83	16.38 ± 0.93	14.90 ± 0.84

TAB. 3.4: Taux d'erreur moyens obtenus pour différentes valeurs de K. Pour chaque cellule du tableau, nous donnons la moyenne sur les $50T_i$, les écart-types ainsi que la valeur de K lorsque c'est nécessaire.

Algo A	K^*			$K_{\sqrt{M}}$		K_1	Forest-RK	
Algo B	K_1	$K_{\sqrt{M}}$	K_{log}	K_1	K_{log}	K_{log}	K^*	$K_{\sqrt{M}}$
Diabetes	2/**48**/0	0/**49**/1	0/**50**/0	2/**48**/0	0/**50**/0	0/**50**/0	0/**50**/0	2/**48**/0
Digits	**48**/2/0	2/**48**/0	1/**49**/0	**48**/2/0	0/**50**/0	0/0/**50**	3/**46**/1	1/**46**/3
DigReject	**50**/0/0	**25**/**25**/0	**37**/13/0	**50**/0/0	13/**37**/0	0/2/**48**	1/**45**/4	0/24/**26**
Gamma	3/**47**/0	1/**49**/0	2/**47**/1	4/**46**/0	1/**48**/1	0/**46**/4	3/**46**/1	1/**49**/0
Isolet	**50**/0/0	0/**50**/0	21/**29**/0	**50**/0/0	23/**27**/0	0/0/**50**	3/**46**/1	1/**49**/0
Letter	**50**/0/0	3/**45**/2	3/**47**/0	**50**/0/0	4/**46**/0	0/1/**49**	5/**45**/0	0/**50**/0
Madelon	**50**/0/0	**50**/0/0	**50**/0/0	**50**/0/0	**45**/5/0	0/0/**50**	5/**45**/0	0/0/**50**
Mfeat-fac	4/**46**/0	0/**49**/1	0/**49**/1	4/**46**/0	0/**50**/0	0/**43**/7	0/**49**/1	1/**49**/0
Mfeat-fou	**47**/3/0	1/**49**/0	1/**49**/0	**46**/4/0	1/**49**/0	0/2/**48**	1/**49**/0	1/**49**/0
Mfeat-kar	**45**/5/0	2/**47**/1	3/**46**/1	**42**/8/0	1/**49**/0	0/9/**41**	2/**48**/0	1/**48**/0
Mfeat-zer	7/**43**/0	0/**49**/1	1/**48**/1	7/**43**/0	1/**48**/1	0/**40**/10	1/**49**/0	0/**50**/0
MNIST	**50**/0/0	7/**43**/0	23/**27**/0	**50**/0/0	6/**43**/1	0/0/**50**	10/**40**/0	4/**44**/2
Musk	**50**/0/0	1/**49**/0	6/**44**/0	**50**/0/0	4/**46**/0	0/0/**50**	0/**50**/0	0/**48**/0
OptDigits	**40**/10/0	0/**50**/0	1/**49**/0	**40**/10/0	0/**50**/0	0/10/**40**	2/**48**/0	1/**49**/0
Page-blocks	5/**45**/0	1/**49**/0	0/**50**/0	4/**46**/0	0/**50**/0	0/**45**/5	0/**50**/0	1/**49**/0
Pendigits	6/**44**/0	0/**49**/1	0/**49**/1	5/**45**/0	1/**49**/0	0/**43**/7	1/**47**/2	2/**47**/1
Segment	10/**40**/0	1/**48**/1	0/**50**/0	9/**41**/0	0/**48**/2	0/**37**/13	2/**48**/0	0/**50**/0
Spambase	0/**50**/0	1/**49**/0	1/**49**/0	0/**48**/2	1/**49**/0	1/**49**/0	9/**41**/0	2/**48**/0
Vehicle	3/**47**/0	0/**50**/0	0/**49**/1	2/**48**/0	0/**49**/1	0/**46**/4	0/**49**/1	0/**49**/1
Waveform	20/**30**/0	1/**49**/0	1/**49**/0	21/**29**/0	2/**46**/2	0/**30**/20	1/**49**/0	2/**48**/0

TAB. 3.5: Résultats des tests de McNemar. Dans chaque cellule sont présentés trois nombres : (i) le nombre de T_i pour lesquels l'algorithme A s'est montré plus performant que l'algorithme B (ii) le nombre de T_i pour lesquels aucune différence significative n'a été observée (iii) le nombre de T_i pour lesquels l'algorithme B s'est montré plus performant que l'algorithme A

Bases de données	N	M	c	K^*
Diabetes*	768	8	2	2
Digits	38142	330	10	10
DigReject	14733	330	2	150
Gamma*	19020	10	2	3
Isolet*	7797	616	26	40
Letter*	20000	16	26	3
Madelon*	2600	500	2	261
Mfeat-factors*	2000	216	10	10
è Mfeat-fourier*	2000	76	10	19
Mfeat-karhunen*	2000	64	10	6
Mfeat-zernike*	2000	47	10	7
Mnist	60000	85	10	17
Musk*	6597	166	2	88
OptDigits*	5620	64	10	11
Page-blocks*	5473	10	5	3
Pendigits*	10992	16	10	5
Segment*	2310	19	7	7
Spambase*	4610	57	2	6
Vehicle*	946	18	4	5
Waveform*	5000	40	3	4

TAB. 3.6: Nombres de données, de caractéristiques et de classes, ainsi que valeur de K^* pour chaque base de données.

les résultats qui nous permettent de comparer chacune des valeurs par défaut à la meilleure valeur relevée durant l'expérience. On peut alors constater que pour une majorité des cas les valeurs $K = \sqrt{M}$ et $K = \lfloor \log_2(M) + 1 \rfloor$ permettent d'obtenir des performances significativement comparables à K^*. Dans 2 cas sur 20 seulement ces deux valeurs se sont toutes deux montrées clairement sous-optimales (DigReject et Madelon). La valeur $K = 1$ quant à elle ne permet d'approcher "l'optimalité" que dans la moitié des cas seulement (11 cas sur 20).

Pour répondre aux deux premières questions que nous nous sommes posées en introduction, à savoir tout d'abord *parmi les valeurs de la littérature, y en a-t-il une qui permet d'obtenir de meilleurs résultats en moyenne que les autres ?* et ensuite *existe-t-il d'autres valeurs plus pertinentes ?*, il nous reste à déterminer si une des deux valeurs $K = \sqrt{M}$ et $K = \lfloor \log_2(M) + 1 \rfloor$ se montre préférable en moyenne à l'autre — nous pouvons d'ores et déjà écarter la valeur $K = 1$. Pour cela, il suffit d'examiner la cinquième colonne de résultats du tableau 3.5. Cette colonne nous indique laquelle des deux valeurs a permis le plus souvent d'obtenir des résultats significativement meilleurs. On constate alors que dans la majorité des cas, les deux valeurs donnent des résultats statistiquement équivalents, avec un léger avantage pour la valeur $K = \sqrt{M}$, qui est préférable à $K = \lfloor \log_2(M) + 1 \rfloor$ pour la base Madelon et qui pour les autres bases présente globalement un plus grand nombre de victoires que $K = \lfloor \log_2(M) + 1 \rfloor$. En définitive, on peut donc conclure de cette première analyse que $K = \sqrt{M}$ est, dans une majorité des cas, une solution de paramétrisation raisonnable pour le réglage du paramètre K de Forest-RI.

Cependant, le problème des bases de données pour lesquelles aucune de ces trois valeurs n'est une bonne solution de paramétrisation se pose toujours. Les deux bases pour lesquelles nous avons constaté que la valeur de K^* était très éloignée de $K = \sqrt{M}$, de $K = \lfloor \log_2(M) + 1 \rfloor$ et de $K = 1$ (DigReject et Madelon), le sont pour K mais aussi et surtout le sont en termes de performances.

Pour amorcer une analyse de ce phénomène, nous proposons d'étudier l'évolution globale des taux d'erreur en test en fonction de la valeur de K. Pour cela nous présentons l'ensemble des résultats que nous avons obtenus avec nos tests systématiques de toutes les valeurs de K sous la forme de courbes. Nous présentons ces courbes dans les figures 3.6 et 3.7. Sur chacun de ces diagrammes est représentée en rouge l'évolution du taux d'erreur moyen pour des valeurs de K croissantes allant de 1 à M. Les écart-types sont quant à eux représentés avec des barres d'erreur sur chaque figure. Enfin les taux d'erreur obtenus avec Forest-RI/K^* sont marqués à l'aide d'un cercle sur les courbes.

Dans un premier temps, on constate que certaines des bases de données que nous avons testées présentent des écarts-type importants en comparaison avec l'évolution des taux d'erreur en fonction de K. On observe notamment des barres d'erreurs très grandes par rapport aux écarts des taux d'erreur d'une valeur de K à une autre, sur les diagrammes des bases Diabetes, Gamma, Page-blocks et Vehicle. Il est difficile alors de déduire un schéma d'évolution des taux d'erreur en fonction de K à cause de cette variabilité. Une deuxième raison qui rend l'interprétation des résultats de ces quatres bases difficile est qu'elles sont décrites par des espaces de description de dimension faible. Par conséquent, les valeurs possibles pour K sont peu nombreuses et l'évolution des performances vis à vis de ce paramètre est assez restreinte. Nous avons par conséquent décidé de ne pas inclure ces quatres bases dans l'analyse de ces courbes.

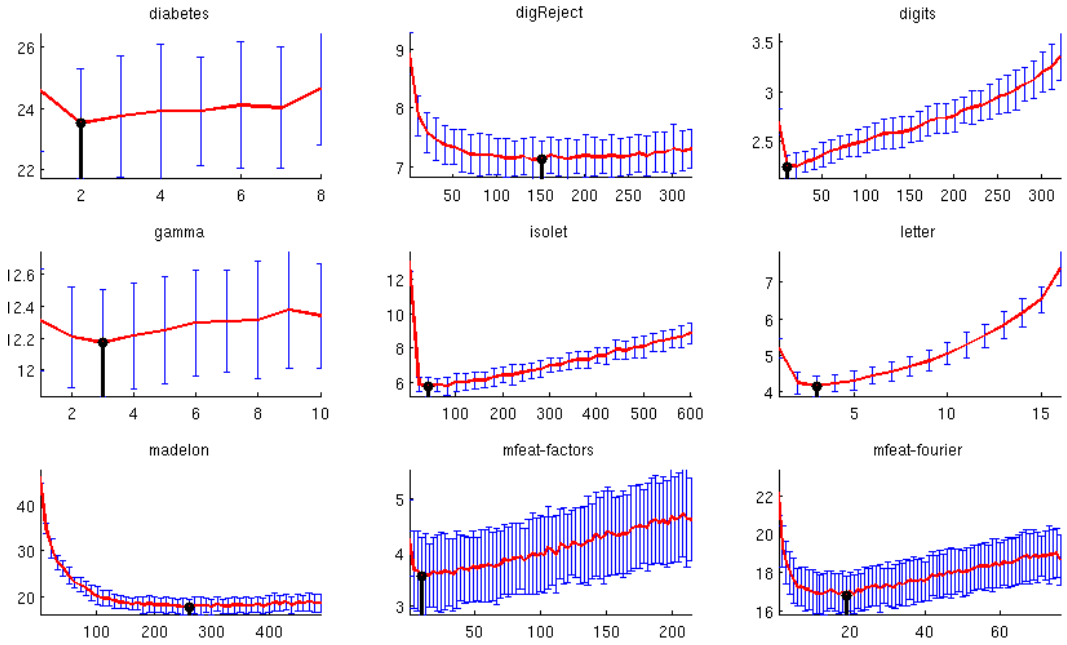

FIG. 3.6: Taux d'erreur moyen en fonction des valeurs de K. Les taux d'erreur minimum sont marqués sur chaque diagramme.

Dans un second temps, on peut distinguer deux principaux schémas d'évolution de ces taux d'erreur en fontion de K : (i) pour 13 des 16 bases de données étudiées (Digits, Isolet, Letter, Mfeat-factors, Mfeat-fourier, Mfeat-karhunen, Mfeat-zernike, MNIST, OptDigits, Pendigits, Segment, Spambase et Waveform) le taux d'erreur minimum est atteint pour une valeur très faible de K en comparaison avec M (souvent inférieur à $\frac{M}{10}$) et l'augmentation des valeurs à partir de ce point jusqu'à $K = M$ est monotone et presque linéaire (ii) pour 3 d'entre elles (DigReject, Madelon et Musk) cette augmentation est pratiquement inexistante et la courbe est décroissante jusqu'à converger vers une valeur autour de laquelle les taux d'erreur se stabilisent jusqu'à $K = M$. Le taux d'erreur minimum est alors atteint pour une valeur de K bien plus élevée, *i.e.* toujours supérieure à $\frac{M}{2}$.

Dans la suite de nos expérimentations, nous sommes allés un peu plus en profondeur dans l'analyse de ces comportements afin de comprendre les raisons de ces différences importantes dans le comportement des forêts vis à vis de ce paramètre K. Nous nous sommes notamment intéressés à la "qualité" des caractéristiques qui composent l'espace de description de chaque base de données.

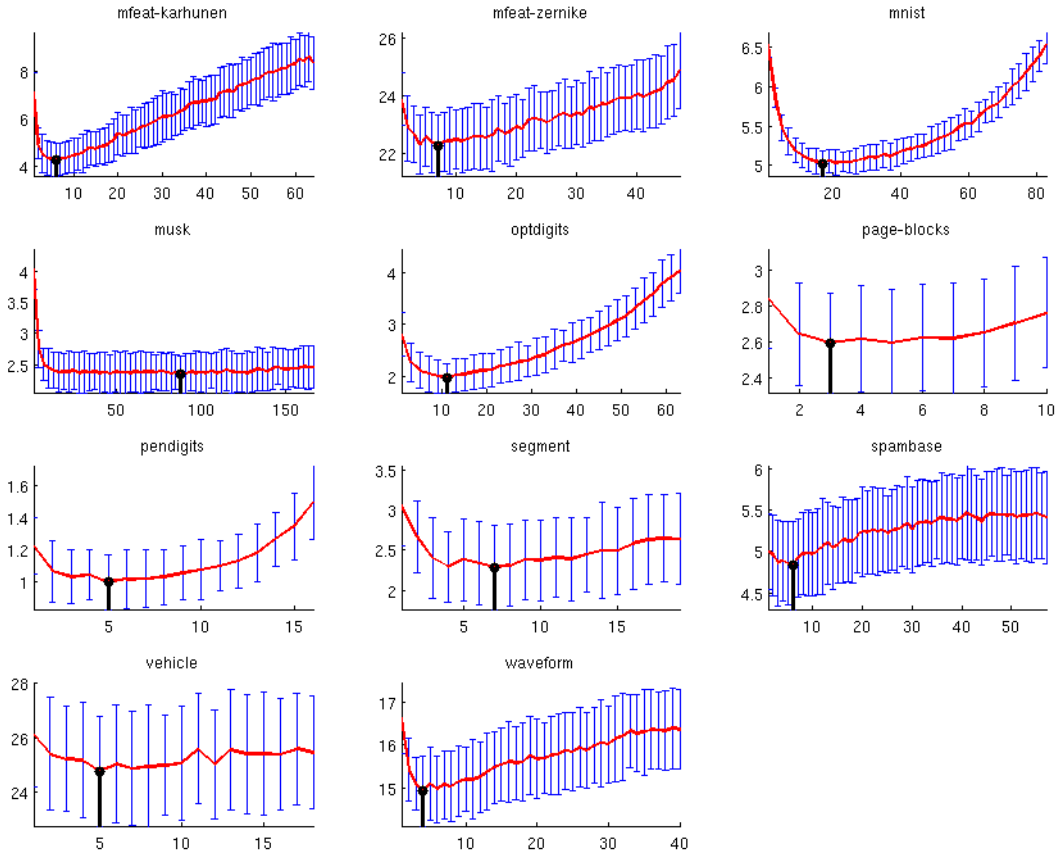

FIG. 3.7: Taux d'erreur moyen en fonction des valeurs de K. Les taux d'erreur minimum sont marqués sur chaque diagramme.

3.3 Étude de la qualité des caractéristiques

Les comportements que nous venons d'observer dans la section précédentes sont relativement difficiles à expliquer à l'aide uniquement des caractéristiques des bases de données, *i.e.* de leur nombre de données, la dimensionnalité de leur espace de description et nombre de classes (*cf.* tableau 3.6). Geurts *et al.* suggèrent dans [Geurts 2006] que la nature des caractéristiques pourrait expliquer les comportements particuliers qu'ils ont constatés avec leurs Extra-Trees sur certaines bases de données. Ils conjecturent que plus il y a de caractéristiques non discriminantes et plus la valeur de K^* est élevée, puisqu'une grande valeur pour K donne de meilleures chances de filtrer ces caractéristiques non discriminantes. Cette affirmation nous a conduit à nous intéresser à la qualité des caractéristiques pour expliquer les variations de performances en fonction de K. Pour ce faire, nous avons décidé d'évaluer l'informativité de chaque caractéristique en mesurant son gain d'information individuel et intrinsèque. De façon générale, le gain d'information mesure la variation d'entropie entre un état initial et un état après l'apport d'information

([Guyon 2003]). Cette mesure est souvent utilisée pour le choix du critère de partitionnement dans les arbres de décision. Elle mesure dans ce cas l'homogénéité d'un groupe de données en fonction de leur classe (*cf.* section 2.2.2). C'est la mesure que nous avons par ailleurs utilisée pour l'induction de nos arbres aléatoires. C'est pourquoi il nous a semblé judicieux de l'utiliser ici pour évaluer l'informativité globale des caractéristiques. Pour cela, nous mesurons le gain d'information pour chaque caractéristique, indépendamment des forêts, et tel que définis dans la section 2.2.2 du chapitre 2.

Pour nos expérimentations, la valeur du gain d'information a été mesurée pour toutes les caractéristiques sur chacun des T_{r_i} de chaque base de données. De cette façon, $50 \times M$ valeurs de gain d'information ont été calculées pour chaque base. Ces valeurs ont ensuite été moyennées sur les T_{r_i}. La figure 3.8 synthétise ces résultats sous la forme de courbes de nombres cumulés de caractéristiques en fonction de leur valeur de gain d'information moyen, de telle sorte que chaque point de ces courbes indique le nombre de caractéristiques pour lesquelles ce gain d'information est inférieur ou égal à la valeur correspondante sur l'axe des abscisses. Comme nous l'avons expliqué précédemment, les quatres bases de données Diabetes, Gamma, Page-blocks et Vehicle ont été retirées de cette analyse, aussi pour ne pas surcharger la figure 3.8 nous ne les avons pas inclues ici. Cette représentation permet de visualiser simultanément la qualité des caractéristiques ainsi que la proportion d'entre elles qui ne sont pas discriminantes. On peut observer sur cette figure que les valeurs de gain d'information sont gloablement plus grandes (typiquement supérieures à 0.1) pour les bases de données pour lesquelles la valeur de K^* est petite par rapport à M, comme c'est le cas par exemple pour Digits, Mfeat-factors et Mfeat-Karhunen pour lesquelles K^* est plus petit que $\frac{M}{10}$. À l'inverse, pour les trois bases de données pour lesquelles K^* est plus grand que $\frac{M}{2}$ (DigReject, Madelon et Musk), le gain d'information est toujours inférieur à environ 0.1. Cela prouve que l'information intrinsèque apportée par les caractéristiques explique très fortement les variations de performances en fonction de K.

Le choix de K permet en fait d'établir un compromis entre deux besoins : (i) forcer, via l'aléatoire, le processus d'induction d'arbre à diversifier les choix de la règle de partitionnement dans le but de créer des arbres différents les uns des autres, (ii) choisir des caractéristiques discriminantes pour la règle de partitionnement dans le but d'induire des arbres suffisamment performants. Trop d'aléatoire injecté dans la sélection de la règle de partitionnement produit des arbres qui globalement ne sont pas adaptés au problème, alors que ne pas injecter suffisamment d'aléatoire crée des arbres qui tendent à sur-apprendre, et donc à être relativement similaires les uns aux autres en terme de prédictions. K permet donc d'établir un juste compromis entre les performances individuelles des arbres, et la diversité dans l'ensemble. Cependant, nous pensons que l'impact sur les performances de la "quantité" d'aléatoire injecté dans le processus de sélection du critère de partitionnement, est fortement modifié par la qualité globale des caractéristiques. S'il y a trop peu de caractéristiques discriminantes, l'aléatoire va rapidement provoquer une détérioration des performances, et donc rapidement détériorer le

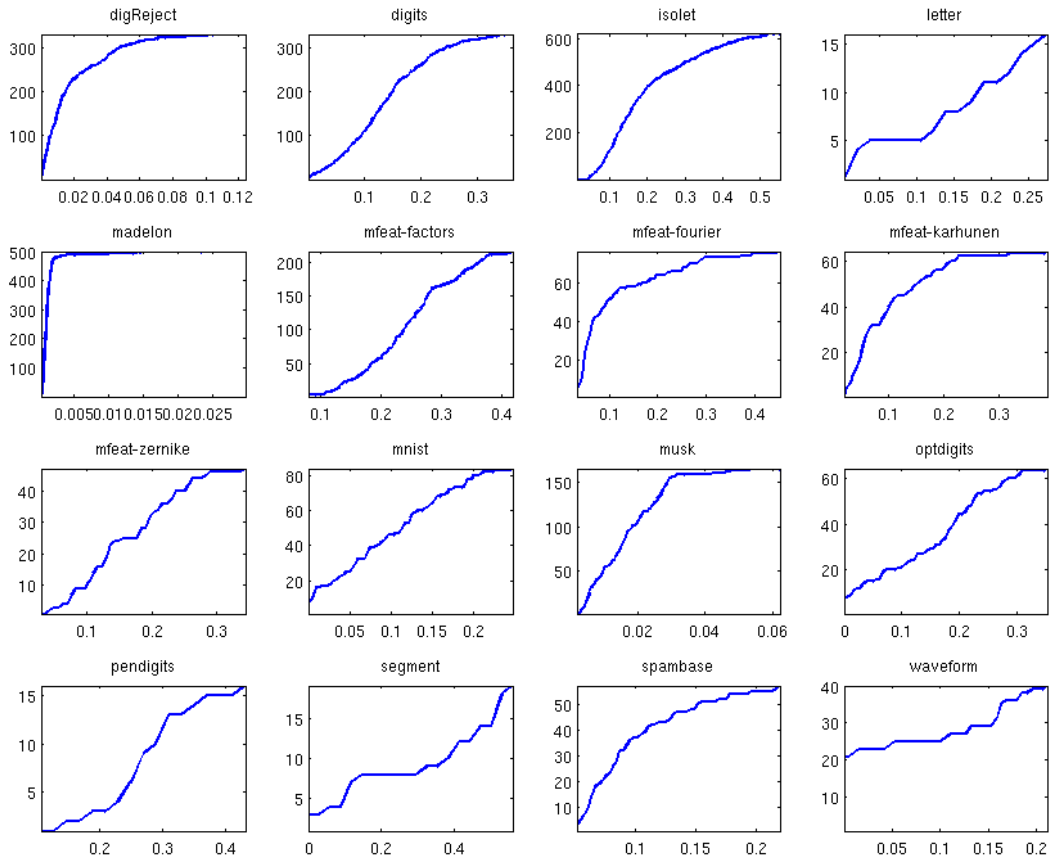

FIG. 3.8: Nombre cumulé de caractéristiques en fonction des valeurs de gain d'information.

compromis "performance individuelle des arbres / diversité de l'ensemble". À l'inverse, une quantité importante de caractéristiques fortement discriminantes va favoriser le sur-apprentissage des arbres et donc affaiblir l'effet de l'aléatoire dans la sélection du partitionnement. Nous pensons que c'est la raison pour laquelle les trois bases de données pour lesquelles le gain d'information est faible (DigReject, Madelain et Musk), présentent des courbes de taux d'erreur qui n'augmentent pas significativement pour des valeurs croissantes de K. Par conséquent, il semblerait que l'information apportée par les caractéristiques soit une propriété importante à prendre en compte pour déterminer une régle de paramétrisation pour l'algorithme Forest-RI.

Avec cette analyse nous avons cerné l'origine du problème de comportement atypique des forêts, mais nous ne l'avons pas résolue. Il faut maintenant concevoir un procédé d'induction qui adapte les forêts induites à la "qualité" de l'espace de description. Pour cela, nous proposons dans la prochaine section un algorithme original qui met directement en application ces conclusions. Nous appelons cet algorithme Forest-RK. La motivation de la conception de cet algorithme est double : (i) éviter

la recherche exhaustive, et donc gloutonne, de la meilleure valeur de K, en testant toutes les valeurs possibles les unes après les autres (ii) proposer une alternative aux valeurs par défaut qui sont proposées dans la littérature pour K, en adaptant le comportement des forêts aléatoires aux propriétés de l'espace de description du problème traité. Nous donnons les détails de cette procédure dans la section suivante, et présentons une évaluation de l'algorithme.

3.4 Forest-RK

L'idée première de cet algorithme est de reprendre le processus de Forest-RI, mais au lieu de fixer *a priori* la valeur de K, celle-ci est choisie aléatoirement à chaque nœud des arbres — d'où Forest-RK pour Random Forest - Random K. De cette façon, on espère pouvoir créer de la diversité au sein de l'ensemble comme le fait Random Feature Selection, tout en laissant une chance à l'algorithme d'induction de filtrer les caractéristiques non discriminantes lorsqu'elles sont en grand nombre dans la base. Cependant, lorsque cela n'est pas le cas, et que l'espace de description est de suffisamment "bonne qualité", il n'est pas forcément souhaitable de choisir complètement aléatoirement cette valeur, *i.e.* par tirage aléatoire selon un loi uniforme.

C'est pourquoi le principe de cet algorithme est d'utiliser un prétraitement pour guider le tirage aléatoire de la valeur de K en fonction des propriétés de l'espace de description. À partir des résultats et des conclusions obtenus avec les expérimentations présentées dans ce chapitre, on sait que lorsque suffisamment de caractéristiques de l'espace de description présentent une valeur de gain d'information globalement supérieure à 0.1, la valeur de K peut être raisonnablement fixée à $\sqrt{M}$. Aussi notre idée est d'analyser en prétraitement la qualité des caractéristiques, à l'aide du gain d'information et en fonction de l'ensemble des données bootstrap, et lorsque ce gain fournit en proportion suffisante des valeurs supérieures à 0.1, d'effectuer à chaque nœud de l'arbre, un tirage aléatoire de la valeur de K, en suivant une loi gaussienne de moyenne $\sqrt{M}$ et d'écart-type relativement faible, *i.e.* égal à $\lceil \frac{M}{50} \rceil$. De cette façon, on concentre les valeurs de K autour de $\sqrt{M}$ tout en maintenant le processus de sélection aléatoire. Dans la situation inverse, c'est-à-dire lorsque l'ensemble des caractéristiques fournit très majoritairement des valeurs de gain d'information inférieures à 0.1, on met alors en place un tirage aléatoire de K suivant une loi uniforme dans l'intervalle $[1..M]$.

La procédure mise en place dans Forest-RK pour l'induction d'un arbre de décision au sein de la forêt, peut être résumée de la façon suivante :

- Tirer aléatoirement avec remise N données d'apprentissage pour former l'ensemble booststrap.
- Mesurer les valeurs de gain d'information pour chacune des caractéristiques de l'espace de description en fonction de cet ensemble bootstrap.
- Calculer la proportion p de caractéristiques présentant un gain d'information inférieur à 0.1.

- A chaque nœud (non terminal) de l'arbre :
  - si $p > \alpha \times M$ alors choisir aléatoirement un entier K entre 1 et M, selon une loi uniforme.
    sinon choisir aléatoirement un entier K, selon une loi normale de moyenne $\sqrt{M}$ et d'écart-type $\lceil \frac{M}{50} \rceil$.
  - Tirer aléatoirement sans remise K caractéristiques.
  - Construire la meilleure règle de partitionnement possible avec l'une de ces K caractéristiques.

Dans cette procédure, la variable α est un seuil défini sous la forme d'une proportion de M, M représentant toujours le nombre de caractéristiques disponibles. Ce seuil doit être relativement grand puisqu'il indique la proportion de caractéristiques ayant un gain d'information inférieur à 0.1, au delà de laquelle on considère que la paramétrisation Forest-RI/$K_{\sqrt{M}}$ n'est plus adaptée. Nous avons vu avec la figure 3.8 de la section précédente que cette proportion est importante pour les 3 bases concernées, *i.e.* DigReject, Madelon et Musk. Elle est de 100% pour les bases Madelon et Musk, et proche de 99% pour la base DigReject. Pour les autres bases en revanche elle est systématiquement inférieure à 70%. Il nous faut donc fixer ce seuil à une valeur comprise entre 0.70 et 1. Les résultats avec n'importe laquelle des valeurs comprises dans cet intervalle sont relativement identiques pour ces 20 bases de données, donc nous avons choisi de fixer cette valeur à 0.80 dans nos expérimentations. Nous avons également fait le choix de fixer l'écart type de la loi normale à $\lceil \frac{M}{50} \rceil$. Pour les espaces de description de dimension faible, lorsque la valeur obtenue par le tirage aléatoire est inférieur à un, la valeur choisie pour K est ramenée à 1 systématiquement. Le choix de cette valeur pour l'écart-type est empirique. C'est-à-dire que parmi les valeurs que nous avons testées, à savoir $\lceil \frac{M}{2} \rceil$, $\lceil \frac{M}{4} \rceil$, $\lceil \frac{M}{10} \rceil$, $\lceil \frac{M}{20} \rceil$ et $\lceil \frac{M}{50} \rceil$, il s'agit de la valeur qui s'est montrée la plus efficace. En outre, plus l'écart-type de la gaussienne est important et plus les valeurs tirées aléatoirement pourront s'écarter du centre $K = \sqrt{M}$. Or, il est important de limiter ces écarts pour ne pas choisir trop souvent des valeurs très proches de 1 ou de M, dont nous avons vu avec les expérimentations précédentes qu'elles n'étaient pas une bonne solution de paramétrisation. Il nous faut donc le plus possible resserer la gaussienne autour de son centre. La valeur $\lceil \frac{M}{50} \rceil$ s'est avérée dans nos expérimentations la valeur la plus efficace pour cela.

Nous donnons avec les algorithmes 6 et 7 la procédure détaillée de Forest-RK.

Dans la section suivante nous présentons les expérimentations menées pour confronter notre algorithme à l'algorithme Forest-RI avec chacune des solutions de paramétrisation proposées dans la littérature, ainsi qu'avec la valeur K^*, trouvée dans les expérimentations précédentes.

3.4.1 Protocole Expérimental

La deuxième série de tests que nous présentons dans ce chapitre a donc eu pour but d'évaluer l'algorithme Forest-RK, en le comparant à Forest-RI avec chacune des

Algorithme 6 ForestRK

Entrée : T l'ensemble d'apprentissage
Entrée : L le nombre d'arbres dans la forêt
Sortie : $foret$ l'ensemble des arbres qui composent la forêt construite

1: **pour** l de 1 à L **faire**
2: $arbre \leftarrow$ un arbre vide, *i.e.* composé de sa racine uniquement
3: $T_l \leftarrow$ un ensemble bootstrap, dont les données sont tirées aléatoirement (avec remise) de T
4: $gains \leftarrow$ un tableau de M éléments
5: **pour tout** i de 1 à M **faire**
6: $gains[i] \leftarrow$ gain d'information de la caractéristique A_i en fonction de T_l
7: $arbre.racine \leftarrow RndTree2(arbre.racine, T_l, gains)$
8: $foret \leftarrow foret \bigcup arbre$
9: **retour** $foret$

Algorithme 7 RndTree2

Entrée : n : Le nœud courant
Entrée : T : L'ensemble des données associées au nœud n
Entrée : $gains[M]$: un tableau de M éléments, pour contenir les valeurs de gains d'information pour chacune des M caractéristiques
Sortie : n : Le même nœud, modifié par la procédure

1: $sommeGains \leftarrow 0$
2: **pour** i de 1 à M **faire**
3: **si** $gains[i] < 0.1$ **alors**
4: $sommeGains \leftarrow sommeGains + 1$
5: **si** n n'est pas une feuille **alors**
6: **si** $sommeGains > \alpha * M$ **alors**
7: $K \leftarrow$ Sélectionner un entier entre 1 et M aléatoirement selon une loi uniforme
8: **sinon**
9: $K \leftarrow$ Sélectionner un entier aléatoirement selon une loi normale de moyenne $\sqrt{M}$ et d'écart-type $\frac{M}{50}$
10: $C \leftarrow K$ caractéristiques choisies aléatoirement ;
11: **pour tout** $A_i \in C$ **faire**
12: Choix de la règle de partitionnement utilisant A_i, de sorte qu'elle optimise le gain d'information par rapport à T ;
13: $partition \leftarrow$ partition qui optimise le gain d'information ;
14: $n.ajouterFils(partition)$;
15: **pour tout** $nFils \in n.noeudFils$ **faire**
16: $RndTree2(nFils, nFils.donnes)$;
17: **retour** n ;

solutions de paramétrisation proposées, à savoir K^*, $K = \sqrt{M}$, $K = \lfloor \log_2(M) + 1 \rfloor$ et $K = 1$. Nous avons complété les expérimentations de la section précédente

en reprenant le même protocole expérimental avec l'algorithme Forest-RK. Pour l'ensemble des bases de données présentées dans le tableau 3.1, nous avons dans un premier temps lancé Forest-RK pour chaque T_i et relevé les taux d'erreur en test. Puis, nous avons appliqué le test de McNemar pour chaque couple Forest-RK/Forest-RI avec chacune des valeurs de K mentionnées ci-dessus dans un second temps. L'algorithme 8 décrit cette deuxième série de tests.

Algorithme 8 Protocole Expérimental 2

Entrée : N le nombre de données disponibles.
Entrée : M le nombre de caractéristiques disponibles.
Sortie : $\epsilon_{RK}[50]$ un tableau 1D pour conserver les taux d'erreur obtenus avec Forest-RK.
Sortie : $\mathcal{M}[50][2]$ un tableau 2D pour conserver les réponses au test de McNemar.

1: **pour** $i \in [1..50]$ **faire**
2: Tirer aléatoirement sans remise $\frac{2}{3}$ des données disponibles pour former l'ensemble d'apprentissage T_{r_i}. Les données restantes forment alors l'ensemble de test Ts_i, le couple (T_{r_i}, T_{s_i}) est noté T_i.
3: $h_{RK} \leftarrow$ Induire une RF avec l'algorithme Forest-RK sur T_{r_i}.
4: $\epsilon_{RK}(i) \leftarrow$ Tester la RF résultante sur T_{s_i}.
5: $H_{RI}(K^*) \leftarrow$ Induire une RF avec l'algorithme Forest-RI/K^* sur T_{r_i}.
6: $H_{RI}(K_{\sqrt{M}}) \leftarrow$ Induire une RF avec l'algorithme Forest-RI/$K_{\sqrt{M}}$ sur T_{r_i}.
7: $H_{RI}(K_1) \leftarrow$ Induire une RF avec l'algorithme Forest-RI/K_1 sur T_{r_i}.
8: $H_{RI}(K_{\log_2}) \leftarrow$ Induire une RF avec l'algorithme Forest-RI/$K_{\log_2}$ sur T_{r_i}.
9: **pour** $h_n \in H_{RI}$ **faire**
10: $\mathcal{M}(i)(h_n) \leftarrow$ Lancer le test de McNemar avec les classifieurs (h_{RK}, h_n) sur T_{s_i}.

Les résultats que nous avons obtenus avec ces expérimentations pour chaque base de données se présentent sous la même forme de deux tableaux, l'un contenant les 50 taux d'erreur obtenus avec Forest-RK pour chaque T_i, et l'autre contenant les résultats aux tests de McNemar pour les 4 comparaisons Forest-RK/Forest-RI. Nous détaillons et discutons ces résultats dans la section suivante.

3.4.2 Résultats et Discussion

Les résultats obtenus avec cette deuxième expérimentation ont été intégrés aux deux tableaux 3.4 et 3.5, présentés dans la section 3.2.2. Nous les reportons également dans deux nouveaux tableaux 3.7 et 3.8 qui ne contiennent cette fois que les résultats de ces expérimentations. La dernière colonne du tableau 3.7 présente les taux d'erreur moyens et les écart-types sur les 50 T_i, obtenus avec Forest-RK. Le tableau 3.8 synthétisent les résultats des comparaisons entre Forest-RK et Forest-RI, obtenues avec le test de McNemar.

Dataset	K^*	$K_{\sqrt{M}}$	Forest-RK
Diabetes	23.51 ± 1.76 (2)	23.74 ± 1.96 (3)	23.71 ± 2.34
Digits	2.24 ± 0.12 (10)	2.25 ± 0.13 (18)	2.24 ± 0.13
DigReject	7.12 ± 0.34 (150)	7.58 ± 0.34 (18)	7.16 ± 0.33
Gamma	12.17 ± 0.33 (3)	12.17 ± 0.33 (3)	12.30 ± 0.38
Isolet	5.72 ± 0.53 (40)	5.85 ± 0.44 (25)	6.87 ± 0.45
Letter	4.16 ± 0.28 (3)	4.23 ± 0.23 (4)	4.28 ± 0.25
Madelon	17.73 ± 1.60 (261)	30.50 ± 1.94 (22)	18.34 ± 1.52
Mfeat-factors	3.56 ± 0.71 (10)	3.57 ± 0.58 (15)	3.48 ± 0.61
Mfeat-fourier	16.81 ± 1 (19)	17.11 ± 1.05 (9)	17.00 ± 0.98
Mfeat-karhunen	4.30 ± 0.68 (6)	4.33 ± 0.69 (7)	4.45 ± 0.69
Mfeat-zernike	22.26 ± 1.06 (7)	22.26 ± 1.06 (7)	22.27 ± 1.14
MNIST	5.03 ± 0.14 (17)	5.19 ± 0.16 (10)	5.15 ± 0.14
Musk	2.34 ± 0.30 (88)	2.40 ± 0.29 (13)	2.34 ± 0.31
OptDigits	1.97 ± 2.28 (11)	2.03 ± 0.34 (8)	1.95 ± 0.29
Page-blocks	2.60 ± 0.28 (3)	2.60 ± 0.28 (3)	2.62 ± 0.33
Pendigits	1.00 ± 0.17 (5)	1.05 ± 0.16 (4)	1.01 ± 0.16
Segment	2.29 ± 0.53 (7)	2.30 ± 0.44 (4)	2.34 ± 0.44
Spambase	$4,83 \pm 0.53$ (6)	$4,98 \pm 0.49$ (8)	5.11 ± 0.59
Vehicle	24.75 ± 2.02 (5)	25.15 ± 2.14 (4)	24.80 ± 2.09
Waveform	14.92 ± 0.81 (4)	14.97 ± 0.78 (6)	14.90 ± 0.84

TAB. 3.7: Taux d'erreur moyens obtenus avec Forest-RI/K^*, Forest-RI/$K_{\sqrt{M}}$ et Forest-RK

La première observation que l'on peut faire à partir du tableau 3.7 est que Forest-RK permet d'obtenir pour l'ensemble des 20 bases des taux d'erreur au moins très proches de Forest-RI/$K_{\sqrt{M}}$, mais aussi et surtout de Forest-RI/K^*. C'est particulièrement le cas pour les deux bases de données DigReject et Madelon pour lesquelles $\sqrt{M}$ est sous-optimale, alors que Forest-RK en revanche est proche de l'optimalité. Pour finir de s'en convaincre nous pouvons ensuite regarder plus attentivement le deuxième tableau qui synthétise les résultats du test de McNemar. On constate alors que Forest-RK dans la totalité des cas permet d'obtenir des performances statistiquement équivalentes à Forest-RI/K^*.

Lorsque l'on compare Forest-RK à Forest-RI/$K_{\sqrt{M}}$ maintenant, on constate que pour toutes les bases pour lesquelles nous avons vu que $\sqrt{M}$ était une valeur raisonnable pour induire des forêts performantes, Forest-RK est logiquement statistiquement aussi performant. "Logiquement" car le processus d'induction de Forest-RK favorise fortement le choix de valeurs de K proche de $\sqrt{M}$ pour chaque nœud de l'arbre. C'est d'autant plus le cas que nous avons choisi une valeur d'écart-type pour la loi gaussienne très faible. Pour les deux bases de données pour lesquelles $\sqrt{M}$ était sous-optimale, Forest-RK se montre proche en termes de performances de Forest-RI/K^*, et par conséquent statisquement bien plus performant que Forest-RI/$K_{\sqrt{M}}$.

Algo A	Forest-RK	
Algo B	K^*	$K_{\sqrt{M}}$
Diabetes	0/**50**/0	2/**48**/0
Digits	3/**46**/1	1/**46**/3
DigReject	1/**45**/4	0/24/**26**
Gamma	3/**46**/1	1/**49**/0
Isolet	3/**46**/1	1/**49**/0
Letter	5/**45**/0	0/**50**/0
Madelon	5/**45**/0	0/0/**50**
Mfeat-fac	0/**49**/1	1/**49**/0
Mfeat-fou	1/**49**/0	1/**49**/0
Mfeat-kar	2/**48**/0	1/**48**/0
Mfeat-zer	1/**49**/0	0/**50**/0
MNIST	10/**40**/0	4/**44**/2
Musk	0/**50**/0	0/**48**/0
OptDigits	2/**48**/0	1/**49**/0
Page-blocks	0/**50**/0	1/**49**/0
Pendigits	1/**47**/2	2/**47**/1
Segment	2/**48**/0	0/**50**/0
Spambase	9/**41**/0	2/**48**/0
Vehicle	0/**49**/1	0/**49**/1
Waveform	1/**49**/0	2/**48**/0

TAB. 3.8: Résultats des tests de McNemar pour Forest-RK

En conclusion de ces résultats, nous pouvons dire que Forest-RK est une solution bien plus intéressante que les solutions de paramétrisation de Forest-RI qui sont proposées dans la littérature. Certes, il ne permet pas d'obtenir dans l'absolu de meilleures performances que Forest-RI, puisqu'il n'est dans la totalité des cas "que" statistiquement aussi performant que Forest-RI/K^*. Mais dans la mesure où nous ne disposons à ce jour d'aucun moyen, d'aucune règle, permettant de trouver *a priori* la valeur de ce K^*, Forest-RK est la meilleure solution qui est proposée pour exploiter au mieux le potentiel des forêts aléatoires.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une première étude du fonctionnement des forêts aléatoires, en nous intéressant plus particulièrement au paramètre K de l'algorithme Forest-RI. Ce paramètre fixe le nombre de caractéristiques à sélectionner aléatoirement à chaque nœud des arbres de décision afin d'en déterminer la règle de partitionnement. À ce jour, très peu de travaux de recherche se sont intéressés à l'influence de ce paramètre sur les performances, les autres se contentant pour la

plupart de le fixer à une des valeurs par défaut les plus répandues dans la littérature. Or nous avons vu que ces valeurs sont toutes arbitraires et ne sont en tout cas justifiées dans aucune étude théorique ou expérimentale. Dans ce chapitre, nous répondons à ce problème du choix de la valeur de K en conduisant une analyse de ses différentes valeurs possibles et en apportant des éléments de compréhension sur leur influence sur les performances en généralisation des forêts.

Nous avons tout d'abord montré que cette influence est réelle, mais surtout qu'elle peut être relativement importante dans certain cas. Certaines valeurs de K conduisent parfois à une chute importante des résultats. Ensuite, nous avons montré que parmi les valeurs de la littérature, l'une d'elles se démarque des autres par de meilleurs résultats sur une grande majorité des bases de données que nous avons testées ; il s'agit de la valeur $K = \sqrt{M}$. En outre, nous avons mis en évidence que cette valeur est proche de l'optimalité pour une majorité des bases que nous avons testées. Cependant, les résultats que nous avons obtenus avec cette première étude mettent également en exergue que cette valeur, tout comme les autres valeurs proposées dans la littérature, ne permet pas toujours d'obtenir de bons résultats. Pour certaines bases de données, les performances obtenues sont significativement en dessous des meilleures performances qu'il est possible d'obtenir avec d'autres valeurs de K. Si cette valeur $K = \sqrt{M}$ semble la plupart du temps être une bonne solution de paramétrisation, ce problème ne sera pas résolu tant qu'on ne pourra pas identifier ces cas pour lesquels elle fonctionne beaucoup moins bien.

C'est l'objet de la deuxième contribution de ce chapitre : expliquer les raisons pour lesquelles ces comportements atypiques des forêts vis à vis des valeurs de K apparaissent dans certains cas plutôt que d'autres. Pour cela, nous avons suivi une intuition répandue dans la littérature qui consiste à dire que la "qualité" des caractéristiques influence ces comportements. Nous avons donc étudié le caractère discriminant des caractéristiques et l'avons confronté aux différents comportements observés des forêts. Nous avons prouvé avec ces travaux que cette intuition est vérifiée. Les problèmes pour lesquels l'évolution des performances en fonction des valeurs de K est atypique, et pour lesquels les valeurs de la littérature sont sous-optimales, sont ceux qui présentent une grande proportion de caractéristiques très peu discriminantes.

Cependant, avoir réussi à cerner ce problème ne permet pas de le résoudre. Notre troisième contribution dans ce chapitre, porte sur un algorithme original qui s'appuie sur ces conclusions pour proposer un alternative plus intéressante aux solutions de paramétrisation de la littérature. Son principe est de choisir aléatoirement la valeur de K à chaque nœud de l'arbre, mais de guider le tirage aléatoire de cette valeur par la "qualité" de l'espace des caractéristiques. En comparant ensuite cet algorithme à la meilleure des valeurs de K que nous avons obtenue dans les expérimentations précédentes, nous montrons que Forest-RK est, pour la totalité des bases testées, au moins aussi performant que Forest-RI. Il permet de plus de palier au problème que pose les comportements atypiques observés, et constitue en cela une très bonne alternative aux paramétrisations par défaut de Forest-RI.

Les perspectives de cette étude sont nombreuses. Dans un premier temps, il serait intéressant d'étudier l'influence du procédé Random Feature Selection sur les propriétés de *force* et de *corrélation* que nous présentons en section 3.5. Breiman amorce dans [Breiman 2001] une étude de ce genre en observant l'évolution de chacune de ces deux propriétés en fonction de la valeur de K. Il ne réalise cette expérience que sur une seule base, et ne dresse finalement que des conjectures. Nous pourrions, en mettant en place ce même type d'expériences sur l'ensemble des 20 bases de données que nous avons utilisé dans ce chapitre, infirmer ou confirmer les intuitions de Breiman, à savoir que l'augmentation de la valeur de K tend à augmenter la *corrélation* dans les forêts, alors qu'elle n'influe pas autant sur la *force*. Et il serait par ailleurs également intéressant d'analyser comment la procédure Forest-RK modifie ce compromis ces évolutions.

Ensuite, nous trouvons qu'il serait également intéressant de mettre en place une variante de Forest-RK qui, à la place d'un tirage aléatoire de la valeur de K effectué à chaque nœud de l'arbre, effectuerait par exemple ce tirage aléatoire pour chaque arbre. Cette valeur serait alors fixée pour toute l'induction d'un arbre, mais différentes d'un arbre à un autre. Un intérêt de cela serait d'approfondir l'étude de l'effet de l'aléatoire sur l'erreur des forêts. On pourrait notamment percevoir dans quelle mesure le degré d'intervention de l'aléatoire dans le choix de cette valeur agit sur le comportement des forêts.

Chapitre 4

Force et *Corrélation* dans les Forêts Aléatoires

Sommaire

4.1 Introduction

Comme nous l'avons évoqué en conclusion du chapitre 2, un processus d'induction statique de forêts aléatoires tel que Forest-RI ou Forest-RK présente de notre point de vue deux principaux inconvénients : (i) le nombre d'arbres à induire dans une forêt doit être fixé *a priori* (ii) les arbres sont indépendamment, et donc arbitrairement, ajoutés à l'ensemble. Avec l'introduction de l'aléatoire dans le processus d'induction de forêts, on espère tirer avantage des complémentarités des arbres. Cela implique cependant de n'avoir aucune garantie que les arbres qui sont ajoutés les uns après les autres vont bien coopérer au sein du comité ; pas plus que cela ne garantit qu'en ajouter d'autres à l'ensemble va permettre d'en améliorer les performances. On peut même imaginer que certains arbres d'une forêt détériorent dans une certaine mesure les performances de l'ensemble. C'est pourquoi nous nous intéressons dans cette section à l'influence de chaque arbre sur le fonctionnement de l'ensemble et sur ses performances.

Un des résultats théoriques importants sur les forêts aléatoires concerne le nombre d'arbres de décision à induire dans une forêt. Nous l'évoquons dans le chapitre 2 de ce manuscrit, au début de la section 2.5 ; il s'agit de la démonstration de la convergence des forêts par Breiman dans [Breiman 2001]. Ce résultat énonce en fait que le taux d'erreur en généralisation d'une forêt aléatoire, noté PE^*, converge nécessairement vers une valeur limite, quand le nombre d'arbres qui la composent augmente. Une interprétation de ce résultat est qu'il n'est pas nécessaire, quand on construit une forêt aléatoire, d'ajouter des milliers et des milliers d'arbres de décision pour obtenir les meilleures performances possibles. Au delà d'une certaine quantité d'arbres aléatoires, aucun gain de performance significatif ne sera réalisé en en ajoutant d'autres.

C'est une assertion qu'il est facile de confirmer expérimentalement. Il suffit pour cela de mesurer les taux d'erreur en généralisation de la forêt tout au long de sa construction. Par exemple, la figure 4.1 présente les résultats que nous avons obtenus avec cette simple expérience sous forme de courbes des taux d'erreur moyens en fonction du nombre d'arbres ajoutés à la forêt au cours de l'induction. Comme au cours des expérimentations du chapitre 3, ces valeurs ont été obtenues en moyennant les taux d'erreur observés à chaque itération sur 50 découpages aléatoires de chaque base de données. L'expérience a par ailleurs été menée avec les deux algorithmes Forest-RI (courbe continue) et Forest-RK (courbe pointillée). On constate alors que les résultats expérimentaux confirment les résultats théoriques, à savoir la convergence des taux d'erreur pour un nombre croissant d'arbres de décision ajoutés à la forêt. C'est d'ailleurs le cas pour les deux algorithmes.

Cette constatation est une indication importante pour quiconque souhaite mettre en œuvre ces méthodes. Cependant, ce résultat soulève une question évidente : quel est ce nombre d'arbres limite au delà duquel il n'est plus "utile" d'en ajouter d'autres ? Lorsque l'on regarde la figure 4.1, on constate que le nombre

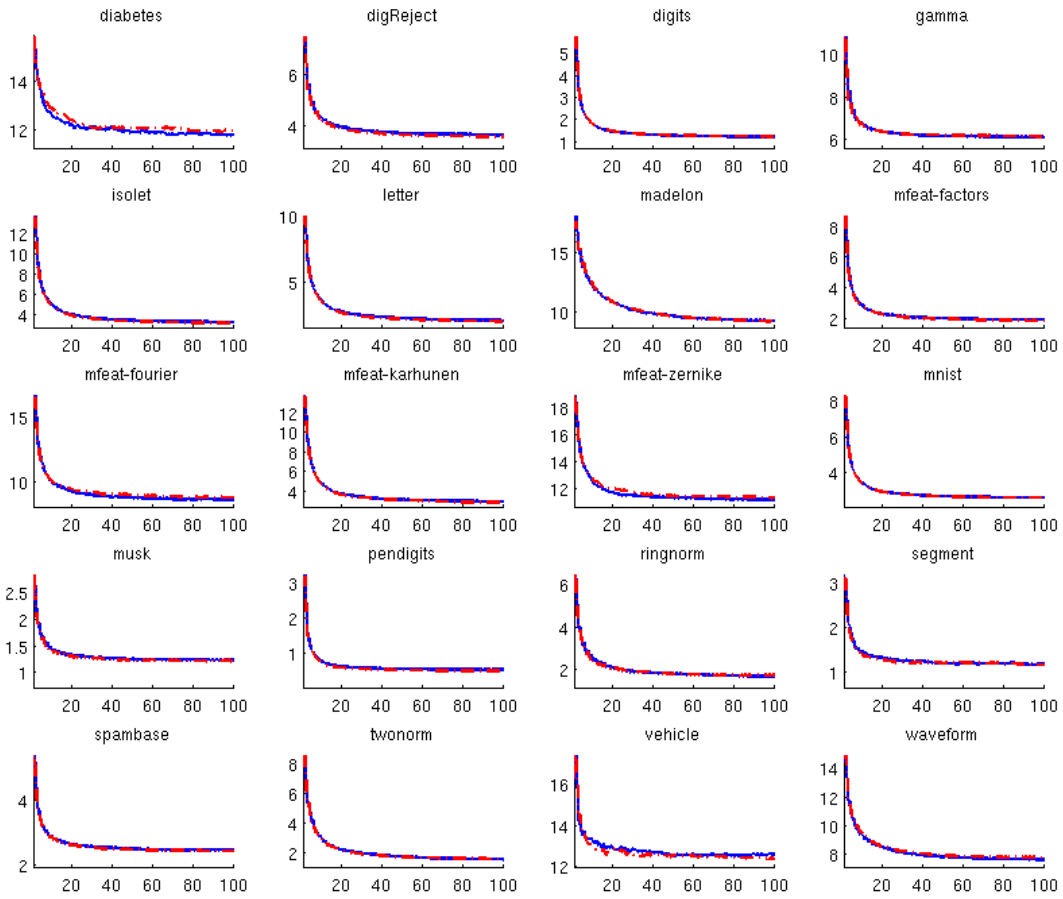

FIG. 4.1: Évolution des taux d'erreur en fonction du nombre d'arbres L, pour les algorithmes Forest-RI (courbes continues) et Forest-RK (courbes pointillées).

d'arbres à partir duquel la courbe semble approcher son asymptote n'est pas identique pour toutes les bases de données. Par exemple, il est égal à environ 80 arbres pour la base Madelon alors qu'il se situe plutôt autour de 20 arbres pour Pendigits. Si ce nombre n'est pas identique pour tous les problèmes d'apprentissage automatique, cette question renvoie alors à la problématique du critère d'arrêt pour l'induction de forêts aléatoires.

À ce propos, Breiman ne donne pas d'éléments expérimentaux, mais suggère que l'utilisation des estimations out-of-bag pourrait être efficace pour la mise au point d'un tel critère d'arrêt ([Breiman 2001]). Son idée est que ces estimations peuvent permettre de déterminer quand les performances en généralisation de la forêt convergent au fur et à mesure que les arbres sont ajoutés. Dès que l'erreur out-of-bag semble converger, on peut stopper l'induction de la forêt en évitant ainsi de fixer *a priori* le nombre d'arbres qu'elle doit contenir. Dans leur étude comparative des différentes techniques d'induction d'ensembles de classifieurs, dont nous avons

déjà parlé dans le chapitre 2 (*cf.* section 2.7), Banfield *et al.* discutent cette idée un peu plus en détail ([Banfield 2006]). Le premier point qu'ils abordent concerne la sur-estimation du nombre d'arbres obtenus avec ce type de critère d'arrêt. Puisque chaque donnée d'apprentissage n'est évaluée que par un sous-ensemble des arbres de la forêt pour le calcul de la mesure out-of-bag (*cf.* section 1.4.1.2), l'utilisation de ce critère d'arrêt mène obligatoirement vers un plus grand nombre d'arbres de décision qu'il n'est réellement nécessaire d'en ajouter pour atteindre le point de convergence. Un deuxième problème qu'ils mentionnent concerne les difficultés d'estimation lorsque le nombre de données de la base est relativement faible — les auteurs précisent pour un nombre de données inférieur à 1000. Ce type de base présente en effet une grande instabilité dans les performances en généralisation, aussi bien sur un ensemble de test que sur les ensembles out-of-bag. La diminution de ces taux d'erreur n'est pas monotone au fur et à mesure que les arbres sont ajoutés, comme c'est normalement le cas pour les autres bases contenant un plus grand nombre de données. C'est un phénomène que nous avons également observé tout au long de nos expérimentations avec les forêts aléatoires.

Pour remédier à ce problème, Banfield *et al.* mettent au point un algorithme d'induction de forêts utilisant ce type de critère d'arrêt, mais qui prétraite la courbe des taux d'erreur out-of-bag en la lissant, puis en anticipant la convergence des taux pour éviter la sur-estimation du nombre d'arbres ([Banfield 2006]). Ils comparent ensuite les forêts qu'ils obtiennent avec cette procédure, dans un premier temps à des forêts dont ils fixent le nombre d'arbres à 2000, puis dans un second temps à des forêts obtenues avec un critère d'arrêt utilisant l'erreur out-of-bag sans prétraitement. Ils confirment ainsi expérimentalement les problèmes mentionnés ci-dessus, et mettent également en évidence que la mise en place d'un tel critère d'arrêt conduit très rarement à une perte de performances en comparaison aux forêts de 2000 arbres.

D'autres travaux se sont penchés sur cette problématique du nombre d'arbres qu'il est nécessaire d'induire dans une forêt aléatoire pour obtenir des performances raisonnablement bonnes ; il s'agit des travaux de Latinne *et al.* dans [Latinne 2001]. Dans cet article, les auteurs présentent une procédure qui détermine le nombre minimum d'arbres à construire pour atteindre des performances raisonnablement bonnes, et qui utilise pour cela le test statistique de McNemar — que nous présentons dans la section 3.2.2 du chapitre 3. L'idée est d'utiliser le test de McNemar pour déterminer si l'ajout d'arbres améliore significativement ou non les performances. Dans leurs expérimentations cependant, Latinne *et al.* n'utilisent pas ce test en tant que critère d'arrêt à proprement parler mais constituent des matrices avec les réponses de McNemar pour chaque paire de forêts aléatoires de 1 à 200 arbres. Quelques-uns des résultats qu'ils obtiennent sont montrés en figure 4.2 sous la forme de régions noires ou blanches selon que le test de McNemar a détecté des différences significatives de performances ou non entre deux forêts de tailles différentes. Les auteurs réitèrent l'expérience pour trois algorithmes d'induction de forêts aléatoires différents : (i) le Bagging (ii) les Random Subspaces (appelé MFS sur la figure) (ii) et Forest-RI (Bagrf sur la figure).

Les auteurs mettent ainsi en évidence qu'il est possible d'obtenir des performances similaires pour un nombre d'arbres relativement faible. Ces résultats illustrent ni plus ni moins ce que nous venons de mettre en évidence avec la figure 4.1, à savoir la convergence des performances en généralisation des forêts aléatoires pour un nombre d'arbres généralement inférieur à 100. Ils mettent également en évidence qu'il est possible de s'appuyer sur cette convergence pour stopper l'induction d'une forêt aléatoire sans avoir à fixer *a priori* le nombre d'arbres.

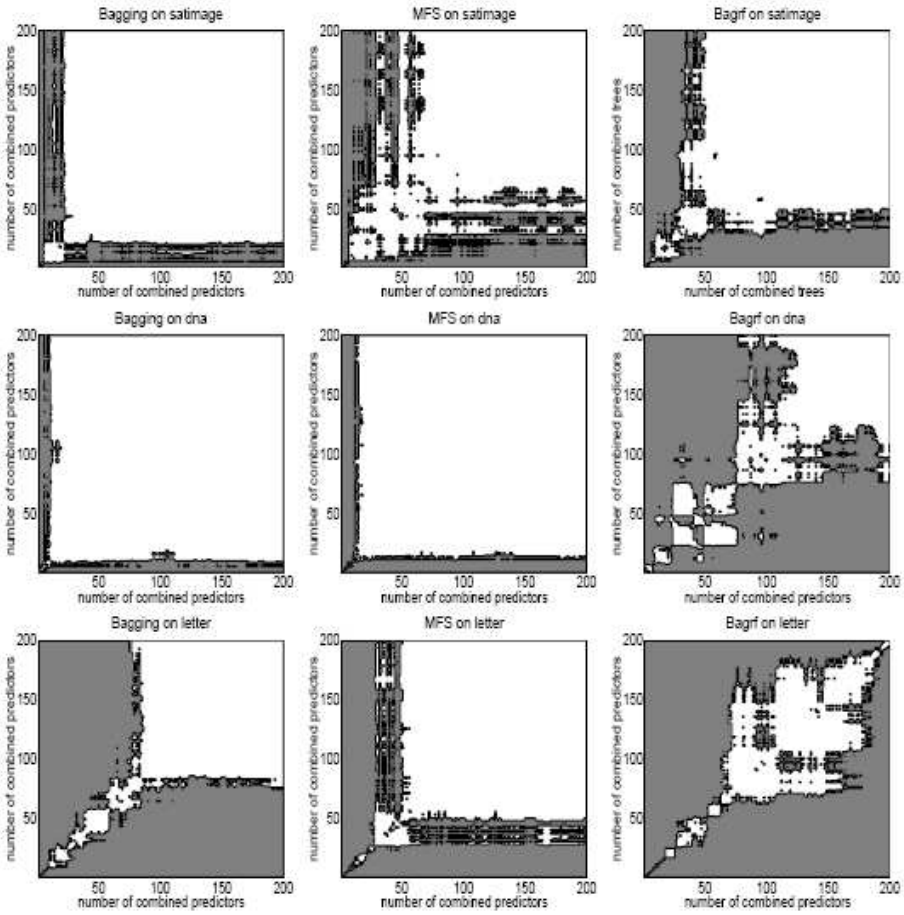

FIG. 4.2: Résultats des expérimentations de Latinne *et al.* dans [Latinne 2001]. Chaque image représente une matrice de résultats de McNemar en fonction du nombre d'arbres dans les forêts comparées. Les espaces blancs regroupent les paires de forêts pour lesquelles aucune différence significative de performances n'a été obtenue d'après le test de McNemar

Le constat avec tous ces travaux est donc qu'en moyenne les performances d'une forêt augmentent jusqu'à converger, au fur et à mesure qu'on ajoute des arbres aléatoires à l'ensemble. Cela nous indique qu'à partir d'une certaine quantité d'arbres les erreurs de quelques-uns d'entre eux sont en grande partie compensées par les bonnes predictions des autres. Cependant, noter qu'au delà d'une certaine quantité

d'arbres, aucun gain de performances ne peut être obtenu en ajoutant de nouveaux arbres, toujours de façon arbitraire, ne nous permet pas d'affirmer qu'on a alors tiré le meilleur parti de l'ensemble. Il n'en reste pas moins que parmi ces arbres, certains détériorent très certainement les performances en introduisant plus d'erreurs en classification qu'ils n'apportent de bonnes prédictions. En d'autres termes, l'intuition suivie dans ce chapitre est qu'il devrait être possible d'obtenir de meilleures forêts aléatoires en utilisant un sous-ensemble d'arbres bien choisis au lieu de l'ensemble des arbres arbitrairement induits. Toute la problématique tourne alors autour du choix de ces arbres. Notre but ici est donc de trouver des mécanismes qui nous permettraient de les identifier et de mieux comprendre pourquoi un sous-ensemble d'arbres en particulier coopère mieux qu'un autre.

Dans la suite de ce chapitre, nous décrivons l'étude que nous avons menée pour répondre à ces questions. Tout d'abord, nous présentons notre approche pour générer un large panel de sous-forêts présentant des performances et des propriétés différentes les unes des autres. Nous avons pour cela utilisé des méthodes de sélection statique de classifieurs. Nous mettons alors en évidence avec nos résultats que dans la plupart des cas, il est possible d'obtenir un gain significatif de performances en éliminant de l'ensemble de départ une quantité importante d'arbres de décision, et qu'un sous-ensemble d'arbres bien choisi est meilleur qu'un ensemble d'arbres arbitraire. Nous étudions ensuite plus en détails les comportements de ces sous-forêts en suivant une piste que Breiman dresse dans [Breiman 2001], en définissant les propriétés de *force*, notée s, et de *corrélation*, notée $\overline{\rho}$. Nous montrons que ces deux propriétés permettent d'expliquer les variations de performances d'une sous-forêt à une autre, via le rapport $\frac{\overline{\rho}}{s^2}$ que nous présentons dans la section 2.5.

En illustrant le fait qu'un sous-ensemble d'arbres bien choisi permet d'obtenir un gain significatif de performances, nous mettons en évidence l'intérêt de la conception d'une procédure d'induction des forêts séquentielle et guidée. Le but final de cette étude est donc *in fine* d'apporter tous les éléments nécessaires à la conception d'un processus d'induction dynamique. Un de ces éléments concerne par ailleurs le critère utilisé pour guider l'induction dynamique. En mettant en évidence le lien entre les performances et les propriétés de *force* et de *corrélation* nous fournissons une piste intéressante pour la mise en place de ce critère.

4.2 Sélection de classifieurs

L'idée générale de cette étude est de trouver des propriétés qui peuvent expliquer les variations de performances d'une forêt à une autre, et surtout qui peuvent *a priori* influer sur ces variations. Pour cela, il nous faut étudier ces propriétés sur un large panel de forêts aléatoires, présentant des performances significativement différentes les unes des autres. Nous proposons ainsi d'utiliser des méthodes de sélection statique de classifieurs, non pas pour trouver des sous-ensembles d'arbres optimaux, mais pour générer plusieurs sous-forêts à partir d'un large ensemble d'arbres aléatoires et d'en étudier les propriétés.

Lorsqu'il s'agit de mettre en œuvre un processus de sélection de classifieurs, deux choix doivent être faits :

- Le choix d'un critère de sélection.
- Le choix d'une méthode d'exploration de l'espace des solutions.

En ce qui concerne le critère de sélection, deux principales approches sont proposées dans la littérature ([Kohavi 1997]) :

- **L'approche "filter"**, qui consiste à sélectionner un sous-ensemble de classifieurs à l'aide d'un critère d'évaluation *a priori* qui ne prend pas en compte les performances de l'ensemble. Ce type de critère est très souvent basé sur la diversité des classifieurs au sein du sous-ensemble sélectionné. Aksela dans [Aksela 2003] ou encore Ruta et Gabrys dans [Ruta 2004] en donnent un large panel d'exemples.
- **L'approche "wrapper"**, qui réalise cette fois une sélection de sous-ensembles de classifieurs en optimisant *a posteriori* les performances de l'ensemble. Dans ce type d'approches, il est souvent nécessaire d'utiliser un ensemble de données de validation, indépendant de l'ensemble d'apprentissage et de celui de test. Les performances au cours de la sélection sont alors évaluées sur cet ensemble de validation et le ou les sous-ensembles de classifieurs qui optimisent ces performances sont celui ou ceux qui sont retenus.

Comme nous cherchons à mettre en évidence qu'il est possible d'obtenir un gain de performance significatif en évinçant d'une forêt les arbres qui contribuent le plus à l'erreur en généralisation, l'approche *wrapper* a été retenue pour ces expérimentations. La sélection de classifieurs a donc été réalisée en tentant d'optimiser les performances des sous-ensembles d'arbres résultants.

Concernant les méthodes de sélection de classifieurs maintenant, il existe plusieurs taxonomies. Nous pouvons en citer deux ; Zouari dans [Zouari 2004] tout d'abord, regroupe ces méthodes en trois types d'approches qui sont résumés dans le tableau 4.1. Ruta *et al.* quant à eux en proposent une catégorisation différente, en trois familles de méthodes : (i) Les méthodes heuristiques (*Single Best*, *N best*, etc.) (ii) les méthodes de recherche gloutonne (*Forward Search*, *Backward Search*, *Exhaustive Search* etc.) (iii) les algorithmes évolutionnaires (*Algorithmes Génétiques*, etc.).

La plupart de ces méthodes étaient à l'origine des méthodes destinées à la sélection de caractéristiques, une problématique très proche de celle de la sélection de classifieurs. Dash *et al.* présentent dans [Dash 1997] un schéma récapitulatif assez complet des méthodes de sélection de caractéristiques qui sont pour la plupart parfaitement adaptables à la sélection de classifieurs (*cf.* figure 4.3).

Beaucoup de méthodes mentionnées ci-dessus sont des algorithmes de recherche qui procèdent par parcours de l'espace des sous-ensembles de classifieurs possibles.

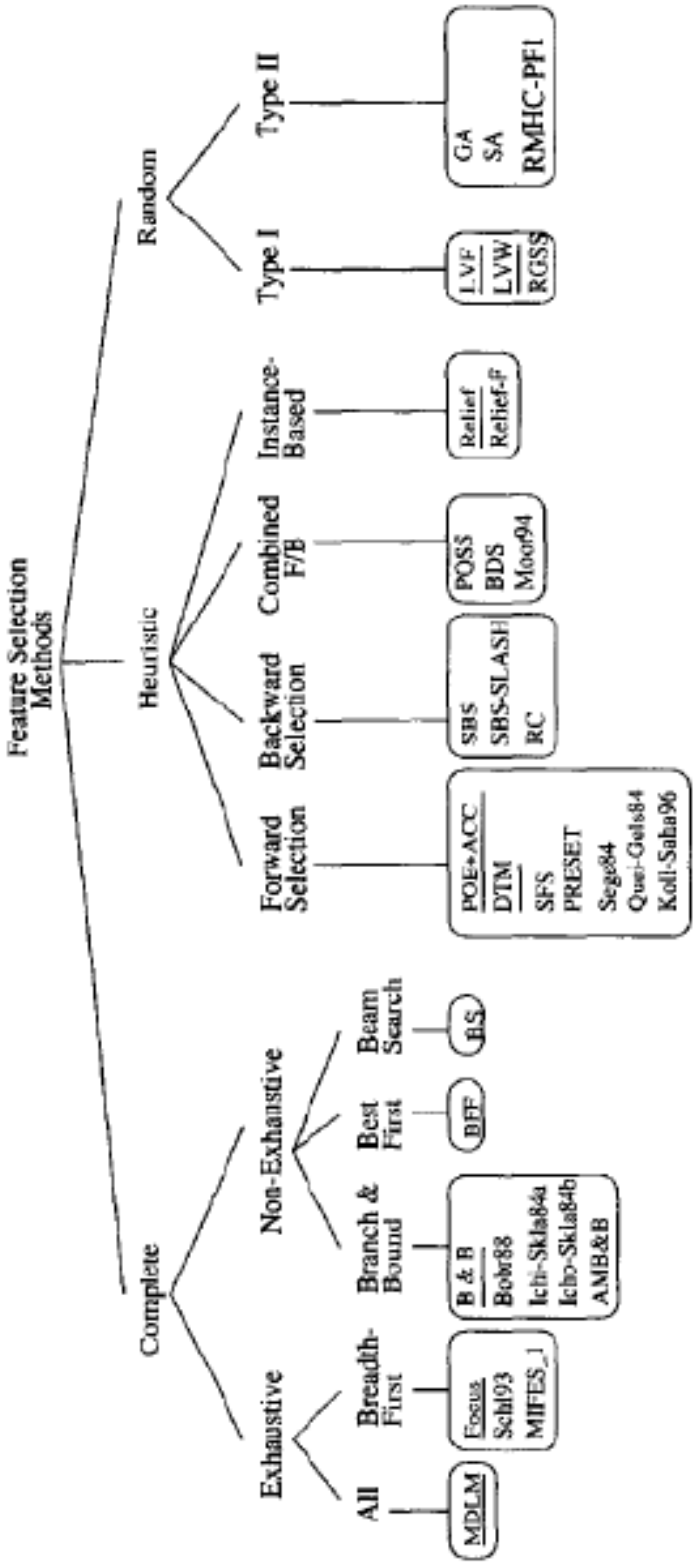

FIG. 4.3: Schéma récapitulatif des méthodes de sélection de caractéristiques [Dash 1997]

Paradigme de Sélection	Description	Références
Surproduire et Choisir	Utilise les méthodes de création d'ensembles pour surproduire et les performances des classifieurs et ou la diversité pour choisir	[Sharkey 2000, Roli 2001, Aksela 2003]
Regrouper et Extraire	Utilise un algorithme de clustering pour regrouper les classifieurs initiaux et n'utilise ensuite qu'un prototype des groupes ainsi formés	[Giacinto 2000, Kuncheva 2000]
Rechercher et Sélectionner	Utilise un algorithme de recherche pour explorer les solutions possibles et sélectionne un optimum en se basant sur un certain critère	[Partridge 1996, Hao 2003, Banfield 2003]

TAB. 4.1: Paradigmes de sélection statique de classifieurs [Zouari 2004]

Le principal inconvénient de ce type de méthodes est que la plupart sont sous-optimales, car pour éviter un parcours exhaustif de l'espace des solutions, qui serait dans une grande majorité des cas beaucoup trop coûteux, elles procèdent par recherches heuristiques. Cependant, elles ont pour certaines l'avantage d'être faciles à mettre en œuvre et rapides à l'exécution. Nous avons décidé dans un premier temps de mettre en œuvre l'une de ces méthodes qui fonctionnent par recherches heuristiques, puisque nous rappelons que notre but n'est pas ici de chercher le sous-ensemble d'arbres optimal, mais bien de générer un grand ensemble de sous-forêts.

4.3 Sélection de sous-forêts aléatoires

Comme nous l'avons déjà mentionné, l'optimalité des méthodes de sélection n'est pas une priorité ici. C'est la raison pour laquelle les deux algorithmes de sélection de classifieurs **Sequential Forward Selection (SFS)** et **Sequential Backward Selection (SBS)** ont été choisis pour ces premières expérimentations. Ce sont deux méthodes très intuitives, qui construisent de façon séquentielle le sous-ensemble final en sélectionnant à chaque itération le classifieur à ajouter ou à retirer ([Hao 2003]).

À la première itération du processus SFS, on sélectionne le classifieur de l'ensemble de départ qui est le plus performant individuellement. Ensuite, on teste toutes les paires de classifieurs possibles que l'on peut composer avec ce premier classifieur. Si la forêt est initialement composée de L arbres, il y a aura $L-1$ paires testées. À la fin de cette deuxième itération on retient la paire qui a minimisé le taux d'erreur. On réitère ensuite l'opération avec tous les triplets possibles contenant cette paire de classifieurs obtenue à la deuxième itération. Le processus se poursuit jusqu'à obtenir

un sous-ensemble de la taille souhaitée. SBS quant à lui est le pendant de SFS, et procède à l'inverse en partant de l'ensemble de classifieurs complet et en retirant à chaque itération le classifieur qui contribue le moins aux performances. Le schéma de la figure 4.4 fournit une illustration de ces deux processus appliqués à une forêt aléatoire composée de L arbres de décision.

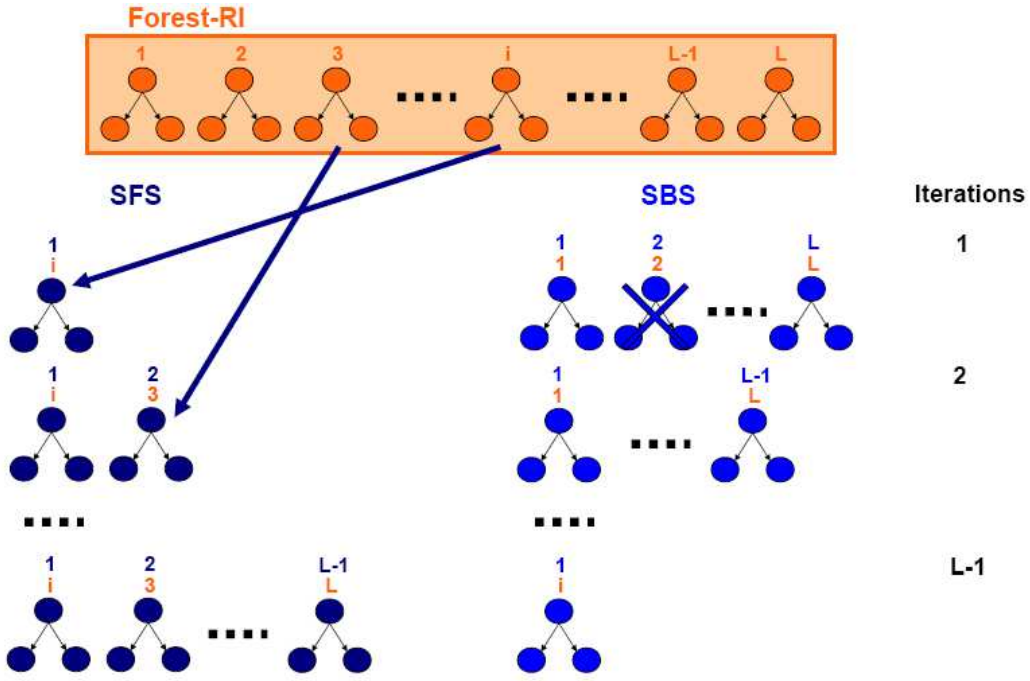

FIG. 4.4: Illustration des processus de SFS et SBS, appliqués à une forêt aléatoire de L arbres.

Le critère d'arrêt utilisé pour ce type de processus est généralement basé sur la convergence des performances, mais il peut également être défini par un nombre maximum d'itérations, de façon à fixer le nombre de classifieurs contenus dans le sous-ensemble final ([Roli 2001]). Pour nos expérimentations, nous avons décidé de laisser les deux processus de sélection explorer tous les sous-ensembles d'un nombre L' d'arbres allant de 1 à L, où L représente le nombre d'arbres dans la forêt initiale. De cette façon, nous avons la possibilité d'étudier l'évolution des performances des RF en fonction du nombre d'arbres qu'elles contiennent. Nous détaillons le protocole expérimental dans la section suivante.

4.3.1 Protocole Expérimental

Pour cette étude, nous avons repris les mêmes bases de données que pour l'étude du chapitre 3, qui sont décrites dans le tableau 3.1 de la section 3.2.1. Il s'agit alors de mettre en œuvre les deux processus de sélection SFS et SBS sur des forêts aléatoires induites avec l'algorithme Forest-RI.

Comme nous l'avons précisé, nous avons choisi d'adopter l'approche *wrapper* qui nécessite donc de disposer d'un ensemble de données indépendant des ensembles d'apprentissage et de test. Celui-ci est appelé l'ensemble de validation. Aussi, nous avons cette fois divisé chaque base de données en trois sous-ensembles de données disjoints, chacun contenant un tiers des données disponibles. Comme pour nos précédentes expérimentations, cette séparation a été réalisée par tirages aléatoires sans remise, et cette procédure a été répétée 10 fois de sorte d'obtenir 10 découpages différents. Nous notons ces découpages $T_i = (T_{r_i}, T_{v_i}, T_{s_i})$, avec $i \in [1..10]$, et où T_{r_i}, T_{v_i} et T_{s_i} représentent respectivement les ensembles d'apprentissage, de validation et de test. De cette façon, l'utilisation d'un ensemble de données indépendant pour effectuer la sélection ne biaise pas les résultats obtenus, et l'ensemble de test pourra alors être utilisé pour mesurer de façon plus fiable les performances en généralisation de chaque sous-forêt obtenue.

Ensuite, pour chaque T_i, une forêt aléatoire a été apprise pour un nombre d'arbres de décision L fixé à 300. La valeur du paramètre K quant à elle a été fixée à l'aide des valeurs de K^* que nous avons obtenues au cours des expérimentations du chapitre 3. Ces valeurs de K^* sont reportées dans le tableau 3.4 de la section 3.2.3. Les méthodes SFS et SBS ont ensuite été appliquées aux forêts de 300 arbres, de sorte qu'à chaque itération, l'arbre ajouté ou retiré du sous-ensemble est celui qui permet d'obtenir les taux d'erreur minimum mesurés sur les T_{v_i}. À titre de comparaison, nous avons également mis en œuvre un troisième processus de sélection, que nous notons **SRS** pour **Sequential Random Selection**, qui reprend le principe de SFS mais en sélectionnant aléatoirement à chaque itération l'arbre à ajouter au sous-ensemble. De cette façon, nous pouvons simuler les résultats obtenus avec un processus d'induction statique qui ajoute de façon arbitraire les arbres de décision à la forêt. Enfin, les taux d'erreur en test de chaque sous-forêt obtenue au cours de ces trois procédures de sélection ont été mesurés sur les ensembles T_{s_i}. Nous obtenons donc trois tableaux de $L \times 10$ taux d'erreur pour chaque base de données. L'algorithme 9 résume le protocole expérimental complet de cette première partie de l'étude.

4.3.2 Résultats et Discussion

Pour discuter des résultats obtenus, nous avons d'abord moyenné les taux d'erreur sur les T_i, pour chacune des valeurs de L. Nous obtenons un taux d'erreur moyen pour chaque itération des processus de sélection. De cette façon, nous pouvons tracer les courbes de l'évolution des taux d'erreur des sous-forêts en fonction du nombre d'arbres de décision qu'elles contiennent ; ce que nous montrons dans les figures 4.5 et 4.6. Nous présentons également dans le tableau 4.2 les meilleurs taux d'erreurs obtenus avec les processus SFS et SBS et le nombre d'arbres de la sous-forêt qui a permis de les obtenir. Nous y reportons enfin les taux d'erreur obtenus avec les forêts de 300 arbres, pour comparaison.

La première observation que l'on peut faire à partir du tableau 4.2 est qu'en dépit de la sous-optimalité des méthodes de sélection SFS et SBS, elles ont permis la

Algorithme 9 Protocole Experimental 3

Entrée : N le nombre de données disponibles.
Entrée : M le nombre de caractéristiques disponibles.
Entrée : K^* la valeur du paramètre K de l'algorithme Forest-RI.
Sortie : $\epsilon[3][10][L]$ un tableau 3D pour conserver les taux d'erreur obtenus au cours des processus de sélection.

pour $i \in [1..10]$ **faire**
- $T_{r_i} \leftarrow$ tirer aléatoirement sans remise de $\frac{1}{3}$ des données disponibles.
- $T_{v_i} \leftarrow$ tirer aléatoirement sans remise de $\frac{1}{3}$ des données disponibles.
- $T_{s_i} \leftarrow$ données restantes.
- $h \leftarrow$ Forest-RI($L = 300$,$K = K^*$,T_{r_i}).
- $h^{(0)}_{SFS} \leftarrow \emptyset$.
- $h^{(0)}_{SBS} \leftarrow h$.
- $h^{(0)}_{SRS} \leftarrow \emptyset$.
- **pour** $j = 1$ *to* L **faire**
  - $h^{(j)}_{SFS} \leftarrow h^{(j-1)}_{SFS} \cup h(k)$, $k = argmin_{h(l) \notin h^{(j-1)}_{SFS}} \{\text{error}(h^{(j-1)}_{SFS} \cup h(l), Tv_i)\}$.
  - $h^{(j)}_{SBS} \leftarrow h^{(j-1)}_{SBS} \setminus h(k)$, $k = argmin_{h(l) \in h^{(j-1)}_{SBS}} \{\text{error}(h^{(j-1)}_{SBS} \setminus h(l), Tv_i)\}$.
  - $h^{(j)}_{SRS} \leftarrow h^{(j-1)}_{SRS} \cup h(k)$, $k = \text{random}(l), h(l) \notin h^{(j-1)}_{SRS}$.
  - $\epsilon(1)(i)(j) \leftarrow$ taux d'erreur de $h^{(j)}_{SFS}$ mesuré sur T_{s_i}.
  - $\epsilon(2)(i)(j) \leftarrow$ taux d'erreur de $h^{(j)}_{SBS}$ mesuré sur T_{s_i}.
  - $\epsilon(3)(i)(j) \leftarrow$ taux d'erreur de $h^{(j)}_{SRS}$ mesuré sur T_{s_i}.

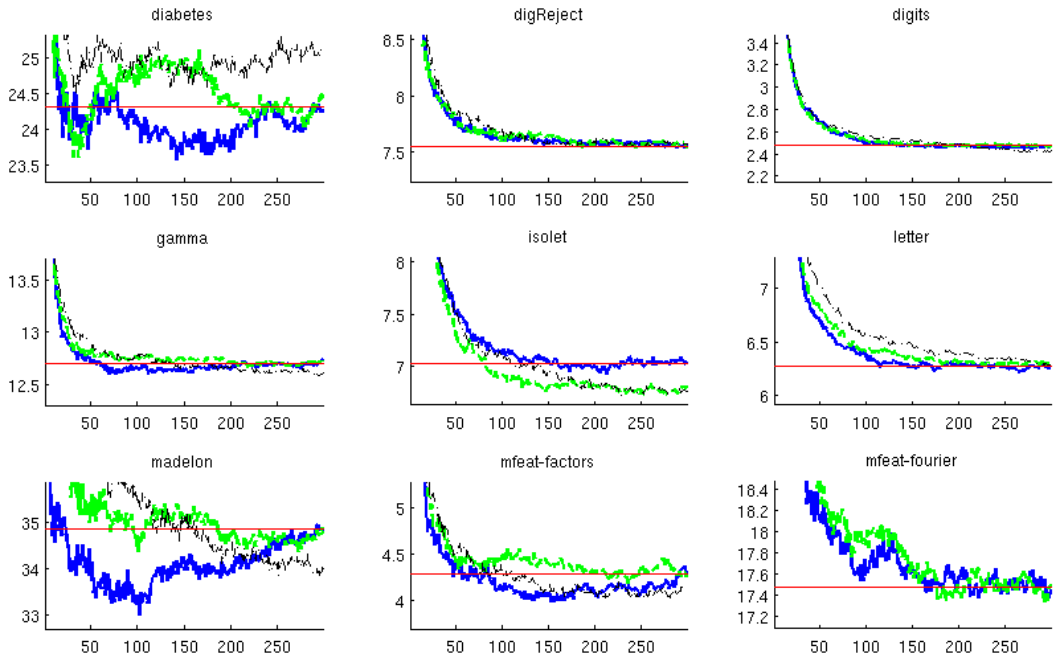

FIG. 4.5: Évolution du taux d'erreur en fonction du nombre d'arbres dans les sous-forêts pour les 9 premières bases de données. La ligne rouge représente le taux d'erreur obtenu avec la forêt initiale de 300 arbres. La courbe noire en pointillés représente les taux obtenus avec SRS. La courbe en vert représente les taux obtenus avec SBS et la courbe en bleu représente ceux obtenus avec SFS.

plupart du temps de produire des sous-forêts plus performantes que la forêt initiale de 300 arbres. On remarque cependant que le nombre d'arbres qui a été retiré de l'ensemble pour obtenir ce gain est parfois négligeable. C'est le cas notamment des bases Digits, DigReject, Mfeat-fourier, Mfeat-karhunen et Mfeat-zernike. De plus, les écarts entre les tailles des sous-forêts reportées dans ce tableau sont parfois très importants d'une base à une autre. À l'inverse des 5 bases que nous venons de citer, pour les bases Diabetes, Musk, Segment, Spambase ou encore Vehicle, la proportion d'arbres retirée de l'ensemble pour obtenir le taux minimum est très importante (jusqu'à plus de 90% des arbres pour Musk). Cette tendance est également observable sur les figures 4.5 et 4.6. L'allure des courbes d'évolution des taux d'erreur en fonction du nombre d'arbres est très différente d'une base à une autre, de sorte qu'elles atteignent leur minimum parfois pour des valeurs faibles en abscisse (Musk, Segment, Vehicle...), parfois au contraire pour des valeurs élevées (Isolet, Mfeat-karhunen, Mfeat-zernike, Waveform...). En regardant simplement les descriptions de ces bases en termes de nombre de caractéristiques, nombre de classes et nombre de données, nous ne trouvons pas d'explication à cela. C'est une particularité du comportement des sous-forêts qu'il serait alors intéressant d'étudier en perspectives de ces travaux.

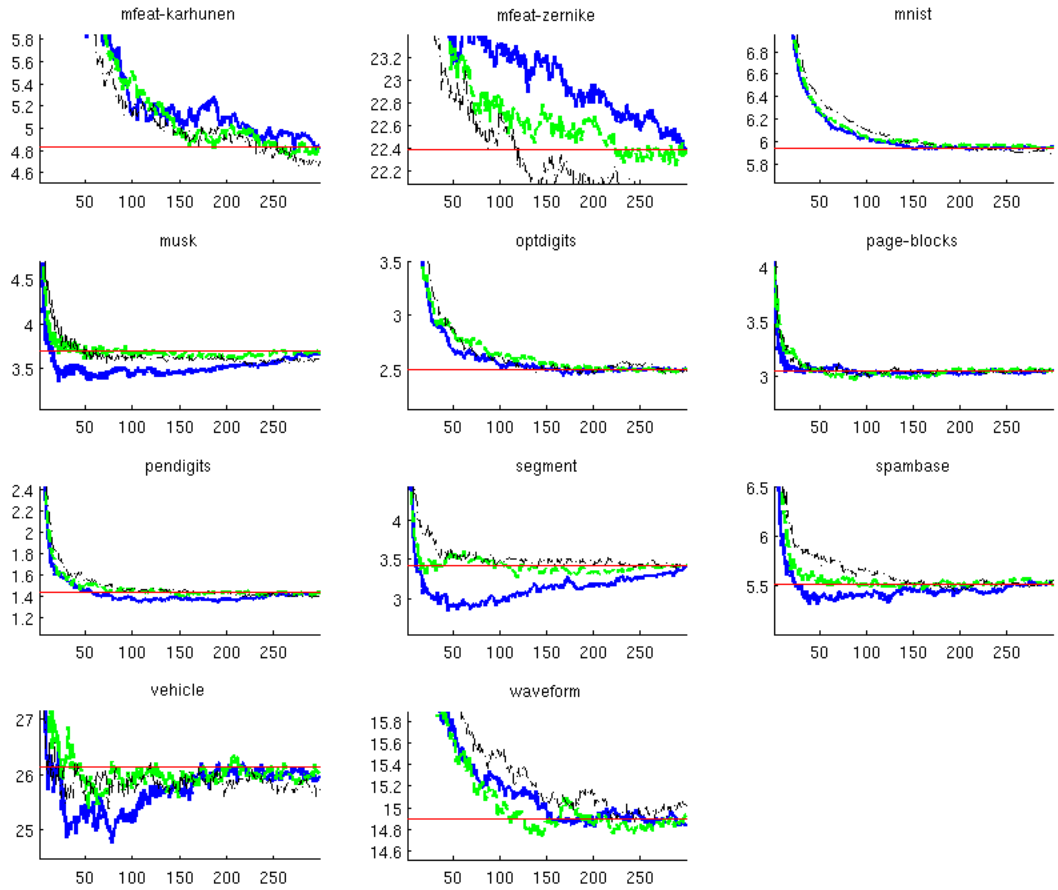

FIG. 4.6: Évolution du taux d'erreur en fonction du nombre d'arbres dans les sous-forêts pour les 12 dernières bases de données. La ligne rouge représente le taux d'erreur obtenu avec la forêt initiale de 300 arbres. La courbe noire en pointillés représente les taux obtenus avec SRS. La courbe en vert représente les taux obtenus avec SBS et la courbe en bleu représente ceux obtenus avec SFS.

On observe également à partir du tableau 4.2 que pour la plupart des bases, ce gain de performances n'est pas très élevé et ne semble par conséquent pas significatif. Pour nous en assurer, nous avons réalisé des tests de McNemar pour chaque itération des processus de sélection, en comparant à chaque fois la sous-forêt courante à la forêt initiale. Nous avons en particulier observé les résultats de ces tests pour les sous-ensembles qui ont permis d'obtenir un taux d'erreur strictement inférieur à celui de la forêt initiale, et ce pour chaque T_i. Nous synthétisons ces résultats dans le tableau 4.3, où nous présentons le nombre de sous-forêts, parmi les $300 \times 10 = 3000$ sous-forêts générées au cours de chaque processus de sélection, qui ont permis d'obtenir un gain significatif. On se rend compte alors que ces nombres sont souvent très faibles comparés au total des 3000 forêts générées. Cela signifie que dans une majorité des cas, le gain de performance obtenu n'est pas significatif.

Bases	SFS		SBS		Forest-RI
	tx d'erreur	arbres	tx d'erreur	arbres	300 arbres
Diabetes	23.57	142	23.61	32	24.31
Digits	2.45	276	2.46	289	2.47
DigReject	7.53	289	7.54	245	7.55
Gamma	12.60	76	12.67	219	12.71
Isolet	6.93	213	6.73	249	7.03
Letter	6.21	265	6.27	228	6.28
Madelon	33.01	103	34.33	101	34.88
Mfeat-fac	3.98	153	4.18	236	4.28
Mfeat-fou	17.37	297	17.33	292	17.46
Mfeat-kar	4.80	272	4.74	268	4.83
Mfeat-zer	22. 39	298	22.26	281	22.40
MNIST	5.92	210	5.93	288	5.95
Musk	3.35	21	3.61	49	3.70
OptDigits	2.43	168	2.47	274	2.50
Page-blo	3.01	105	2.96	88	3.06
Pendigits	1.34	191	1.40	154	1.44
Segment	2.85	44	3.26	119	3.43
Spambase	5.30	47	5.45	161	5.51
Vehicle	24.75	79	25.35	53	26.17
Waveform	14.81	262	14.71	145	14.90

TAB. 4.2: Taux d'erreurs moyens minimum obtenu au cours des processus SFS et SBS, et le nombre d'arbres de décision des sous-forêts correspondantes.

Pour certaines bases ce nombre peut même être nul (Mfeat-factors, Mfeat-Karhunen, ...), ce qui signifie que les processus de sélection n'ont pas permis de générer de sous-forêts significativement plus performantes, et qu'au mieux, ces sous-forêts sont statistiquement aussi performantes que la forêt initiale

Cependant, il est important de noter que la plupart du temps une proportion importante des sous-forêts (jusqu'à plus de 80% pour Madelon) permet d'obtenir un gain de performance, même s'il est relativement faible. Et finalement pour seulement 2 des 20 bases (Mfeat-factors et Mfeat-karhunen) ces gains de performances ne sont jamais significatifs, ni avec SFS, ni avec SBS. Pour les 18 autres, les deux méthodes de sélection de classifieurs ont permis d'obtenir au moins un sous-ensemble d'arbres significativement meilleur que la forêt initiale. Tout cela prouve que certains arbres, lorsqu'ils sont ajoutés à une forêt aléatoire, ne contribuent pas à l'amélioration des performances en généralisation, et même parfois les détériorent sensiblement.

Un autre résultat surprenant que l'on peut souligner en regardant les figures 4.5 et 4.6, est que le taux d'erreur de la forêt initiale, représenté par une ligne horizontale rouge sur les diagrammes, est atteint au cours des processus de sélection

Bases	SFS	SBS
Diabetes	93	4
Digits	12	0
DigReject	113	1
Gamma	47	2
Isolet	1	9
Letter	20	42
Madelon	214	47
Mfeat-factors	0	0
Mfeat-fourier	5	0
Mfeat-karhunen	0	0
Mfeat-zernike	2	43
MNIST	6	2
Musk	641	27
OptDigits	9	2
Page-blocks	60	56
Pendigits	89	21
Segment	157	24
Spambase	155	13
Vehicle	20	5
Waveform	3	46

TAB. 4.3: Nombre total de sous-forêts ayant permis d'obtenir un gain de performances significatif selon McNemar

par des sous-forêts relativement petites. C'est le cas presque systématiquement avec un nombre d'arbres inférieur à la moitié des arbres disponibles, et même parfois avec des sous-ensembles ne contenant qu'un dixième de ces arbres (Diabetes, Musk, Segment, Spambase ou encore Vehicle). On le constate par le fait que les courbes bleues et vertes qui représentent les taux d'erreur obtenus respectivement avec SFS et SBS, atteignent la ligne rouge pour des quantités d'arbres inférieures à 150 et parfois donc inférieures à 30. Ce résultat est intéressant car il indique que parmi l'ensemble des arbres induits de façon arbitraire au cours de la construction d'une forêt, quelques-uns seulement coopèrent bien mieux les uns avec les autres. S'il était possible alors d'identifier ces arbres et de mieux comprendre les raisons pour lesquelles cette combinaison de quelques arbres surpasse la forêt pourtant plus grande, il serait probablement possible de modifier le processus d'induction de forêt pour qu'il n'induise que ces quelques arbres. On peut alors imaginer concevoir un algorithme d'induction de forêts aléatoires plus rapide en apprentissage et en prédiction, et qui permettrait la plupart du temps de produire des classifieurs plus performants. Mais avant d'en arriver là, il faut pouvoir identifier des propriétés et des mécanismes particuliers qui expliquent ce gain de performances. Pour cela nous nous intéressons aux propriétés de *force* et de *corrélation* dont Breiman montre dans [Breiman 2001]

qu'elles sont cruciales pour l'induction d'une forêt performante. Notre but est de transposer ces conclusions théoriques à la mise en œuvre d'un processus de construction dynamique de forêt qui les utiliserait pour guider l'induction des arbres. Nous donnons dans la suite de cette étude des éléments pour confirmer et préciser le lien entre ces deux propriétés et les performances des forêts aléatoires.

4.4 De l'étude de la *force* et de la *corrélation* dans les sous-forêts

Nous venons de mettre en évidence l'intérêt de pouvoir identifier les arbres d'une forêt qui coopérent le mieux, et de pouvoir expliquer ce fonctionnement à l'aide d'une ou plusieurs propriétés. Le but maintenant est donc de trouver ces propriétés, de mettre en évidence leur relation avec les performances, et d'identifier la nature de cette relation pour pouvoir *in fine* l'exploiter dans un processus d'induction dynamique. Pour cela, nous avons décidé de suivre une des pistes dressées par Breiman dans [Breiman 2001]. Il y démontre que l'on peut borner l'erreur en généralisation des forêts aléatoires à l'aide de deux propriétés importantes : la *force*, notée s, et la *corrélation*, notée $\overline{\rho}$. Plus précisément, la borne qu'il obtient s'exprime en fonction du rapport $\frac{\overline{\rho}}{s^2}$. Nous donnons tous les détails de cette démonstration dans la section 2.5, ainsi que la définition de ces deux mesures. La conclusion importante de cette démonstration est que plus ce rapport est faible et meilleure sera la forêt résultante — pour citer l'article, "*the smaller it is, the better*". L'idée ici est donc d'observer l'évolution de ce rapport en fonction des performances des sous-forêts obtenues tout au long de la procédure de sélection d'arbres. De cette façon nous vérifions expérimentalement que ces propriétés sont effectivement corrélées avec les performances des forêts et nous identifions ce lien à travers le rapport $\frac{\overline{\rho}}{s^2}$.

Pour cela, nous avons mesuré la *force* et la *corrélation* telles qu'elles sont définies dans la démonstration de Breiman, pour chaque sous-forêt obtenue et nous avons calculé ensuite le rapport $\frac{\overline{\rho}}{s^2}$. La figure 4.7 montre les résultats que nous avons obtenus sous la forme de nuages de points, chaque point représentant une sous-forêt en fonction de son taux d'erreur en test et de la valeur de $\frac{\overline{\rho}}{s^2}$. D'après les résultats théoriques, le lien entre ce rapport et les performances devrait s'illustrer sur ces diagrammes par des nuages de points dont les droites de régression devraient être strictement croissantes. Quelle que soit la forme du nuage, les points qui correspondent aux valeurs de $\frac{\overline{\rho}}{s^2}$ les plus faibles sont également ceux qui devraient correspondre aux taux d'erreur les plus faibles. La pente de la droite de régression d'un tel nuage de points devrait donc obligatoirement être positive.

Nous avons tracé cette droite de régression pour chacun des nuages de points de la figure 4.7. Nous constatons alors que dans une majorité des cas, la pente est effectivement positive, ce qui confirme les conclusions de Breiman, mais pour 6 des bases elle est clairement négative (Diabetes, DigReject, Gamma, Letter, Pageblocks et Pendigits). La tendance des valeurs de $\frac{\overline{\rho}}{s^2}$ à diminuer pour des taux d'erreur décroissants n'est donc pas toujours vérifiée.

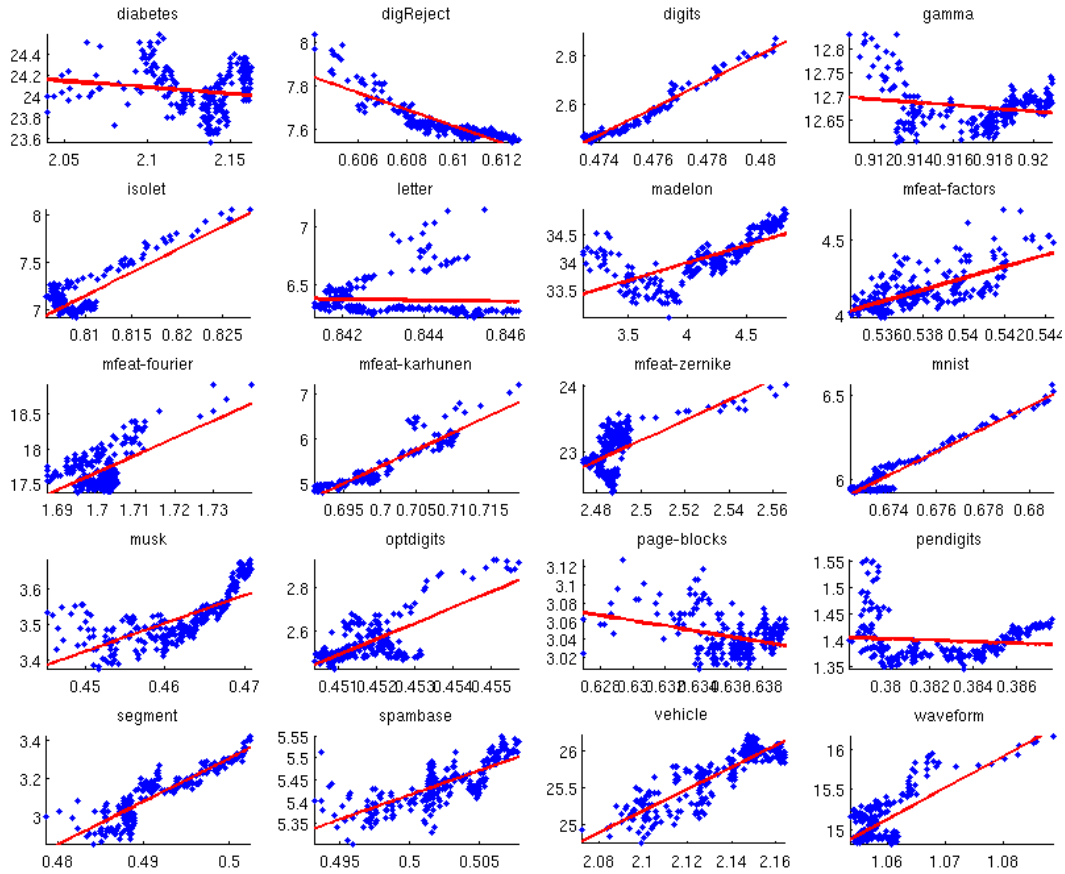

FIG. 4.7: Taux d'erreur des sous-forêts en fonction de $\frac{\overline{\rho}}{s^2}$. Les lignes rouges représentent les droites de régression des nuages de points.

Cependant lorsque l'on regarde la forme de ces nuages de points et les écarts de certains d'entre eux à la droite de régression, notamment pour les 6 bases incriminées, on s'apperçoit que la variation du rapport en fonction du taux d'erreur n'est pas monotone. Pour les bases Diabetes, Gamma, Letter et Page-blocks par exemple, que nous isolons en figure 4.8, sans tracer la droite de régression, le lien entre les performances et $\frac{\overline{\rho}}{s^2}$ n'est pas tellement identifiable. De notre point de vue cette observation illustre en réalité le fait que les nuages sont déformés par une minorité de points atypiques, pour lesquels les valeurs de $\frac{\overline{\rho}}{s^2}$ sont faibles alors que les taux d'erreurs eux sont plutôt élevés en comparaison avec les autres sous-forêts. Ce phénomène est principalement dû aux sous-forêts de petites tailles, et ces points atypiques correspondent aux sous-ensembles d'un faible nombre d'arbres de décision. Comme nous pouvons le constater sur les figures 4.5 et 4.6, ou même encore sur la figure 4.1, les taux d'erreur obtenus avec des forêts constituées d'un faible nombre d'arbres — en général inférieur à 30 arbres — sont significativement plus élevés en comparaison des forêts plus grandes. D'un autre côté la *force* et

la *corrélation* étant des mesures statistiques, un faible nombre d'arbres dans la sous-forêt implique forcément que le calcul du rapport $\frac{\overline{\rho}}{s^2}$ ne soit pas aussi fiable que pour des sous-forêts plus grandes. Pour s'en rendre compte, nous pouvons tracer une deuxième série de diagrammes ; la figure 4.9 présente les courbes de l'évolution de $\frac{\overline{\rho}}{s^2}$ en fonction du nombre d'arbres dans les sous-forêts. Nous avons également isolé, en figure 4.8, les courbes des 4 bases que nous avons pris pour exemple plus haut. L'instabilité de ces courbes pour des forêts à faible nombre d'arbres illustre ce que nous venons de dire. Une trop faible quantité d'arbres dans une forêt biaise le calcul de $\frac{\overline{\rho}}{s^2}$ et rend les résultats très instables.

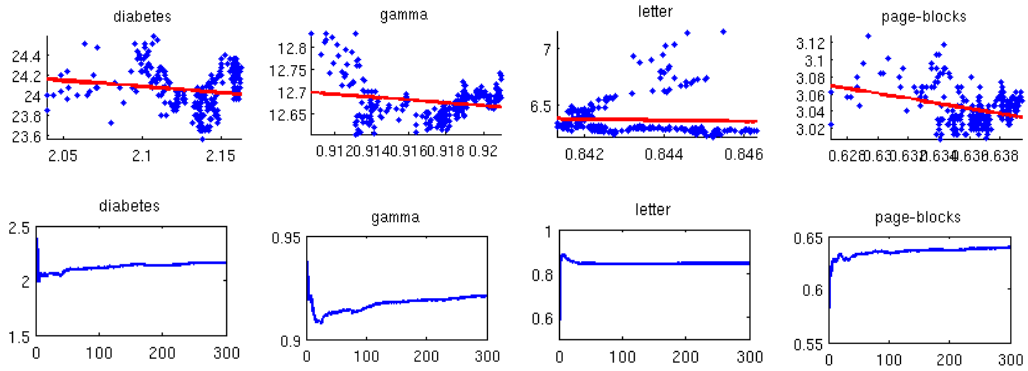

FIG. 4.8: Taux d'erreur des sous-forêts en fonction de $\frac{\overline{\rho}}{s^2}$ pour les bases Diabetes, Gamma, Letter et Page-blocks

En définitive, nous pensons que si les processus de sélection SFS et SBS sont utiles pour percevoir dans quelle mesure les performances des forêts peuvent être améliorées en retirant une certaine quantité d'arbres, ils fournissent des sous-forêts d'un nombre d'arbres systématiquement différent. La séquentialité de ces méthodes et ces différences de tailles des forêts rendent l'analyse des résultats plus difficile et moins fiable. Il faut donc pour appuyer nos conclusions mener ce même type d'expérimentations avec des processus de sélection qui ne seraient plus séquentiels et pour lesquels nous pourrions fixer la taille des sous-forêts générées. De cette manière, les résultats obtenus ne seraient plus biaisés par le nombre d'arbres des sous-ensembles que l'on cherche à comparer les uns aux autres. Ce sont pour toutes ces raisons que nous avons décidé de reproduire le même protocole expérimental que précédemment en utilisant cette fois des algorithmes génétiques au lieu des méthodes SFS et SBS, pour générer des sous-forêts.

4.5 Sélection de sous-forêts par algorithmes génétiques

Dans cette section, nous poursuivons l'étude de la *force* et de la *corrélation* dans les sous-forêts, en appliquant cette fois une méthode de sélection de classifieurs différente des méthodes de recherche séquentielle telles que SFS et SBS qui s'appuie

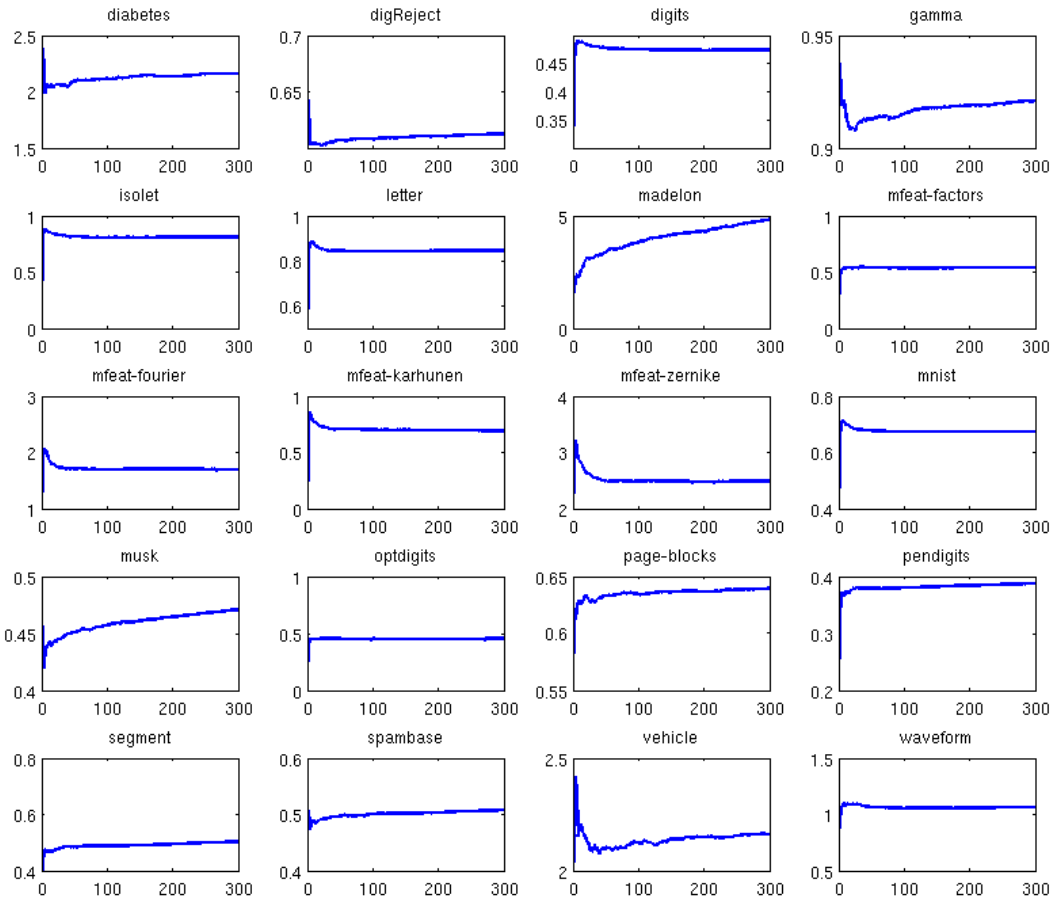

FIG. 4.9: Évolution du rapport $\frac{\bar{\rho}}{s^2}$ en fonction du nombre d'arbres dans les sous-forêts correspondantes

toujours sur une approche de type *wrapper*, mais qui permet cette fois de générer un ensemble de sous-forêts de taille fixe et dans une gamme de performances plus large et mieux répartie. Nous avons choisi d'utiliser des algorithmes génétiques pour la sélection de classifieurs car ils remplissent toutes ces conditions. Plus précisément, nous avons utilisé des algorithmes génétiques mono-objectifs dont la fonction d'évaluation est directement basée sur les performances en généralisation des sous-forêts. Nous expliquons plus en détail le fonctionnement des algorithmes génétiques de façon générale, puis dans le cadre de la sélection de classifieurs dans la section suivante.

4.5.1 Algorithmes Génétiques

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes, et servent initialement à trouver une solution approchée, en un temps raisonnable, à un problème d'optimisation ([Goldberg 1989]). Pour la mise en œuvre d'un

algorithme génétique il faut s'intéresser dans un premier temps à trois principaux aspects :

- Une solution possible du problème d'optimisation est représentée "chromosomiquement" par une chaîne de lettres appartenant à un alphabet particulier. Le plus souvent cet alphabet est l'alphabet binaire et une solution est donc représentée par une chaîne de bits ; mais ça n'est pas une condition *sine qua non*. Un premier problème à régler avant de mettre en place un algorithme génétique est donc le codage des solutions du problème, c'est-à-dire le codage des chromosomes.
- Il est nécessaire également de réfléchir aux moyens de créer une population initiale de solutions possibles. Les algorithmes génétiques travaillent en effet non pas sur un individu mais sur un ensemble d'individus appelé population.
- Des opérateurs pseudo-aléatoires inspirés de la génétique (reproduction, croisement, mutation...) permettent au fur et à mesure de l'exécution de l'algorithme de faire évoluer la population, en assurant notamment que les générations successives seront de plus en plus adaptées à la résolution du problème. Pour cela ces opérateurs utilisent à la fois les principes de la survie des structures les mieux adaptées et les échanges d'information pseudo-aléatoires, pour constituer un algorithme d'exploration qui possède certaines des caractéristiques de l'exploration humaine. Ce mécanisme favorise donc les éléments les meilleurs, les plus aptes. De ce fait, la définition de ces opérateurs est nécessaire en préambule de la mise en place d'un algorithme génétique.

À partir de ces principes, un algorithme génétique comprend une phase d'initialisation et 3 étapes répétées en boucle à chaque génération, chacune consistant à appliquer aux individus de la population un opérateur :

- La phase d'initialisation est réalisée une seule fois au début de la procédure. Il s'agit d'initialiser chaque individu de la première génération de façon aléatoire. Une fois que le codage des chromosomes est défini, il est nécessaire d'initialiser chacun des gènes (éléments de la chaîne) aléatoirement selon son domaine de définition. Par exemple, si la fonction à optimiser compte deux paramètres a et b, dont les valeurs varient de 0 à 31 par pas de 1, chacun de ces paramètres pourra être codé par une chaîne de 5 bits ($2^5 = 32$). La population de la première génération sera donc constituée de n individus de la forme :

0	0	1	0	1	1	1	1	0	0
Paramètre a					Paramètre b				

FIG. 4.10: Exemples de chaîne de bits pour le codage d'un individu ($a = 5$,$b = 28$)

- La **reproduction** est le premier opérateur. Celui-ci consiste à copier ou non chaque individu de la population pour constituer la génération suivante. Le

principe pour la constitution de cette nouvelle génération est de conserver les individus les plus "forts" et d'éliminer les plus "faibles". Cette phase va donc nécessiter de mettre en place une fonction d'adaptation — souvent appelé fonction de *fitness* — pour chaque individu, qui va déterminer sa probabilité de faire partie de la génération suivante ; cette probabilité étant par conséquent directement liée à la "qualité" de la solution au problème d'optimisation. Il faut ensuite déterminer quels seront les individus qui survivront à la prochaine génération en fonction de ce score d'adaptation. Une des solutions les plus répandues est d'utiliser un tirage aléatoire à l'aide d'une roue de loterie biaisée pour laquelle chaque individu de la population courante occupe une section de la roue proportionnelle à son adaptation.

- Le **croisement** est un opérateur permettant de simuler le mélange des gènes qui a lieu lors de la reproduction naturelle d'un couple d'individus. Il permet de créer de nouvelles solutions à partir des anciennes, par échange de matériel génétique. Cet opérateur peut se décomposer en deux étapes : (i) les éléments issus de la reproduction sont appariés aléatoirement (ii) chaque paire formée peut subir une permutation (un nouveau tirage aléatoire est mis en place, guidé par une probabilité de croisement). Cette permutation peut s'effectuer en un ou plusieurs endroits de la chaîne (les sites de croisement) qui sont eux aussi choisis aléatoirement. La figure 4.11 illustre le fonctionnement de cet opérateur.

FIG. 4.11: Illustration de l'opérateur de croisement ([Adam 2001])

- La **mutation** est un opérateur qui permet de résoudre les problèmes d'inexploration de sous-espaces de solutions et donc de se sortir des problèmes d'*extrema* locaux. La mutation consiste à modifier la valeur d'un gène de façon aléatoire — là encore une probabilité de mutation est utilisée pour guider l'aléatoire.

En suivant ce schéma de fonctionnement un algorithme génétique est guidé par 5 principaux paramètres :

- le nombre de générations. Ce paramètre peut ne pas être nécessaire si un critère d'arrêt est mis en place.
- la taille de la population
- la probabilité de mutation qui va guider la décision de l'opérateur de mutation de modifier ou non la valeur d'un gène.
- la probabilité de croisement, pour décider de permuter ou non des parties d'une paire d'individus pour former de nouveaux individus.
- le nombre d'individus parmi les meilleures de leur génération à conserver dans la génération suivante. Le fait de conserve les r individus les plus "forts" dans la génération suivante, est désigné par le terme élitisme.

Ces algorithmes sont souvent utilisés pour la sélection de classifieurs ([Opitz 1999b, Santos 2008]), car ils permettent en général de trouver un sous-ensemble de classifieurs proche du sous-ensemble optimal, en un temps raisonnable. Lorsqu'ils sont utilisés pour la sélection de classifieurs, chaque individu représente une solution et donc un sous-ensemble de classifieurs possibles. La plupart du temps, ces individus sont définis par des chaînes de bits, où chaque bit indique si le classifieur élémentaire qu'il désigne est présent ou non dans le sous-ensemble correspondant. Ces chaînes contiennent donc autant de bits qu'il y a de classifieurs dans l'ensemble initial.

Dans la section suivante nous donnons les détails de la mise en œuvre des algorithmes génétiques pour la sélection de sous-forêts.

4.5.2 Protocole Expérimental

Chaque individu de chaque génération représente donc une sous-forêt dont on contraint le nombre d'arbres. Les opérateurs de croisement et de mutation ont été redéfinis de sorte de maintenir cette contrainte. Les paramètres de l'algorithme quant à eux ont été fixés empiriquement, c'est à dire à des valeurs usuelles dans un premier temps, que nous avons ensuite affinées par des tests successifs. Ces valeurs sont les suivantes :

- nombre de générations : 300
- taille de la population : 50
- probabilité de mutation : 0.01
- probabilité de croisement : 0.60
- pourcentage d'élitisme : 0.80

Ensuite, nous avons fixé le nombre d'arbres dans la forêt initiale à 500 arbres. Un grand nombre d'arbres dans la forêt initiale nous permet de réitérer l'expérience pour des tailles de sous-forêts différentes, afin de pouvoir nous assurer que nos conclusions sont identiques pour des nombres d'arbres différents dans les sous-forêts. Nous

avons choisi de lancer la procédure de sélection pour des nombres d'arbres dans les sous-ensembles $L' \in [50, 100, 150, 200]$. Pour chaque base de données donc, nous avons lancé 4 fois le processus de sélection par algorithme génétique, *i.e.* pour les 4 valeurs de L'. Un seul découpage aléatoire des bases a été réalisé et chacune de ces 4 procédures de sélection a été lancée à partir du même ensemble d'apprentissage T_r et testé sur le même ensemble de test T_s.

Nous détaillons l'intégralité du protocole expérimental dans l'algorithme 10. Nous présentons dans la section suivantes les résultats que nous avons obtenus avec ce protocole.

Algorithme 10 Protocole Expérimental 4

Sortie : ϵ une liste pour conserver les taux d'erreur en test de toutes les sous-forêts
Sortie : s une liste pour conserver les valeurs de *force* de toutes les sous-forêts
Sortie : ρ une liste pour conserver les valeurs de *corrélation* de toutes les sous-forêts

Tirer aléatoirement sans remise $\frac{2}{3}$ des données disponibles pour former l'ensemble d'apprentissage T_r. Les données restantes forment l'ensemble de test T_s
pour $L' \in \{50, 100, 150, 200\}$ **faire**
  $G_1 \leftarrow$ ensemble de 50 sous-forêts de L' arbres choisis aléatoirement
  pour $i \in [1..300]$ **faire**
    $g \leftarrow$ les $50 \times 20\%$ meilleures sous-forêts de G_{i-1}
    $G_{i-1} \leftarrow G_{i-1}/g$
    $G_i \leftarrow g$
    $G_i \leftarrow G_i \bigcup$ mutation$(G_{i-1}, 0.01)$
    $G_i \leftarrow G_i \bigcup$ croisement$(G_{i-1}, 0.60)$
  pour tout nouvel individu l de G_i **faire**
    $\epsilon \leftarrow \epsilon \bigcup$ taux d'erreur de l sur T_s
    $s \leftarrow s \bigcup$ *force* de l sur T_s
    $\rho \leftarrow \rho \bigcup$ *corrélation* de l sur T_s

4.5.3 Résultats et Discussion

Pour analyser les résultats de cette expérience nous avons relevé plusieurs mesures pour chaque nouvelle sous-forêt générée au cours de la sélection. C'est-à-dire qu'à chaque fois qu'un nouvel individu est créé à l'aide des opérateurs de mutation et de croisement, nous avons relevé les mesures suivantes :

- l'erreur en test sur l'ensemble T_s
- le résultat au test statistique de McNemar pour la comparaison des performances entre la sous-forêt et la forêt initiale
- la *force* s
- la *corrélation* $\overline{\rho}$
- le rapport $\frac{\overline{\rho}}{s^2}$

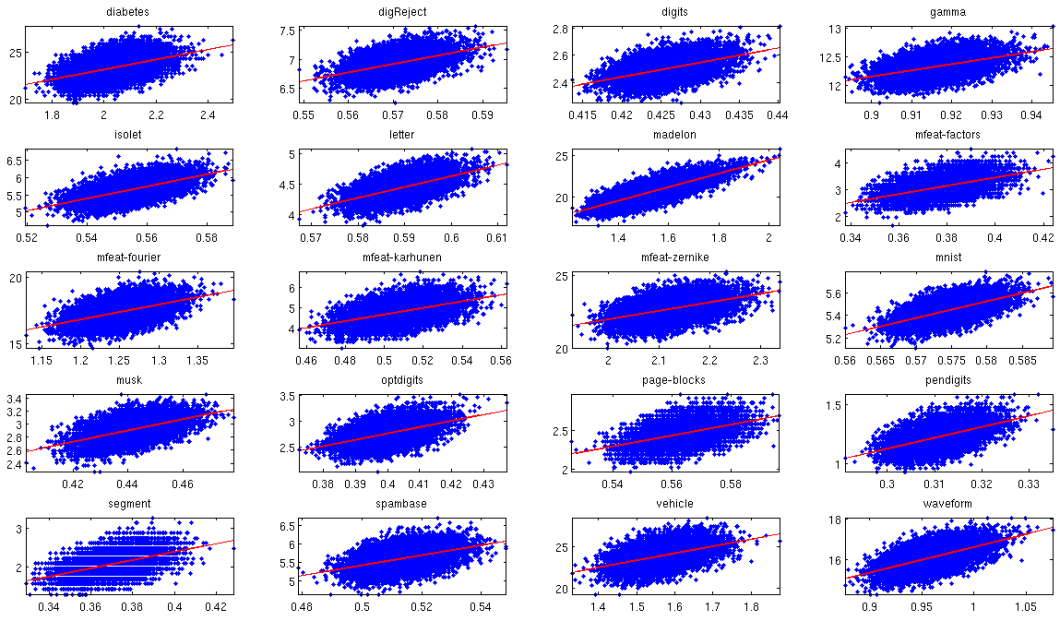

FIG. 4.12: Taux d'erreur en test en fonction de $\frac{\overline{\rho}}{s^2}$ obtenus au cours du processus de sélection par algorithme génétique, pour des sous-forêts de 50 arbres. Chaque point représente une de ces sous-forêts avec en abscisse sa valeur de $\frac{\overline{\rho}}{s^2}$ et en ordonnée son taux d'erreur sur l'ensemble de test. La ligne rouge représente la droite de régression du nuage de points.

De cette façon, nous disposons d'un grand nombre de mesures sur une large gamme de performances, puisque les "bons" individus comme les "mauvais" ont été considérés pour ces relevés. Si nous traçons avec ces nouveaux résultats les mêmes nuages de points que pour la figure 4.7, c'est-à-dire les diagrammes des taux d'erreur en fonction du rapport $\frac{\overline{\rho}}{s^2}$ pour toutes les sous-forêts générées, nous obtenons une distribution de points bien plus représentative qu'avec les méthodes SFS et SBS. Ces résultats sont présentés dans les figures 4.12, 4.13, 4.14 et 4.15. Cette fois-ci, la conclusion est sans équivoque : nous obtenons des résultats cohérents avec la théorie, à savoir une valeur du taux d'erreur en généralisation qui tend à décroitre pour des valeurs de $\frac{\overline{\rho}}{s^2}$ qui diminuent. C'est le cas pour les quatres procédures de sélection lancées sur chaque base de données. Cela prouve notamment que les résultats et les conclusions sont similaires quel que soit le nombre d'arbres dans les sous-forêts. De plus, si on observe les valeurs de $\frac{\overline{\rho}}{s^2}$ d'une base de données à une autre, on constate que les problèmes pour lesquels $\frac{\overline{\rho}}{s^2}$ est grand présentent des taux d'erreur grands également. Par exemple pour les bases Diabetes, Madelon, Mfeat-Zernike et Vehicle, pour lesquelles les taux d'erreur en test sont supérieurs à 20%, les valeurs de $\frac{\overline{\rho}}{s^2}$ se situent aux alentours de 2. À l'inverse, pour les bases de données Digits, MFeat-Factors, Musk, Opt-Digits ou encore Segment, pour lesquelles les taux d'erreur approchent les 2.5%, les valeurs de $\frac{\overline{\rho}}{s^2}$ se situent aux alentours de 0.4. Là encore, cette observation vient confirmer

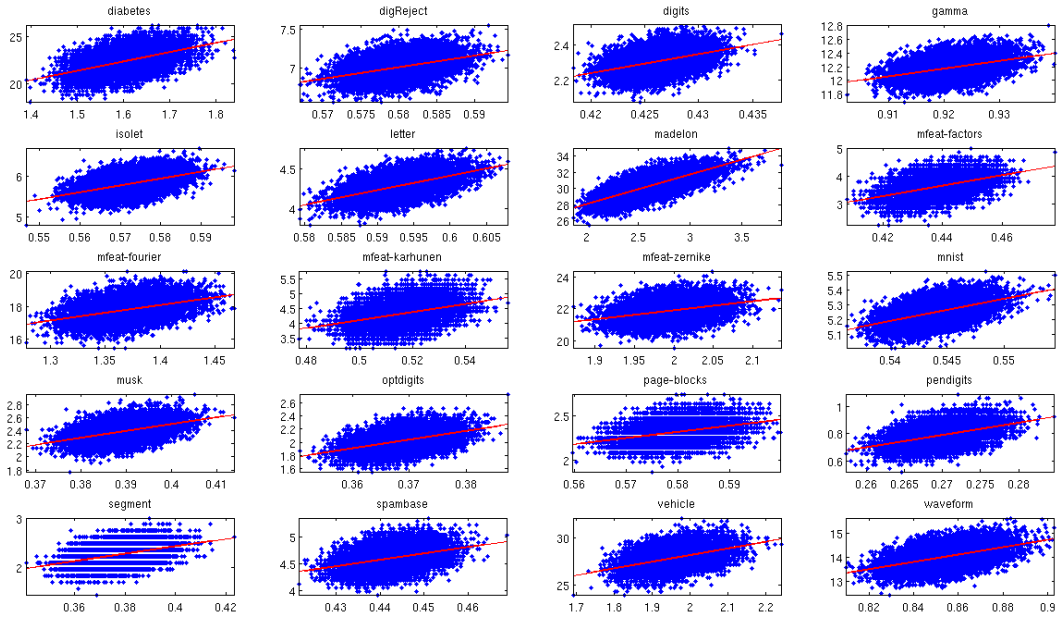

FIG. 4.13: Taux d'erreur en test en fonction de $\frac{\overline{\rho}}{s^2}$ obtenus au cours du processus de sélection par algorithme génétique, pour des sous-forêts de 100 arbres. Chaque point représente une de ces sous-forêts avec en abscisse sa valeur de $\frac{\overline{\rho}}{s^2}$ et en ordonnée son taux d'erreur sur l'ensemble de test. La ligne rouge représente la droite de régression du nuage de points.

les résultats théoriques de Breiman, puisque ce dernier a introduit le rapport $\frac{\overline{\rho}}{s^2}$ en prouvant qu'il constituait une borne de l'erreur en généralisation (*cf.* section 2.5).

En conclusion, cette étude prouve que les propriétés de *force* et de *corrélation* sont liées aux performances en généralisation des forêts aléatoires, via le rapport $\frac{\overline{\rho}}{s^2}$. Ces résultats démontrent que ces propriétés permettent d'expliquer les variations de performances d'une forêt aléatoire à une autre. Si nous disposons de deux forêts aléatoires différentes, nous savons que celle qui présentera la valeur de $\frac{\overline{\rho}}{s^2}$ la plus faible sera vraisemblablement la plus performante des deux. Ce résultat est très intéressant dans l'optique de construction dynamique de forêts que nous adoptons dans le chapitre 5. Il nous indique en effet qu'en réussissant à diriger l'induction des arbres de décision de sorte que $\frac{\overline{\rho}}{s^2}$ diminue, nous avons toutes les chances de réussir à en optimiser les performances en généralisation.

4.6 Conclusion

Nous avons présenté dans ce chapitre une deuxième contribution à l'étude du comportement des forêts aléatoires, en examinant de plus près l'apport de chaque arbre aux performances de l'ensemble. L'idée a été de générer un grand

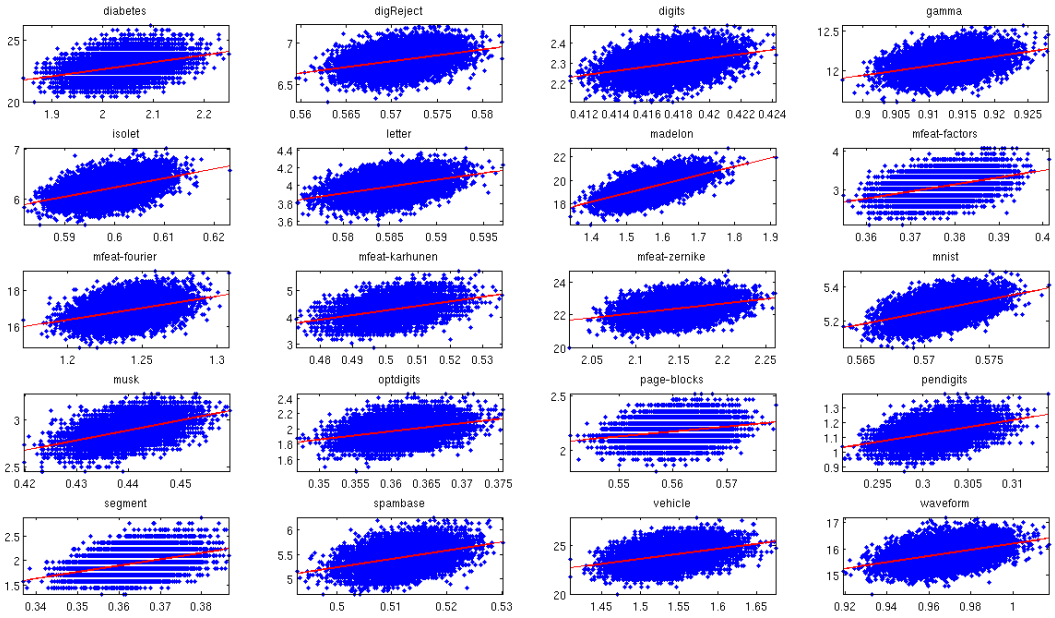

FIG. 4.14: Taux d'erreur en test en fonction de $\frac{\overline{\rho}}{s^2}$ obtenus au cours du processus de sélection par algorithme génétique, pour des sous-forêts de 150 arbres. Chaque point représente une de ces sous-forêts avec en abscisse sa valeur de $\frac{\overline{\rho}}{s^2}$ et en ordonnée son taux d'erreur sur l'ensemble de test. La ligne rouge représente la droite de régression du nuage de points.

nombre de forêts aléatoires présentant des performances les plus variées possibles et d'expliquer ces variations de performances. L'intérêt de cela est de définir des mécanismes pouvant permettre le guidage de l'induction des arbres dans une forêt, de sorte que chacun d'eux coopère au mieux avec le reste de l'ensemble. Il s'agit en cela d'un préambule à la conception d'un algorithme d'induction dynamique de forêts aléatoires, que nous menons dans le chapitre suivant.

Nous avons montré dans un premier temps que, dans la plupart des cas, en retirant d'une forêt certains des arbres de décision qu'elle contient, il est possible d'obtenir un gain de performances statistiquement significatif. Plus surprenant, la quantité d'arbres retirée de l'ensemble pour obtenir la sous-forêt la plus performante parmi l'ensemble des sous-forêts générées, est particulièrement importante pour certaines des bases de données (jusqu'à 90% des arbres dans certains cas). Ces résultats mettent en évidence l'intérêt de pouvoir identifier et caractériser ces arbres, et ceci pour permettre à un processus d'induction dynamique de cibler la contruction d'une forêt sur ce type d'arbre en particulier.

Nous avons mis ensuite en évidence les liens entre le rapport $\frac{\overline{\rho}}{s^2}$ et les performances en généralisation. Le premier constat que nous faisons est que nos résultats expérimentaux sont cohérents avec la théorie puisqu'ils illustrent que pour des valeurs $\frac{\overline{\rho}}{s^2}$ qui diminuent, l'erreur en test diminue également. Nous avons montré

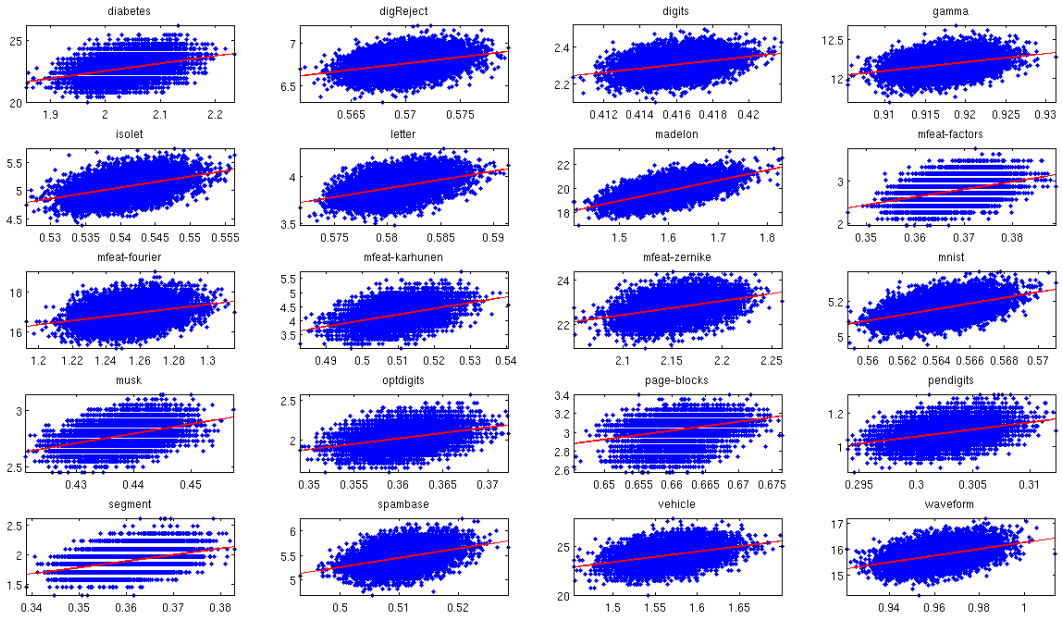

FIG. 4.15: Taux d'erreur en test en fonction de $\frac{\overline{\rho}}{s^2}$ obtenus au cours du processus de sélection par algorithme génétique, pour des sous-forêts de 200 arbres. Chaque point représente une de ces sous-forêts avec en abscisse sa valeur de $\frac{\overline{\rho}}{s^2}$ et en ordonnée son taux d'erreur sur l'ensemble de test. La ligne rouge représente la droite de régression du nuage de points.

aussi que d'un point de vue plus global pour les problèmes pour lesquels $\frac{\overline{\rho}}{s^2}$ est faible, les taux d'erreur sont en moyenne très faibles également. Ces résultats mettent en exergue l'intérêt de l'utilisation de ces deux propriétés pour le guidage d'un processus d'induction dynamique de forêts aléatoires. La perspective directe de cette étude est la mise en pratique de ces conclusions, avec la conception d'un algorithme d'induction dynamique. Dans le chapitre suivant nous présentons un algorithme original d'induction dynamique de forêts aléatoires qui s'appuie directement sur ces conclusions.

Cependant, pour compléter ces résultats, plusieurs autres pistes peuvent être suivies à court ou moyen terme. Par exemple, un des point fort de l'algorithme Forest-RI est que l'utilisation du Bagging lui permet de disposer des mesures out-of-bag pour l'estimation des performances en généralisation. Ces mesures pourraient alors directement être utilisées pour le guidage de l'algorithme d'induction dynamique. Bien que, comme nous l'avons évoqué en introduction de ce chapitre, leur fiabilité est parfois remise en question (*cf.* travaux de Banfield *et al.* dans [Banfield 2006]), il serait quand même intéressant de reproduire le schéma expérimental que nous venons de suivre avec les mesures out-of-bag au lieu des propriétés de *force* et de *corrélation*. Si les résultats obtenus mettent en évidence une corrélation entre ces mesures et l'erreur en test, nous pourrions envisager de guider l'algorithme d'induction dynamique par l'optimisation d'un ou plusieurs critères, qui s'appuierait simultanément

sur la *force*, la *corrélation* et l'erreur out-of-bag.

Chapitre 5

Construction Dynamique de Forêts Aléatoires

Sommaire

5.1 Introduction

Les travaux présentés dans ce chapitre vise à mettre à profit les résultats et conclusions du chapitre 4 pour concevoir un processus d'induction de forêts aléatoires.

Nous avons montré dans ce chapitre, que les processus d'induction statique tels qu'ils sont mis en œuvre dans les algorithmes Forest-RI et Forest-RK présentent des inconvénients majeurs. En premier lieu, bien que la convergence des performances ait été démontrée mathématiquement ([Breiman 2001]) et expérimentalement (*cf.* section 4.1), il est assez difficile d'appréhender le nombre d'arbres qu'il est nécessaire d'induire dans une forêt pour obtenir des performances raisonnablement bonnes. Nous avons montré dans la section 4.1 que ce nombre est par ailleurs très différent d'une base à une autre et qu'il n'est pas relié *a priori* aux nombres de données, de caractéristiques et de classes. Ensuite, nous avons expliqué que l'induction statique des forêts nécessitait de construire et d'ajouter les arbres de façon indépendante. Cela implique que l'apport de chaque arbre à l'amélioration de l'erreur en généralisation n'est pas garanti. Nous avons même montré que certains de ces arbres détérioraient ces taux d'erreur. Nous avons conclu de cela qu'il est alors plus pertinent d'adopter une approche d'induction dynamique de forêts aléatoires.

Pour concevoir un tel algorithme d'induction dynamique, il est cependant nécessaire de réfléchir à un moyen de guider l'apprentissage de l'ensemble, pour faire en sorte que chaque arbre contribue au mieux à l'amélioration de l'erreur en généralisation. En d'autres termes, il faut trouver un moyen de forcer l'apprentissage des arbres à s'adapter le plus possible à l'ensemble déjà construit. Pour cela, concrétement, il faut trouver un critère pour guider l'induction de sorte qu'elle permette de minimiser l'erreur ene généralisation. Nous avons montré dans le chapitre 4 que cette erreur en généralisation pourrait être reliée "directement" aux propriétés de *force* et de *corrélation*. En effet, la propriété de *force* permet dans ce cadre d'exprimer la capacité de la forêt à fournir des prédictions de confiance. Elle est calculée en faisant la moyenne des marges sur l'ensemble des données (*cf.* section 2.5). Or, la marge permet de quantifier la qualité d'une prédiction en s'appuyant sur la quantité d'arbres ayant voté pour la bonne classe de la donnée. Par conséquent, plus la *force* est grande, et plus on peut avoir confiance dans les prédictions d'une forêt. La propriété de *corrélation* quant à elle exprime la capacité des arbres de décision à fournir des prédictions décorrélées. Il s'agit en cela d'une mesure que l'on peut relier à la propriété de diversité (*cf.* section 1.3). Plus cette mesure est faible, et plus la diversité dans l'ensemble est grande. En résumé, il est donc nécessaire pour obtenir une forêt performante de construire un ensemble d'arbres capables tout d'abord de s'accorder au mieux sur les bonnes prédictions, mais aussi d'être le plus possible en désaccord sur les prédictions incorrectes. Par conséquent, une amélioration des performances des forêts repose sur ce double objectif : maximiser la *force* et minimiser la *corrélation*.

Sur la base de cette interprétation des performances des forêts, il faut faire

plusieurs choix pour la conception d'un algorithme d'induction dynamique de forêts aléatoires. Tout d'abord, il faut décider du critère pour guider l'induction. Ce critère doit permettre d'indiquer à l'algorithme d'induction des arbres, à la fois comment renforcer les bonnes prédictions, et à la fois comment compenser les erreurs. Ensuite, il faut trouver un moyen d'agir sur l'apprentissage des arbres en fonction de ce critère. Enfin, il faut définir une condition d'arrêt qui nous permettrait de déterminer *a priori* quand le nombre d'arbres dans l'ensemble est suffisant pour atteindre une erreur en généralisation raisonnablement faible.

Dans ce chapitre, nous présentons tout d'abord nos choix pour les deux premières de ces trois étapes. Dans une première section, nous expliquons quel critère nous avons choisi et pourquoi nous l'avons choisi. Puis, nous détaillons le procédé que nous avons mis en œuvre pour utiliser ce critère au sein du processus d'induction d'arbre. Le choix de ce critère et le choix de ce procédé définissent notre procédure d'induction dynamique que nous nommons *Dynamic Random Forest* (DRF). Dans une deuxième section, nous proposons plusieurs variantes de cet algorithme, que nous évaluons en les comparant aux algorithmes Forest-RI et Forest-RK. Nous montrons que certaines de ces variantes permettent d'améliorer de façon significative les performances en généralisation en comparaison avec les processus d'induction statique, démontrant ainsi l'apport de la construction dynamique aux forêts aléatoires. Dans une troisième section, nous discutons de l'évolution et de la convergence de l'erreur en généralisation au cours du processus d'induction dynamique. Nous démontrons notamment que la convergence de l'erreur des DRF est similaire à celle des processus d'induction statique. Cette analyse nous permet par ailleurs de dresser des pistes intéressantes pour définir le critère d'arrêt pour notre algorithme, bien que nous n'ayons pas eu le temps de le mettre en place dans le cadre des DRF. Enfin dans une dernière section, nous présentons une analyse de l'influence du processus d'induction dynamique sur les propriétés de *force* et de *corrélation*. Nous montrons avec cette analyse que le double objectif que nous avons présenté dans cette introduction est bien atteint dans une grande majorité des cas par les DRF.

5.2 *Dynamic Random Forest* : de l'induction dynamique de forêts aléatoires

5.2.1 *Dynamic Random Forest*

Dans cette section, nous présentons un processus d'induction dynamique de forêts aléatoires que nous appelons *Dynamic Random Forest* (DRF). Son but est d'adapter l'induction de chaque arbre à la forêt courante, pour construire un ensemble à la fois divers et dont on sait que ses membres coopèrent bien les uns avec les autres. Dans une taxonomie des méthodes d'induction d'EoC qui opposerait les approches déterministes, telles que le principe de Boosting, aux approches "randomisantes", telles que le Bagging par exemple, notre idée se situerait à mi-chemin entre ces deux paradigmes. Rappelons ici que le principe fondamental

du Boosting est de focaliser l'apprentissage de chaque classifieur élémentaire sur les erreurs de prédiction des classifieurs précédents. De ce fait, les classifieurs sont construits et ajoutés à l'ensemble de façon dépendante et séquentielle. C'est en cela que notre algorithme d'induction dynamique se rapproche de ce principe d'apprentissage.

Nous avons expliqué en introduction de ce chapitre que les propriétés de *force* et de *corrélation* permettent d'exprimer formellement l'objectif de l'induction dynamique : construire un ensemble d'arbres capable dans un premier temps de s'accorder au mieux sur les bonnes prédictions, et dans un second temps d'être le plus possible en désaccord sur les prédictions incorrectes.

En suivant ces deux idées, un processus d'induction dynamique de forêts doit guider la construction des arbres pour que ceux-ci renforcent le plus possible les majorités de bonnes prédictions, mais aussi et surtout pour qu'ils compensent au mieux les erreurs de la forêt courante. Un moyen d'y parvenir est d'indiquer à l'algorithme d'induction d'arbres les données qu'il faut privilégier au cours de l'apprentissage de chaque arbre, comme le fait le principe de Boosting. L'idée est de forcer l'induction du prochain arbre à se focaliser sur les données d'apprentissage qui ont été jusqu'à présent les moins bien classées par la forêt. De cette façon, on construit au fil des itérations des arbres qui compensent au mieux les erreurs de la forêt courante.

Pour définir la contribution de chaque donnée dans l'apprentissage des arbres, il est nécessaire alors d'estimer la capacité de la forêt courante à bien prédire sa classe. C'est en fonction de cette estimation que la donnée sera valorisée ou négligée au cours de l'induction du prochain arbre. Une estimation possible de cette capacité à fournir une bonne prédiction peut être donnée par la proportion d'arbres de la forêt ayant voté pour la bonne classe. Plus cette proportion est grande et moins cette donnée nécessite d'être apprise, et par conséquent moins le prochain arbre devrait se focaliser sur cette donnée. À l'inverse, plus cette proportion est faible, et plus cette donnée engendre d'erreurs de prédiction parmi les votes des arbres de la forêt, et donc plus il est nécessaire de s'y intéresser pour l'induction du prochain arbre.

On peut percevoir cette proportion d'arbres ayant voté pour la bonne classe, comme un score de confiance de la prédiction de la forêt pour une donnée. Plus ce score est grand et plus la forêt sait la prédire correctement. C'est là la raison de ce choix pour guider le processus d'induction dynamique. En utilisant ce score pour définir le comportement de l'induction des arbres vis à vis de chaque donnée, on espère renforcer au mieux cette confiance pour les données déjà bien prédites, tout en contrebalançant les scores faibles, pour les données mal prédites. De cette façon, le but est de progressivement augmenter la *force* tout en dimininuant la *corrélation*.

Nous avons choisi le critère pour guider l'induction dynamique, il faut maintenant définir sa prise en compte dans le processus. Là aussi, à l'image des principes de Boosting, nous choisissons pour cela de mettre en place une pondération des données. Cette pondération attribue des poids aux données, de sorte que la prise en compte de ces données dans l'apprentissage du prochain arbre soit accentuée si le score de

confiance est faible, et à l'inverse soit atténuée si le score de confiance est élevé. Par conséquent, ces poids doivent être augmentés pour les données qui présentent un score faible, et diminués pour les données avec un score élevé. En d'autres termes, le poids des données doit être inversement proportionnel à ce score de confiance.

L'idée générale est donc d'attribuer un poids identique à toutes les données au cours de l'initialisation de la procédure, puis de les "mettre à jour" à chaque itération. Le premier arbre de la forêt sera donc entrainé de façon classique, comme dans les processus statique d'induction, sans établir de distinction particulière dans la prise en compte de ces données. Puis, à l'issue de la première itération, après que le premier arbre ait été construit, les poids de chaque donnée seront modifiés de sorte qu'ils soient augmentés pour les données qui ont été mal classées par ce premier arbre, et diminués pour les données bien classées. Le deuxième arbre est alors entrainé sur ces données "re-pondérées". À la troisième itération, cette pondération est mise à jour sur la base des prédictions des deux premiers arbres simultanément. C'est-à-dire que des poids maximum seront attribués aux données qui ont été simultanément mal classés par ces deux arbres ; des poids un peu plus faibles seront attribués aux données qui ont été mal classées par l'un de ces deux arbres seulement ; et des poids minimum seront attribués aux données qui ont été bien classées par les deux arbres. Le troisième arbre est donc entrainé sur les données associées à ces poids recalculés, et ainsi de suite pour chaque itération du processus d'induction dynamique. À ce sujet, nous discutons plus en détails la mise en place de cette pondération dans la section suivante.

Breiman qualifie ce procédé dans [Breiman 1998] de rééchantillonnage adaptatif, et l'oppose dans l'approche au rééchantillonnage aléatoire du Bagging. On peut en effet percevoir le Bagging comme un principe de pondération aléatoires des données d'apprentissage, qui attribuerait par exemple un poids supérieur aux données qui ont été choisies à plusieurs reprises pour un même ensemble bootstrap. Ce que nous mettons en place avec notre algorithme *Dynamic Random Forest*, est un rééchantillonnage qui combine ces deux approches. Notre idée est de procéder en premier lieu à un tirage aléatoire avec remise de N données d'apprentissage, où N désigne le nombre de données dans la base, et de pondérer ensuite ces données comme nous venons de l'expliquer. La raison à cela est que nous souhaitons conserver les principes de randomisation mis en œuvre dans l'algorithme Forest-RI. Tout d'abord nous souhaitons utiliser l'algorithme d'induction d'arbres mis en œuvre dans Forest-RK et décrit dans l'algorithme 7 de la section 3.4. Cet algorithme implémente le processus de *Random Feature Selection* que nous estimons important pour l'induction de forêts aléatoires. Comme nous l'avons évoqué en section 3.3, ce principe de randomisation nécessite cependant pour être efficace d'induire les arbres jusqu'à leur taille maximale. Or, cela implique que l'ensemble des feuilles des arbres soient toutes pures, et donc que l'erreur en apprentissage soit nulle. Comme c'est le cas pour le principe de Boosting (*cf.* section 1.4.4), le procédé de rééchantillonnage adaptatif ne peut fonctionner ici si cette erreur est nulle, car comme nous venons de l'expliquer, c'est sur la base des erreurs de prédiction

pour les données d'apprentissage que sont calculés les poids. Pour remédier à cela, nous mettons donc en œuvre le principe de Bagging en amont. En utilisant des ensembles boostrap, on s'assure qu'à chaque itération une partie des données d'apprentissage ne sera pas utilisée du tout, et que par conséquent une proportion des arbres de la forêt est susceptible de se tromper dans la prédiction de chaque données.

Tous les choix que nous venons d'expliquer définissent la procédure des *Dynamic Random Forest*. Cette procédure établit une façon de réaliser l'induction dynamique d'une forêt aléatoire, mais il est évident que l'on peut imaginer procéder de façon différentes. Nous discutons de cela plus amplement en conclusion de ce chapitre, où nous détaillons les perspectives de ces travaux.

Pour le moment, nous résumons cette procédure de la façon suivant :

- Initialiser les poids des données à la même valeur.
- Pour chaque itération du processus :
    - Former l'ensemble bootstrap
    - Entrainer l'arbre sur les données bootstrap pondérées
    - Mettre à jour les poids des données d'apprentissage.

L'algorithme 11 donne tous les détails de cette procédure des *Dynamic Random Forest*. Dans cet algorithme la procédure $RndTree2$ fait donc référence à l'algorithme 7 qui décrit le principe d'induction d'un arbre aléatoire dans le cadre de la méthode Forest-RK (*cf.* section 3.4). La fonction $W_\alpha(c(\boldsymbol{x}, y))$, qui fournit un coefficient de pondération inversement proportionnel au score de confiance noté ici $c(\boldsymbol{x}, y)$, n'est pas définie ici. Pour ces expérimentations nous avons essayé plusieurs fonctions différentes, qui établit chacun une logique de pondération différente. Nous présentons dans la section suivante quelques-unes de ces solutions, et expliquons plus en détails ces différentes logique de pondération.

5.2.2 Fonctions de Pondération des Données

Il n'est pas évident de comprendre comment la pondération des données permet de moduler leur prise en compte lors de l'induction d'un arbre de décision. Pour cela, il faut bien avoir en tête la procédure de sélection de la règle de partitionnement à chaque nœud des arbres — À ce sujet, nous reportons le lecteur à la section 2.2.2. Chaque règle de partitionnement partage le groupe de données d'apprentissage du nœud concerné en deux sous-groupes, sur la base d'un certain critère d'évaluation. Lorsqu'aucun système de pondération n'est mis en place, ces critères sont basés sur les effectifs de chaque classe du problème dans chacun de ces sous-groupes. Dans le cas contraire, ces effectifs sont remplacés par la somme des poids des données contenues dans les sous-groupes.

Par conséquent, les valeurs des critères d'évaluation des règles de partitionnement sont très dépendantes des données présentant les poids les plus élevés. Selon qu'une donnée de poids très élevé se retrouve dans l'un ou l'autre des sous-groupes, les critères d'évaluation fournissent des valeurs très différentes. De cette façon, les

Algorithme 11 *Dynamic Random Forest*

Entrée : T l'ensemble d'apprentissage
Entrée : N le nombre de données dans T
Entrée : M le nombre de caractéristiques
Entrée : L le nombre d'arbres dans la forêt
Entrée : $W_\alpha(c(\boldsymbol{x}, y))$ une fonction de pondération inversement proportionnelle à $c(\boldsymbol{x}, y)$
Sortie : $foret$ l'ensemble des arbres qui composent la forêt construite

pour tout $\boldsymbol{x}_i \in T$ **faire**
  $D_1(\boldsymbol{x}_i) \leftarrow \frac{1}{N}$
pour l de 1 à L **faire**
  $arbre \leftarrow$ un arbre vide, i.e. composé de sa racine uniquement
  $Z \leftarrow 0$
  $T_l \leftarrow$ un ensemble bootstrap, dont les données sont tirées aléatoirement (avec remise) de T
  $gains \leftarrow$ un tableau M éléments
  pour tout m de 1 à M **faire**
    $gains[m] \leftarrow$ gain d'information de la caractéristique A_m en fonction de T_l
  $T_l \leftarrow T_l$ pondéré avec D
  $arbre.racine \leftarrow RndTree2(arbre.racine, T_l, gains)$
  $foret \leftarrow foret \bigcup arbre$
  pour tout $\boldsymbol{x}_i \in T$ **faire**
    $D_{l+1}(\boldsymbol{x}_i) \leftarrow W_\alpha(c(\boldsymbol{x}_i, y_i))$
    $Z \leftarrow Z + D_{l+1}(\boldsymbol{x}_i)$
  pour tout $\boldsymbol{x}_i \in T$ **faire**
    $D_{l+1}(\boldsymbol{x}_i) \leftarrow \frac{D_{l+1}(\boldsymbol{x}_i)}{Z}$
retour $foret$

partitionnements récursifs s'adaptent à ces poids pour faire en sorte de privilégier la prédiction des données de poids élevé. Cela se traduit par une tendance des arbres à rapidement séparer les données de poids élevé des données qui n'appartiennent pas à leur classe.

Nous présentons donc dans cette section trois fonctions $W_\alpha(c(\boldsymbol{x}, y))$ différentes, qui nous semblent intéressantes à utiliser comme fonction de pondération des données, et que nous avons testées avec les *Dynamic Random Forest*. Avant de les expliquer plus en détails nous définissons la fonction $c(\boldsymbol{x}, y)$ qui fournit le score de confiance d'une prédiction d'une forêt, pour la donnée d'apprentissage $\boldsymbol{x}$ (*cf.* section 5.2) :

$$c(\boldsymbol{x}, y) = \frac{1}{L} \times \sum_{i=1}^{L} I(h_i(x) = y) \tag{5.1}$$

où $I(.)$ est la fonction indicatrice qui renvoie 1 si la condition en argument est vraie et 0 sinon, et où L représente le nombre d'arbres dans la forêt. Étant donné que ce score est calculé à l'aide de la proportion d'arbres ayant votés pour la bonne classe, ses valeurs sont comprises dans l'intervalle $[0, 1]$.

Comme nous l'avons expliqué en détails dans la section précédente, ce score a été choisi pour les DRF comme critère pour guider l'induction dynamique. Il est utilisé pour définir la pondération des données de la méthode. Pour que cette pondération remplisse correctement son rôle, nous avons vu qu'il est nécessaire que les poids soient inversement proportionnels à ce score de confiance. Le point commun de toutes les fonctions de pondération que nous présentons dans cette section est donc qu'elles sont toutes décroissantes pour des valeurs croissantes de $c(\boldsymbol{x}, y)$ comprises dans l'intervalle $[0, 1]$. De cette façon, plus la sous-forêt courante accordera de votes à la vraie classe d'une donnée, et plus la pondération de celle-ci sera faible pour l'induction de l'arbre de l'itération suivante.

Fonction inverse : $W_\alpha(c(\boldsymbol{x}, y)) = \frac{1}{c(\boldsymbol{x},y)^\alpha}$

La première des fonctions que nous choisissons de mettre en œuvre ici, utilise un terme inversement proportionnel à $c(\boldsymbol{x}, y)$. L'équation de cette fonction est la suivante :

$$W_\alpha(c(\boldsymbol{x}, y)) = \frac{1}{c(\boldsymbol{x}, y)^\alpha} \tag{5.2}$$

Nous la représentons en figure 5.1, pour plusieurs valeurs de α. Comme le montre cette figure, le paramètre α permet d'accentuer ou d'affaiblir les écarts de pondération entre deux données qui présenteraient deux valeurs de $c(\boldsymbol{x}, y)$ différentes. La valeur renvoyée par la fonction $W_\alpha(c(\boldsymbol{x}, y))$ n'est pas importante en soi puisque les pondérations sont systématiquement normalisées à chaque itération. Ce qui est important, ce sont les écarts entre les poids pour deux valeurs de $c(\boldsymbol{x}, y_i)$ différentes. Ce sont ces écarts qui vont conditionner la prise en compte de chaque donnée dans les itérations suivantes. Avec cette fonction typiquement ces écarts sont décuplés pour des valeurs de α grandes. Pour illustrer cela prenons un exemple ; considérons 2 données $\boldsymbol{x}_1$ et $\boldsymbol{x}_2$, présentant les valeurs $c(\boldsymbol{x}_1, y_1) = 0.2$ et $c(\boldsymbol{x}_2, y_2) = 0.8$ — ce qui signifie qu'elles ont été correctement prédites par respectivement 20% et 80% des arbres de la sous-forêt. Calculons maintenant leur poids pour des valeurs de α égales à 0.5, puis à 5 :

$$\begin{aligned}
W_{0.5}(0.2) &= \frac{1}{0.2^{0.5}} = 2.23 \\
W_{0.5}(0.8) &= \frac{1}{0.8^{0.5}} = 1.12 \\
W_{5}(0.2) &= \frac{1}{0.2^{5}} = 3125 \\
W_{5}(0.8) &= \frac{1}{0.8^{5}} = 3.05
\end{aligned}$$

Après normalisation on obtient les valeurs de poids suivantes :

$$\begin{aligned} W_{0.5}(0.2) &= 0.66 \\ W_{0.5}(0.8) &= 0.33 \\ W_{5}(0.2) &= 0.999 \\ W_{5}(0.8) &= 9.75 \times 10^{-4} \end{aligned}$$

On constate alors que les écarts entre les poids sont bien plus importants dans le cas où $\alpha = 5$. Pour cette valeur de paramètre, les données mal classées par un nombre important d'arbres de la sous-forêt courante se verront attribuer un poids considérablement plus élevé pour l'induction du prochain arbre et à l'inverse les données bien classées par beaucoup de ces arbres n'auront pratiquement aucune influence dans la construction de la structure finale.

En définitive, avec une fonction de ce genre (DRF/W_5^{inv}), qui définit une logique de pondération plutôt radicale, seules les données qui sont correctement prédites par une très faible proportion d'arbres seront utilisées pour l'apprentissage du prochain arbre. Les autres données ne seront pratiquement pas prises en compte pour construire la structure de décision. Le risque à cela est de générer avec les nouveaux arbres de nouvelles erreurs de prédiction sur ces données qui ont été préalablement bien classées, mais qui le seront possiblement de moins en moins.

Par la suite nous notons DRF/W_α^{inv} l'algorithme *Dynamic Random Forest* qui utilise cette fonction inverse comme fonction de pondération.

Fonction polynomiale : $W_\alpha(c(\boldsymbol{x}, y)) = 1 - c(\boldsymbol{x}, y)^\alpha$

Une deuxième fonction que nous avons testée pour la pondération de l'algorithme DRF est la fonction polynomiale définie par :

$$W_\alpha(c(\boldsymbol{x}, y)) = 1 - c(\boldsymbol{x}, y)^\alpha \tag{5.3}$$

La figure 5.2 donne une représentation graphique de cette fonction pour des valeurs de $c(\boldsymbol{x}, y)$ comprises dans l'intervalle $[0, 1]$ et pour $\alpha \in \{0.5, 1, 2, 5\}$. Là encore, le paramètre α permet de moduler les écarts entre les poids correspondant à deux valeurs de $c(\boldsymbol{x}, y)$ différentes. En revanche, dans ce type de fonction polynomiale, les coefficients des différents degrés du polynome ne sont pas cruciaux dans la mesure où il n'ont pas d'influence sur ces écarts après normalisation. C'est la raison pour laquelle nous les avons fixés à 1 ici.

Ce type de fonction est intéressant car il permet d'obtenir des logiques de pondération plus variées, en retardant ou accélérant la diminution des valeurs de $W_\alpha(c(\boldsymbol{x}, y))$ pour des valeurs de $c(\boldsymbol{x}, y)$ croissantes. Pour $\alpha = 5$ par exemple, les données présentant des valeurs de $c(\boldsymbol{x}, y)$ inférieures à 0.5, se verront toutes attribuer le poids maximum tandis que celui-ci diminuera rapidement pour des données avec des $c(\boldsymbol{x}, y)$ immédiatement supérieurs à 0.5. En d'autres termes, toutes les don-

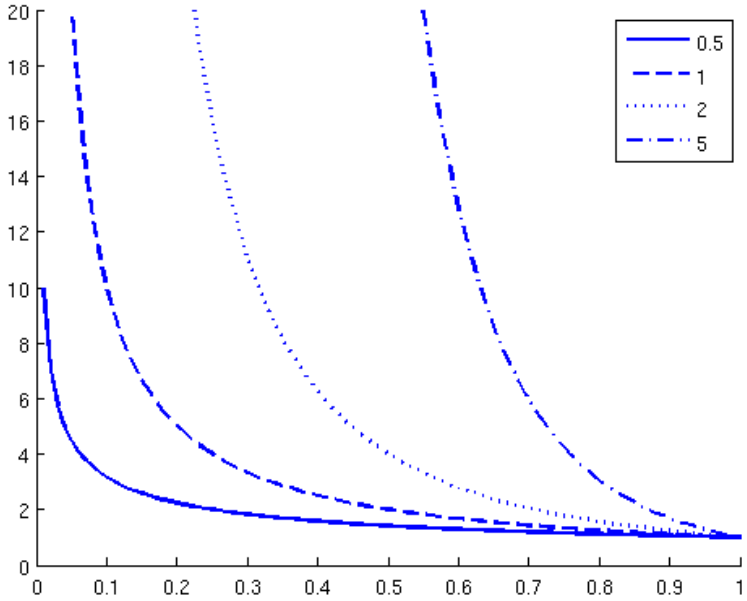

FIG. 5.1: fonction $W_\alpha(c(\boldsymbol{x}, y)) = \frac{1}{c(\boldsymbol{x},y)^\alpha}$ pour $\alpha = \{0.5, 1, 2, 5\}$

nées mal classées par plus de la moitié des arbres de la forêt courante se verront attribuer un poids maximum, sans qu'une différence ne soit faite entre les données mal classées par une faible majorité d'arbres et les données mal classées par une forte majorité d'arbres. Les données bien classées par plus de la moitié des arbres quant à elles se verront attribuer des poids qui diminuent de plus en plus rapidement pour un nombre croissant de "bons" votes. À l'inverse, la logique de pondération de la fonction $W_{0.5}(c(\boldsymbol{x}, y))$ attribue des poids qui augmentent de plus en plus rapidement à mesure que les valeurs de $c(\boldsymbol{x}, y)$ diminuent. La structure de l'arbre attachera dans ce cas beaucoup plus d'importance à la prédiction des données mal classées par une forte majorité d'arbres qu'à celle des données mal classées par une faible majorité d'arbres.

Contrairement aux fonctions inverse que nous venons de présenter, les logiques de pondération définies par ces fonctions polynomiales sont plus modérées, et laisse plus de place aux compromis. Lorsque $\alpha = 1$ par exemple, les écarts de poids pour deux données de $c(\boldsymbol{x}, y)$ différents sont parfaitement homogènes. De cette façon, on ne privilègie pas plus les différenciations de pondération entre les données de $c(\boldsymbol{x}, y)$ faibles, que celles entre les données de $c(\boldsymbol{x}, y)$ élevés. Les données qui ont été bien prédites par une forte proportion d'arbres dans la forêt seront alors beaucoup moins négligées dans l'induction des arbres qu'avec les fonctions inverse.

Nous notons par la suite DRF/W_α^{pol} l'algorithme *Dynamic Random Forest* qui utilise cette fonction polynomiale comme fonction de pondération.

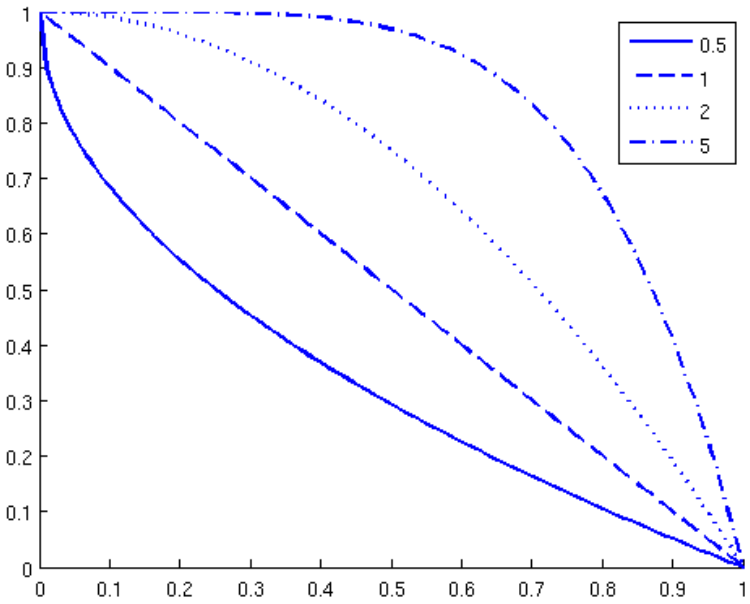

FIG. 5.2: fonction $W_\alpha(c(\boldsymbol{x}, y)) = 1 - c(\boldsymbol{x}, y)^\alpha$ pour $\alpha = \{0.5, 1, 2, 5\}$

Fonction exponentielle : $W_\alpha(c(\boldsymbol{x}, y)) = \exp(-\alpha \times c(\boldsymbol{x}, y))$

La troisième et dernière fonction de pondération que nous avons testée est une fonction exponentielle de la forme :

$$W_\alpha(c(\boldsymbol{x}, y)) = \exp(-\alpha \times c(\boldsymbol{x}, y)) \tag{5.4}$$

Cette fonction est celle qui est utilisée pour calculer le terme de pondération dans l'algorithme AdaBoost. C'est la raison pour laquelle nous avons décidé de l'inclure dans nos tests. Elle est représentée en figure 5.3, pour les 4 valeurs de α suivantes : 0.5, 2, 10 et 50. Les logiques de pondération définies ici sont proches dans le principes de celle des fonctions inverse. Le paramètre α a cependant une influence moins prononcée sur la courbe de pondération. C'est la raison pour laquelle nous avons testé ici des valeurs plus élevés pour ce paramètre. Pour $\alpha = 50$ notamment, on remarque que seules les données dont $c(\boldsymbol{x}, y)$ est inférieur à 0.1 — *i.e.* qui ont été bien classées par un maximum de 10% des arbres de la forêt — se verront attribuer un poids non nul. À l'inverse pour $\alpha = 0.5$ la pondération est plus homogène et donc plus proche dans le principe d'une fonction de pondération linéaire comme la fonction polynomiale avec $\alpha = 1$. Les écarts entre les poids seront par contre bien plus faibles dans le cas de la fonction exponentielle.

Nous notons DRF/W_α^{exp} l'algorithme *Dynamic Random Forest* qui utilise cette fonction exponentielle comme fonction de pondération.

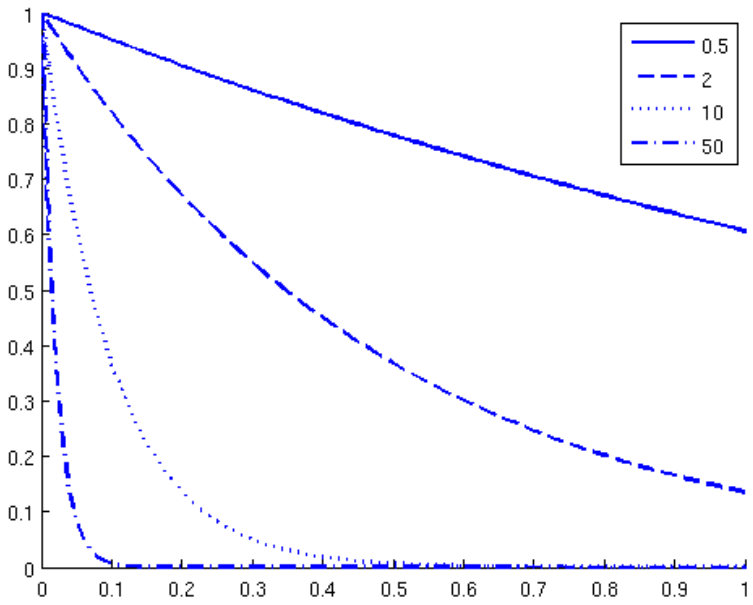

FIG. 5.3: fonction $W_\alpha(c(\boldsymbol{x}, y)) = \exp -\alpha \times c(\boldsymbol{x}, y)$ pour $\alpha = \{0.5, 2, 10, 50\}$

Nous disposons avec ces trois fonctions de pondération de plusieurs logiques de pondération différentes pour l'algorithme *Dynamic Random Forest*. Chacune de ces fonctions dépend d'un paramètre α dont la valeur permet d'affiner cette logique. Ils nous faut tester pour chacune des fonctions de pondération plusieurs valeurs de ce paramètre pour avoir une idée plus précise de la solution de pondération qui fonctionne le mieux dans le cadre de cet algorithme. Pour les expérimentations que nous présentons dans la prochaine section, nous avons décidé de tester les valeurs suivantes de α pour chacune des trois fonctions de pondération que nous venons de présenter :

- Pour DRF/W_α^{inv} nous avons choisi les valeurs : 0.5, 1, 2 et 5.
- Pour DRF/W_α^{pol} nous avons choisi les valeurs : 0.5, 1, 2 et 5.
- Pour DRF/W_α^{exp} nous avons choisi les valeurs : 0.5, 2, 10 et 50.

Pour différencier chacune de ces solutions de paramétrisation des fonctions de pondération, nous notons DRF/W_a^{inv}, DRF/W_a^{pol} et DRF/W_a^{exp} les différentes variantes des DRF qui utilisent l'une des trois fonctions de pondération avec une valeur de α égale à a.

Dans la section suivante nous présentons l'évaluation de ces 12 variantes de DRF (4 valeurs de α pour chacune des 3 fonctions de pondération) que nous avons menée en les confrontant aux deux algorithmes d'induction statique de forêts aléatoires Forest-RI et Forest-RK.

5.3 Évaluation de l'algorithme *Dynamic Random Forest*

5.3.1 Protocole expérimental

Nous présentons dans cette section le protocole expérimental mis en œuvre pour confronter chacune des 12 variantes de DRF aux algorithmes Forest-RI et Forest-RK. Nous nous sommes focalisés pour ce protocole sur deux objectifs :

- Évaluer et comparer les gains de performances obtenus avec chacune des variantes de DRF ; les comparer aux algorithmes d'induction statique.
- Observer l'évolution des propriétés de *force* et de *corrélation*, ainsi que du rapport $\frac{\bar{\rho}}{s^2}$, pour mettre en évidence l'influence du rééchantillonnage adaptatif sur ces propriétés.

Nous discutons principalement dans cette section du premier de ces objectifs. Les propriétés de *force* et *corrélation* sont discutées dans la section 5.5.

Pour ce protocole expérimental, nous avons donc construit pour chaque ensemble d'apprentissage 14 forêts au total, avec les algorithmes suivants :

- Forest-RI/$K_{\sqrt{M}}$
- Forest-RK
- DRF/W^{α}_{inv} pour $\alpha \in \{0.5, 1, 2, 5\}$
- DRF/W^{α}_{pol} pour $\alpha \in \{0.5, 1, 2, 5\}$
- DRF/W^{α}_{exp} pour $\alpha \in \{0.5, 2, 10, 50\}$

Pour chacune de ces forêts, nous relevons le taux d'erreur en test à chaque fois qu'un arbre est ajouté à l'ensemble. Nous mesurons également de la même manière la *force* et la *corrélation* de ces forêts pour déterminer quel impact cette pondération a sur ces propriétés, et plus particulièrement sur le rapport $\frac{\bar{\rho}}{s^2}$.

Nous avons repris pour cela le même schéma expérimental que précédemment, à savoir dans un premier temps diviser chacune des 20 bases de données du tableau 3.1 en 10 découpages $T_i = (T_{r_i}, T_{s_i})$, avec $i = [1..10]$. Pour chacun de ces T_i, nous avons lancé l'induction des 14 forêts aléatoires pour un nombre d'arbres de décision égal à 500. Pour chaque base de données, nous construisons donc $14 \times 10 = 140$ forêts de 500 arbres. Nous avons choisi ici de fixer le nombre d'arbres à 500 dans la mesure où nous ne connaissons pas la vitesse de convergence de l'erreur en généralisation en fonction du nombre d'arbres, pour l'algorithme de construction dynamique, ni même si convergence il y a. La démonstration de convergence du taux d'erreur en généralisation donnée par Breiman dans [Breiman 2001], que nous

Algorithme 12 Protocole Expérimental 5

Entrée : $param[3][4]$ un tableau 2D contenant les différentes valeurs du paramètre α

Sortie : $\epsilon_{RI}[10][3]$ un tableau 2D pour conserver les taux d'erreur, la force et la correlation des forêts obtenues avec Forest-RI/$K_{\sqrt{M}}$.

Sortie : $\epsilon_{RK}[10][3]$ un tableau 2D pour conserver les taux d'erreur, la force et la correlation des forêts obtenues avec Forest-RK.

Sortie : $\epsilon_{inv}[4][10][3]$ un tableau 3D pour conserver les taux d'erreur, la force et la correlation des forêts obtenues avec DRF/W_α^{inv}, pour chaque valeur de α.

Sortie : $\epsilon_{pol}[4][10][3]$ un tableau 3D pour conserver les taux d'erreur, la force et la correlation des forêts obtenues avec DRF/W_α^{pol}, pour chaque valeur de α.

Sortie : $\epsilon_{exp}[4][10][3]$ un tableau 3D pour conserver les taux d'erreur, la force et la correlation des forêts obtenues avec DRF/W_α^{exp}, pour chaque valeur de α.

Sortie : $\mathcal{M}[10][12]$ un tableau 3D pour conserver les réponses au test de McNemar.

1: **pour** $i \in [1..10]$ **faire**
2: Tirer aléatoirement sans remise $\frac{2}{3}$ des données disponibles pour former l'ensemble d'apprentissage T_{r_i}. Les données restantes forment alors l'ensemble de test T_{s_i}, le couple (T_{r_i}, T_{s_i}) est noté T_i.
3: $h_{RI} \leftarrow$ Induire une RF avec l'algorithme Forest-RI/$K_{\sqrt{M}}$ sur T_{r_i}.
4: $\epsilon_{RI}[i][1] \leftarrow$ taux d'erreur de h_{RI} sur T_{s_i}
5: $\epsilon_{RI}[i][2] \leftarrow$ *force* de h_{RI} sur T_{s_i}
6: $\epsilon_{RI}[i][3] \leftarrow$ *corrélation* de h_{RI} sur T_{s_i}
7: $h_{RK} \leftarrow$ Induire une RF avec l'algorithme Forest-RK sur T_{r_i}
8: $\epsilon_{RK}[i][1] \leftarrow$ taux d'erreur de h_{RK} sur T_{s_i}
9: $\epsilon_{RK}[i][2] \leftarrow$ *force* de h_{RK} sur T_{s_i}
10: $\epsilon_{RK}[i][3] \leftarrow$ *corrélation* de h_{RK} sur T_{s_i}
11: **pour** $j \in [1..4]$ **faire**
12: **pour** $w \in \{inv, pol, exp\}$ **faire**
13: $\alpha \leftarrow param[w][j]$
14: $H_{DRF}[w][j] \leftarrow$ Induire une RF avec l'algorithme DRF/W_w^α sur T_{r_i}.
15: $\epsilon_w[j][i][1] \leftarrow$ taux d'erreur de $H_{DRF}[w][j]$ sur T_{s_i}
16: $\epsilon_w[j][i][2] \leftarrow$ *force* de $H_{DRF}[w][j]$ sur T_{s_i}
17: $\epsilon_w[j][i][3] \leftarrow$ *corrélation* de $H_{DRF}[w][j]$ sur T_{s_i}
18: **pour** $h \in H_{DRF}$ **faire**
19: $\mathcal{M}[i][h] \leftarrow$ test de McNemar pour le couple (h_{RK}, h) sur T_{s_i}.

détaillons dans la section 2.5, est basée sur l'hypothèse que les vecteurs aléatoires θ_k qui définissent le principe de randomisation sont indépendants et identiquement distribués. Or avec le processus de construction dynamique, ces conditions ne sont plus remplies et la convergence n'est plus mathématiquement prouvée. Il est donc intéressant de laisser l'induction des forêts se faire jusqu'à un nombre important d'arbres pour observer et comparer les vitesses de convergence des différents algorithmes.

Ensuite, au cours de l'induction de ces forêts nous avons réalisé 5 mesures, et ce pour chaque itération de l'induction, c'est-à-dire pour chaque sous-forêt de l arbres, l allant de 1 à 500. Ces mesures sont les suivantes :

- les taux d'erreur en test
- les tests statistiques de McNemar pour chaque paires d'algorithmes Forest-RK - DRF
- les mesures de la *force* s
- les mesures de la *correlation* $\overline{\rho}$
- la valeur du rapport $\frac{\overline{\rho}}{s^2}$

Nous pouvons de cette façon examiner l'évolution de chacune de ces propriétés au fur et à mesure que les arbres sont ajoutés à la forêt. Nous détaillons le protocole complet de ces expérimentations dans l'algorithme 12. Il permet d'obtenir plusieurs tableaux de résultats contenant les mesures listées ci-dessus, pour chaque algorithme testé. Dans la section suivante nous présentons et discutons ces résultats.

5.3.2 Résultats et Discussions

Nous présentons tout d'abord trois tableaux de taux d'erreur, obtenus avec ces 14 algorithmes d'induction. Dans le tableau 5.1, nous présentons les taux d'erreur des algorithmes Forest-RI/$K_{\sqrt{M}}$, Forest-RK et DRF/W_α^{inv}. Dans le tableau 5.2, nous redonnons les taux d'erreur de Forest-RI/$K_{\sqrt{M}}$ et de Forest-RK, mais accompagnés cette fois de ceux de DRF/W_α^{pol}. Et dans le tableau 5.3, nous présentons de la même façon les taux de Forest-RI/$K_{\sqrt{M}}$, Forest-RK et DRF/W_α^{exp}. Nous pouvons ainsi comparer les DRF, utilisant chacune des trois fonctions de pondération, aux algorithmes d'induction statique de forêts. Comme pour nos expérimentations précédentes, ces taux d'erreur ont été moyennés sur les 10 découpages des bases de données. Nous avons également mis en évidence, en gras dans ces tableaux, les meilleurs taux d'erreur obtenus avec chacun des algorithmes de DRF. De plus, nous annotons à l'aide d'une astérisque ou d'un rond blanc les taux d'erreur des DRF qui sont en moyenne significativement meilleurs ou moins bons que ceux obtenus avec Forest-RK, selon le test de McNemar. C'est-à-dire que nous avons examiné les résultats de ce test pour chaque T_i et lorsque nous avons obtenu dans une majorité des cas des gains de performances significatifs, nous avons marqué le taux correspondant d'une astérisque, et lorsqu'à l'inverse nous avons obtenu en majorité des détériorations significatives de ces performances, nous avons marqué le taux

correspondant d'un rond blanc. Enfin, dans la dernière ligne de ces tableaux, nous indiquons le nombre de fois où l'algorithme de la colonne correspondante a permis d'obtenir un taux d'erreur moyen inférieur au taux obtenu avec chacun des deux algorithmes Forest-RI et Forest-RK, et entre parenthèses le nombre de cas où ce gain est significatif en comparaison avec Forest-RK.

Bases	F-RI	F-RK	DRF/W_α^{inv}			
			0.5	1	2	5
Diabetes	25.25	24.86	24.74	24.74	**24.63**	24.84
Digits	2.28	2.21	1.98	2.20	**2.06** *	2.14 *
DigReject	7.27	7.31	7.23	7.09 *	6.89 *	**6.77** *
Gamma	11.99	12.09	12.10	12.02	**11.92** *	12.00 *
Isolet	5.35	5.38	5.40	5.38	5.27	**5.03** *
Letter	3.94	4.02	3.98	3.89 *	3.90 *	**3.58** *
Madelon	32.78	**19.22**	19.60	19.64 ◦	20.72 ◦	23.39 ◦
Mfeat-fac	3.37	3.46	3.59	3.25	3.47	**3.18** *
Mfeat-fou	17.61	17.06	17.07	17.20	**16.86** *	17.20
Mfeat-kar	3.36	3.39	3.35	3.44	3.40	**3.09**
Mfeat-zer	21.55	21.50	21.68	21.50	21.31	**21.18**
MNIST	4.97	4.95	4.90	4.88	**4.80** *	4.81 *
Musk	2.62	2.53	2.48	2.55	2.50	**2.46**
OptDigits	1.65	1.60	1.65	1.60	1.65	**1.55**
Page-blo	2.70	2.71	2.71	**2.67**	**2.67**	2.71
Pendigits	0.91	0.92	0.92	0.87	0.89	**0.86**
Segment	2.33	2.41	2.38	2.47	**2.25**	2.46
Spambase	4.88	5.03	4.99	4.99	4.79 *	**4.78** *
Vehicle	25.60	25.50	**25.22**	25.40	25.52	25.70
Waveform	14.48	14.48	14.44	14.32	**13.97** *	13.99 *
Victoires			10 (0)	13 (2)	15 (8)	14 (9)

Tab. 5.1: Taux d'erreur moyens des forêts obtenues avec les algorithmes Forest-RI, Forest-RK et DRF/W_α^{inv}. Les astériques annotent les gains statistiquement significatifs, et les ronds blancs annotent les taux significativement moins bons en comparaison avec Forest-RK.

Ces 3 tableaux nous permettent tout d'abord de comparer les DRF utilisant chacune des trois fonctions de pondération, avec les algorithmes Forest-RI et Forest-RK. On peut ainsi dans un premier temps étudier le gain de performances qu'elles permettent toutes d'obtenir, puis dans un second temps analyser les 4 paramétrisations des fonctions de pondération et déterminer laquelle ou lesquelles sont les plus efficaces.

Tout d'abord pour l'algorithme DRF/W_α^{inv}, dont les résultats sont reportés dans le tableau 5.1, on constate de façon générale que lorsqu'un gain est obtenu celui-ci

Bases	F-RI	F-RK	DRF/W_α^{pol}			
			0.5	1	2	5
Diabetes	25.25	24.86	25.02	**24.59**	24.70	24.63
Digits	2.28	2.21	2.09 *	2.10 *	**2.04** *	2.06 *
DigReject	7.27	7.31	6.71 *	**6.56** *	6.61 *	6.80 *
Gamma	11.99	12.09	11.84 *	**11.74** *	11.85 *	11.98 *
Isolet	5.35	5.38	5.32	**4.96** *	5.02	4.98 *
Letter	3.94	4.02	3.45 *	**3.31** *	3.47 *	3.36 *
Madelon	32.78	**19.22**	23.11 °	22.66 °	23.49 °	21.09 °
Mfeat-fac	3.37	3.46	3.03	**2.89** *	3.09	3.08
Mfeat-fou	17.61	17.06	17.14	16.83 *	**16.81** *	17.02
Mfeat-kar	3.36	3.39	**3.04**	3.23	3.07	3.21
Mfeat-zer	21.55	21.50	20.90 *	21.05 *	20.74 *	**20.30** *
MNIST	4.97	4.95	4.64 *	4.61 *	**4.60** *	4.71 *
Musk	2.62	2.53	**2.15** *	2.41 *	2.13 *	2.32 *
OptDigits	1.65	1.60	**1.40** *	1.51 *	1.43 *	1.45 *
Page-blo	2.70	2.71	2.72	**2.64**	2.69	2.70
Pendigits	0.91	0.92	0.90	**0.87**	0.91	0.87
Segment	2.33	2.41	**1.83** *	2.02 *	1.96 *	2.12 *
Spambase	4.88	5.03	**3.89** *	4.04 *	4.00 *	4.31 *
Vehicle	25.60	25.50	25.14	**25.07** *	25.43	25.37
Waveform	14.48	14.48	14.40	14.35	**14.05** *	14.18 *
Victoires			16 (10)	19 (14)	19 (12)	19 (12)

TAB. 5.2: Taux d'erreur moyens des forêts obtenues avec les algorithmes Forest-RI, Forest-RK et DRF/W_α^{pol}. Les astériques annotent les gains statistiquement significatifs, et les ronds blancs annotent les taux significativement moins bons en comparaison avec Forest-RK.

n'est pas souvent significatif. La valeur $\alpha = 0.5$ par exemple n'a permis d'obtenir des gains majoritairement significatifs pour aucune des bases de données testées. La valeur $\alpha = 1$ s'est également montrée plutôt inefficace à améliorer les taux d'erreur puisqu'elle n'a permis d'obtenir de gains significatifs en moyenne que pour deux des 20 bases de données seulement. Les valeurs $\alpha = 2$ et $\alpha = 5$ quant à elles ont permis à l'algorithme d'induction dynamique d'améliorer ces taux en comparaison avec Forest-RK dans respectivement 13 et 15 cas sur 20, dont 2 fois de façon significative pour la première valeur, et 8 fois pour la deuxième.

Si l'on devait se poser la question de quelle valeur de α utiliser pour ce type de fonction de pondération dans le cadre des DRF, ces résultats nous indiqueraient qu'il faut choisir une valeur relativement élevée. On pourrait d'ailleurs vouloir tester des valeurs de α supérieures à 5 pour savoir si une amélioration des performances peut être obtenue. Cependant, ces résultats montrent de notre point de vue que cette fonction n'est globalement pas adaptée à une utilisation dans les DRF.

Bases	F-RI	F-RK	DRF/W_α^{exp}			
			0.5	2	10	50
Diabetes	25.25	24.86	25.02	24.80	**24.70**	25.01
Digits	2.28	2.21	2.29	2.16	1.97 *	2.25
DigReject	7.27	7.31	7.17	7.14 *	6.71 *	7.65
Gamma	11.99	12.09	**12.02**	12.14	**12.02**	12.13
Isolet	5.35	5.38	5.24	5.45	**5.07**	5.28
Letter	3.94	4.02	3.93	**3.81** *	3.87 *	4.27
Madelon	**18.95**	19.22	19.87 °	20.06 °	23.49 °	24.09 °
Mfeat-fac	3.37	3.46	3.03	3.14	**2.97** *	3.08
Mfeat-fou	17.61	17.06	17.14	16.83	**16.81** *	17.13
Mfeat-kar	3.36	3.39	3.42	3.16	**3.07**	3.21
Mfeat-zer	21.55	21.50	21.49	21.15 *	**20.87** *	21.30 *
MNIST	4.97	4.95	4.89	4.89	4.66 *	5.04
Musk	2.62	2.53	2.51	2.54	**2.49**	2.52
OptDigits	1.65	1.60	1.62	1.57	**1.53**	1.65
Page-blo	**2.70**	2.71	2.73	2.71	2.75	**2.70**
Pendigits	**0.91**	0.92	**0.91**	0.93	0.92	**0.91**
Segment	2.33	2.41	2.42	2.11 *	**1.96** *	2.18 *
Spambase	4.88	5.03	4.89	4.45 *	**4.00** *	4.83
Vehicle	25.60	25.50	25.62	25.57 *	**25.54**	25.89
Waveform	14.48	14.48	14.48	14.31	**14.28**	14.59
Victoires			11 (0)	13 (6)	15 (9)	9 (2)

Tab. 5.3: Taux d'erreur moyens des forêts obtenues avec les algorithmes Forest-RI, Forest-RK et DRF/W_α^{exp}. Les astériques annotent les gains statistiquement significatifs, et les ronds blancs annotent les taux significativement moins bons en comparaison avec Forest-RK.

De la même manière, nous analysons maintenant les résultats obtenus avec la fonction polynomiale en regardant le tableau 5.2. Le premier constat est cette fois-ci plus encourageant, puisque pour une grande majorité des cas, ces 4 variantes des DRF ont permis d'obtenir des gains de performances, et que ce gain est par ailleurs souvent significatif. De façon générale donc, les méthodes de DRF qui utilisent une fonction de pondération de type polynomiale permettent d'améliorer les taux d'erreur en comparaison avec un processus d'induction statique.

Ensuite, si l'on s'intéresse à la valeur du paramètre α, on constate que les différentes solutions que nous avons testées ne se démarquent pas énormément les unes des autres. En dehors d'une légère hausse de l'erreur pour l'algorithme DRF/$W_{0.5}^{pol}$, les différentes paramétrisations de α pour cette variante des DRF ne donnent pas des performances significativement différentes.

Enfin, nous pouvons étudier de plus près les résultats des DRF utilisant des

fonctions de pondération de type exponentielle à l'aide du tableau 5.3. De façon générale, ces résultats ne sont pas plus encourageants que ceux obtenus avec la fonction inverse. Les gains de performances obtenus avec ces variantes des DRF sont rarement significatifs.

On constate par ailleurs que les valeurs $\alpha = 2$ et $\alpha = 10$ fournissent des taux d'erreur en moyenne légèrement inférieurs aux deux autres paramétrisations de la fonction exponentielle. Cependant, nous nous posons à nouveau ici la question de l'efficacité globale de ce type de fonction de pondération pour les DRF. Ces résultats montrent en effet de notre point de vue qu'une fonction de pondération de cette forme est, tout comme la fonction inverse, inadaptée à une utilisation dans le cadre des DRF.

Nous venons de comparer entre elles les différentes paramétrisations des fonctions de pondération, mais il est intéressant maintenant de confronter ces fonctions les unes aux autres pour mieux percevoir dans quelle mesure l'un ou l'autre des algorithmes de DRF permet d'obtenir les meilleurs gains de performances. Nous avons déjà commencé à observer de meilleurs résultats avec la fonction polynomiale, mais pour nous en assurer il est intéressant de réunir tous ces résultats dans un même tableau. Pour cela, nous avons sélectionné pour chacune des fonctions de pondération la valeur de α qui a permis d'obtenir les meilleurs taux d'erreur en moyenne. Dans le tableau 5.4 nous reportons donc les résultats suivants :

- les taux d'erreur des forêts induites avec Forest-RI/$K_{\sqrt{M}}$
- les taux d'erreur des forêts induites avec Forest-RK
- les taux d'erreur des forêts induites avec DRF/W_5^{inv}
- les taux d'erreur des forêts induites avec DRF/$W_{0.5}^{pol}$
- les taux d'erreur des forêts induites avec DRF/W_{10}^{exp}

Nous avons mis en évidence en gras les taux d'erreur les plus faibles pour chaque base de données. Nous avons également, tout comme pour les tableaux précédents, récapitulé les nombres de bases pour lesquelles chacune des DRF a permis d'obtenir un gain de performances et les nombres de cas pour lesquels ce gain est en moyenne significatif.

Ces résultats confirment rapidement la première impression donnée par les tableaux 5.1, 5.2 et 5.3, à savoir que parmi les différentes variantes des DRF testées ici, seules celles qui utilisent une fonction de pondération polynomiale permettent d'obtenir des performances intéressantes. Les autres solutions mises en œuvre sont globalement plutôt décevantes. Ce qui fait la particularité des fonctions polynomiales que nous avons utilisées ici est que la logique de pondération qu'elles définissent est beaucoup plus modérée que les autres solutions proposées. Plus précisemment, elles négligent beaucoup moins les données qui ont été correctement prédites par une proportion importante d'arbres de la forêt. Nous l'avons expliqué dans la section 5.2.2, les fonctions inverses et exponentielles pénalisent, de façon assez radicale pour certaines valeurs de α, les données pour lesquelles le score de confiance est élevé. Nous

Bases	F-RI	F-RK	DRF/W_5^{inv}	DRF/W_1^{pol}	DRF/W_{10}^{exp}
Diabetes	25.25	24.86	24.84	**24.59**	24.70
Digits	2.28	2.21	2.14 *	2.10 *	**1.97** *
DigReject	7.27	7.31	6.77 *	**6.56** *	6.71 *
Gamma	11.99	12.09	12.00 *	**11.74** *	12.13
Isolet	5.35	5.38	5.03 *	**4.96** *	5.07
Letter	3.94	4.02	3.58 *	**3.31** *	3.87 *
Madelon	**18.95**	19.22	22.39 °	22.66 °	23.49 °
Mfeat-fac	3.37	3.46	3.18 *	**2.89** *	2.97 *
Mfeat-fou	17.61	17.06	17.20	16.83 *	**16.81** *
Mfeat-kar	3.36	3.39	3.09	3.23	**3.07**
Mfeat-zer	21.55	21.50	21.18	**21.05** *	20.87 *
MNIST	4.97	4.95	4.81 *	**4.61** *	4.66 *
Musk	2.62	2.53	2.46	**2.41** *	2.49
OptDigits	1.65	1.60	1.55	**1.51** *	1.53
Page-blo	**2.70**	2.71	2.71	**2.64**	2.75
Pendigits	**0.91**	0.92	**0.86**	0.87	0.92
Segment	2.33	2.41	2.46	2.02 *	**1.86** *
Spambase	4.88	5.03	4.78 *	4.04 *	**4.00** *
Vehicle	25.60	25.50	25.70	**25.07** *	25.54
Waveform	14.48	14.48	**13.99** *	14.35	14.28
Victoires			14 (9)	19 (14)	15 (9)

Tab. 5.4: Taux d'erreur moyens des forêts obtenues avec les algorithmes Forest-RI, Forest-RK, DRF/W_5^{inv}, DRF/W_1^{pol} et DRF/W_{10}^{exp}. Les astériques annotent les gains statistiquement significatifs, et les ronds blancs annotent les taux significativement moins bons en comparaison avec Forest-RK.

pensons que la conséquence de cela est que les arbres, tout en compensant les erreurs des arbres précédents, introduisent de nouvelles erreurs en prédiction, en pondérant trop radicalement les données dont le score de confiance est plus élevé.

Il serait intéressant alors d'étudier plus en détails ce phénomène, en examinant notamment les données mal prédites tout au long du processus d'induction dynamique. De cette façon on pourrait percevoir si de nouvelles erreurs sont introduites par les nouveaux arbres, et si ces erreurs concernent effectivement les données qui ont été préalablement bien prédites par la forêt.

Nous observons également un autre résultat surprenant à partir de ces trois tableaux. Une des 20 bases de données présente des résultats particulièrement atypiques ; il s'agit de la base Madelon. Pour cette base, aucune des 12 variantes de DRF n'a permis d'obtenir un gain de performances. À l'inverse elles ont toutes détériorées ces performances en comparaison avec Forest-RI et Forest-RK, parfois même de façon importante. Ce sont de plus deux des variantes les moins

performantes sur l'ensemble des autres bases, à savoir DRF/$W_{0.5}^{inv}$ et DRF/$W_{0.5}^{exp}$ qui ont permis en moyenne d'obtenir les meilleurs taux parmi les DRF. Pour les 10 autres variantes, la pondération adaptative des données a provoqué une détérioration conséquente de ces taux d'erreur. Nous ne pouvons pas expliquer ce comportement atypique avec ces résultats seuls, mais nous notons que Madelon est une des bases pour lesquelles nous avions déjà observé un comportement atypique de l'algorithme Forest-RI, lors de notre étude sur le paramètre K (*cf.* chapitre 3). La particularité de cette base est que son espace de description est composé de caractéristiques très peu discriminantes. Nous pensons qu'il serait intéressant de poursuivre l'analyse du comportement des DRF en étudiant de plus près le lien entre l'évolution de l'erreur et la qualité des caractéristiques, comme nous l'avons fait à propos de l'étude sur K (*cf.* chapitre 3).

En conclusion, nous pouvons dire que notre approche d'induction dynamique a amélioré avec succés les performances des forêts aléatoires en comparaison avec les procédures d'induction statique traditionnelles. Pour une grande majorité des bases que nous avons testées, nous avons obtenu un gain de performances significatif avec au moins une des fonctions de pondération que nous proposons. On constate également avec ces résultats que la logique de pondération qui est la plus performante est celle qui ne néglige pas exagérément les données particulièrement bien classées par la sous-forêt courante. Cependant, nous pensons qu'il serait intéressant d'approfondir l'analyse du comportement des DRF en fonction de ces fonctions de pondération. Nous avons notamment observé pour une des bases de données un comportement atypique qui nous laisse penser que la nature de l'espace de description pourrait avoir une influence sur les performances des DRF.

5.4 Étude de l'Évolution des Performances en Généralisation

Comme nous l'avons expliqué à plusieurs reprises dans ce chapitre, l'objectif de la mise en place d'un processus d'induction dynamique de forêts aléatoires est de progressivement améliorer les performances, en tentant d'optimiser un score de confiance des prédiction des forêts, à mesure que les arbres sont ajoutés à l'ensemble. Il est intéressant donc d'étudier de plus près dans quelle mesure les performances sont améliorées et à quelle vitesse elles convergent pour un nombre croissant d'arbres. Pour cela nous avons sélectionné pour chacune des fonctions de pondération testées, la valeur de α la plus performante selon les résultats que nous venons de présenter dans la section précédente. Sur chaque diagramme nous traçons donc 4 courbes de taux d'erreur :

- la courbe des taux d'erreur des forêts induites avec Forest-RK
- la courbe des taux d'erreur des forêts induites avec DRF/W_5^{inv}
- la courbe des taux d'erreur des forêts induites avec DRF/W_1^{pol}

- la courbe des taux d'erreur des forêts induites avec DRF/W_{10}^{exp}

Nous présentons deux ensembles de courbes d'évolution des taux d'erreur en fonction du nombre d'arbres ajoutés dans la forêt, pour ces 4 algorithmes :

- la figure 5.4 présente l'évolution des taux d'erreur pour les sous-forêts de taille l allant de 20 à 500
- la figure 5.5 présente l'évolution de ces taux pour les 100 premières sous-forêts, *i.e.* de taille l allant de 1 à 100.

Par souci de clarté nous présentons ces courbes pour seulement 6 des 20 bases de données, mais les conclusions que nous en tirons sont généralisables à l'ensemble des bases de données que nous avons testées. Les courbes des Forest-RK sont représentées en noir sur chacune des figures. Les taux de DRF/W_1^{inv} correspondent aux courbes rouges ; ceux de DRF/W_5^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

Ce qu'il est intéressant d'observer avec ces courbes, c'est l'évolution des taux d'erreur des processus de DRF en comparaison avec l'évolution des taux obtenus avec Forest-RK. On constate tout d'abord, à partir de la figure 5.4, que cette évolution est globalement favorable aux DRF puisque leurs courbes de taux descendent clairement en dessous de celle de Forest-RK pour la plupart des bases. C'est le cas pour 14 des 20 bases avec DRF/W_1^{pol}, pour 10 d'entre elles avec DRF/W_5^{inv}, et c'est par contre moins souvent le cas pour DRF/W_{10}^{exp}, *i.e.* pour 8 des 20 bases.

On constate ensuite que la convergence de ces taux est atteinte pour un nombre d'arbres à peu près équivalent à celui des Forest-RK, c'est-à-dire environ 200 arbres. De fait, la diminution des taux d'erreur est bien plus rapide avec les DRF qu'avec Forest-RK pour un nombre croissant d'arbres dans les forêts. C'est un constat que l'on retrouve sur la figure 5.5, qui se focalise plus particulièrement sur les 100 premières itérations. On observe sur ces diagrammes qu'au début de l'induction les taux d'erreur sont très proches pour l'ensemble des 4 algorithmes. Puis, assez rapidement à mesure que les arbres sont ajoutés à la forêt, les taux d'erreur diminuent plus conséquemment avec les DRF qu'avec Forest-RK. Dans certain cas, comme par exemple avec la base de données Musk, les taux d'erreurs sont même plus élevés avec les DRF au début de l'induction, pour diminuer ensuite et descendre assez rapidement en dessous des taux de Forest-RK. Là encore cependant nous notons que c'est un phénomène moins prononcé pour DRF/W_{10}^{exp}. Ses taux sont souvent les plus élevés des trois algorithmes de DRF au début du processus d'induction, et ils ne diminuent pas de façon aussi prononcée à mesure que les arbres sont ajoutés. Quant aux deux autres algorithmes de DRF, on constate que les taux de DRF/W_5^{inv} sont souvent les plus faibles au tout début de l'induction, mais qu'ils diminuent ensuite généralement moins rapidement que ceux de DRF/W_1^{pol}. L'objectif des DRF qui est de progressivement compenser les erreurs des arbres déjà induits, tout en évitant de générer de nouvelles erreurs de prédiction, semble mieux atteint avec la fonction de pondération polynomiale qu'avec les deux autres types de fonctions. Cela confirme les remarques de la section précédente,

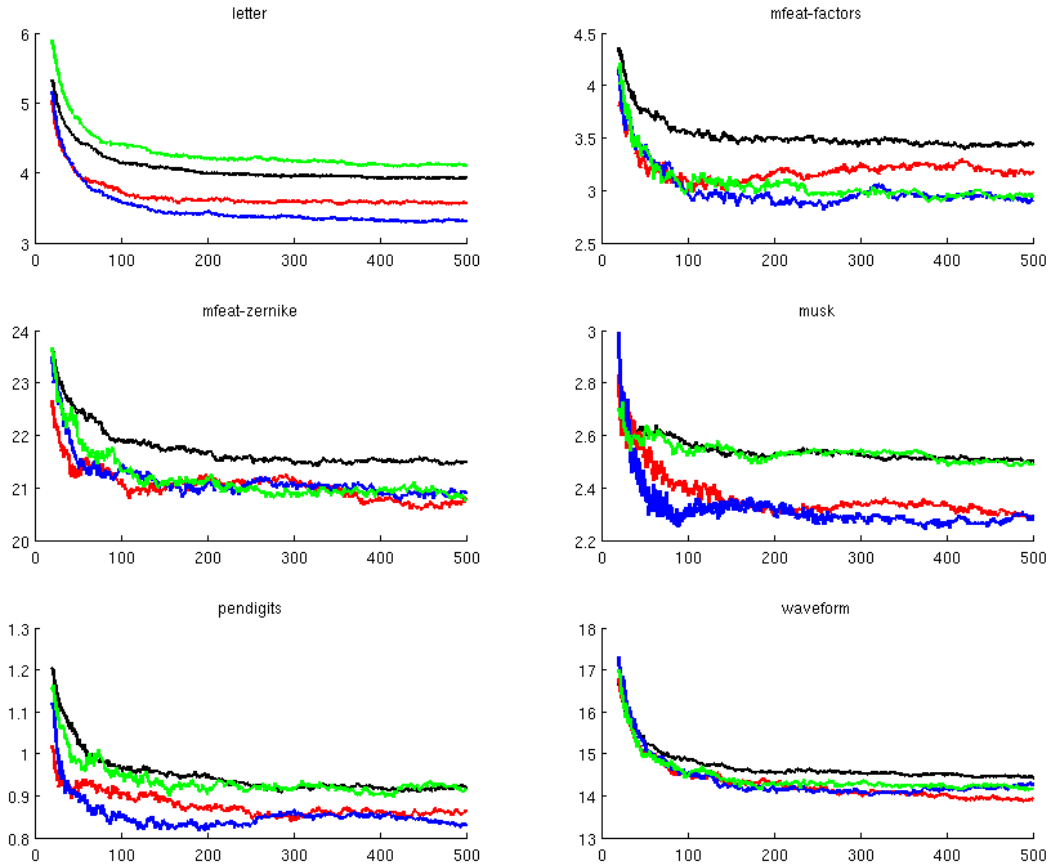

FIG. 5.4: Évolution des taux d'erreur en test pour les sous-forêts de l arbres, $l \in [20..500]$, pour 6 bases de données. Les courbes de Forest-RK sont représentées en noir. Les taux de DRF/W_1^{inv} correspondent aux courbes rouges ; ceux de DRF/W_5^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

selon lesquelles il est nécessaire pour obtenir cette évolution de faire attention à ne pas trop négliger les données biens classées par la plupart des arbres de la forêt, sous peine d'introduire trop de nouvelles erreurs de prédiction sur ces données.

De façon plus générale, ces résultats illustrent de notre point de vue le fait que la pondération adaptative agit rapidement sur les performances en généralisation des forêts. Il ne faut pas plus de 200 arbres en moyenne pour atteindre un gain de performances maximal avec les DRF. Ajouter ensuite plus d'arbres à la forêt ne permet pas d'obtenir de meilleures performances avec ces processus d'induction dynamique. Bien que la condition d'indépendance et de distribution identique pour tous les θ_k soit nécessaire pour la démonstration de la convergence (notamment pour l'application du théorème de la loi forte des grands nombres, *cf.* section 2.5), ces résultats montrent que la convergence de l'erreur en généralisation est tout de même atteinte de la même façon qu'avec les algorithmes d'induction statique, et

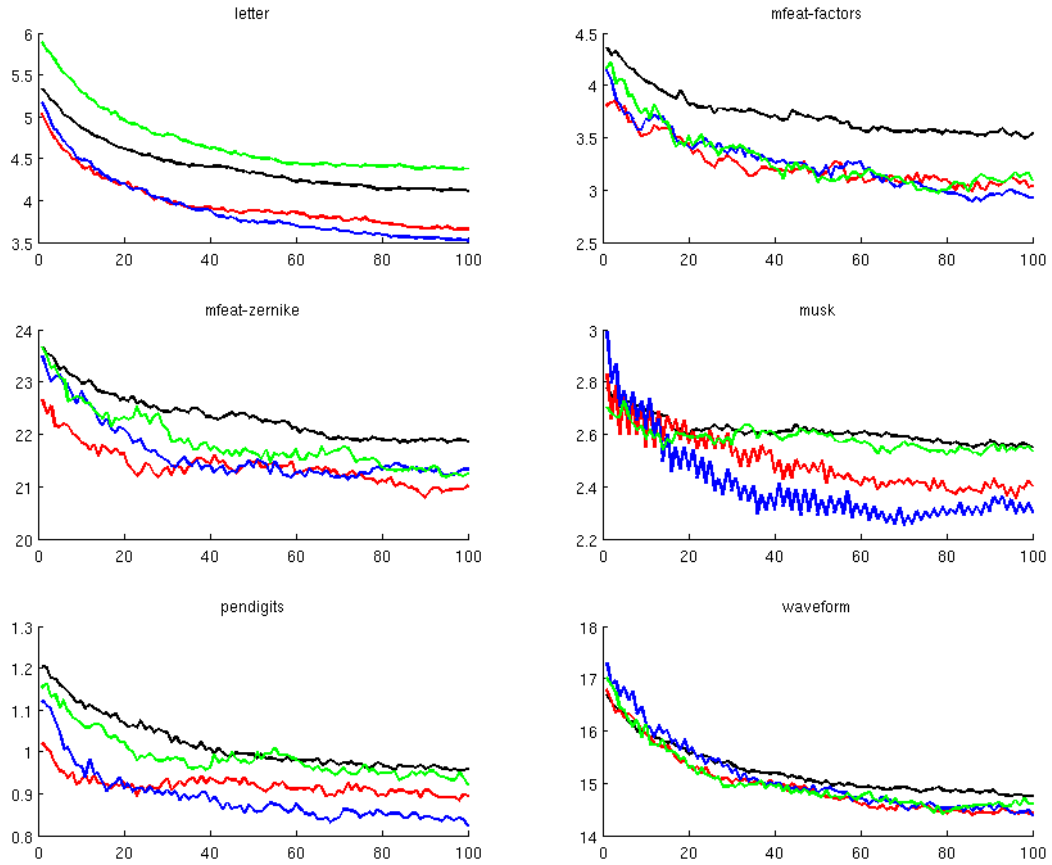

FIG. 5.5: Évolution des taux d'erreur en test pour les sous-forêts de l arbres, $l \in [1..100]$, pour 6 bases de données. Les courbes de Forest-RK sont représentées en noir. Les taux de DRF/W_1^{inv} correspondent aux courbes rouges ; ceux de DRF/W_5^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

qu'au delà d'un certain nombre d'arbres, en ajouter à l'ensemble ne permet pas d'obtenir de meilleures performances en généralisation.

La condition de convergence de l'erreur en généralisation est importante, notamment pour la mise en place d'un critère d'arrêt pour l'induction dynamique. Comme nous l'avons expliqué dans le chapitre 4, l'intérêt de ce type d'induction, outre l'amélioration des performances, est de pouvoir réduire l'ensemble des arbres à un sous-ensemble plus performant et plus rapide en phase d'apprentissage et de prédiction. Le but de l'induction dynamique est finalement de parvenir plus directement à ce sous-ensemble. Or, sans cette convergence de l'erreur en généralisation pour un nombre croissant d'arbres, le risque est grand de ne pas parvenir à stopper le processus d'induction au moment adéquat. Ces résultats mettent en exergue la possibilité de la mise en place d'un tel critère. La mise en place d'un critère d'arrêt n'a malheureusement pas pu être étudiée dans le cadre de ces travaux, faute de

temps. C'est néanmoins une des principales perspectives à court terme.

5.5 Force et Corrélation des *Dynamic Random Forest*

Le protocole que nous avons présenté dans la section 5.3.1 nous a permis d'évaluer nos algorithmes de DRF, en les confrontant aux algorithmes d'induction statique Forest-RI et Forest-RK, mais nous a également permis de mesurer les propriétés de *force* et de *corrélation* des forêts obtenues. Dans cette section nous étudions plus en détails le comportement des DRF vis à vis de ces deux propriétés, et plus particulièrement à travers le rapport $\frac{\overline{\rho}}{s^2}$ dont nous avons déjà beaucoup parlé dans la section 2.5 et dans le chapitre 4. En choisissant d'optimiser la proportion des arbres qui ont voté pour la bonne classe dans l'induction dynamique, le but était dans un premier temps d'augmenter la propriété de *force*, et dans un second temps de minimiser la propriété de *correlation* (*cf.* section 5.2). Pour vérifier si la pondération adaptative a eu les effets escomptés, nous traçons trois types de courbes :

- les courbes de l'évolution de la *force* en fonction du nombre d'arbres ajoutés à la forêt.
- les courbes de l'évolution de la *correlation* en fonction du nombre d'arbres ajoutés à la forêt.
- les courbes de l'évolution du rapport $\frac{\overline{\rho}}{s^2}$ en fonction du nombre d'arbres ajoutés à la forêt.

Ces courbes sont présentées dans les figures 5.6, 5.7 et 5.8. Toujours dans le souci de ne pas surcharger les figures, nous avons ici aussi sélectionné pour chacune d'elles 6 des 20 bases de données testées. Là encore, les conclusions que nous tirons de ces résultats peuvent être étendues à l'ensemble de ces 20 bases. Les courbes des Forest-RK sont représentées en noir sur chacune des figures. Les taux de DRF/W_5^{inv} correspondent aux courbes rouges ; ceux de DRF/W_1^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

Tout d'abord, si l'on observe sur la figure 5.6 l'évolution de la *force* en fonction du nombre d'arbres dans les sous-forêts, on constate que toutes les variantes de DRF ne permettent pas toujours d'augmenter les mesures de *force* en comparaison avec l'algorithme Forest-RK. Ce premier objectif n'est atteint qu'avec DRF/W_1^{pol} pour une grande majorité des bases de données (toutes sauf 2 bases : Diabetes et Madelon). Avec cette variante, qui nous le rappelons est la plus performante des trois, on constate que la *force* suit une évolution cohérente avec l'évolution du taux d'erreur. Les sous-forêts les plus petites présentent dans un premier temps des mesures de *force* inférieures à celles obtenues avec Forest-RK, mais celles-ci augmentent ensuite bien plus rapidement au fur et à mesure que les arbres sont ajoutés à l'ensemble, jusqu'à dépasser la courbe de Forest-RK. Ce phénomène n'est par contre pas toujours obtenu avec les deux autres algorithmes de DRF. La *force* pour ces deux algorithmes a tendance à converger pour un nombre croissant d'arbres, vers une valeur avoisinant celle de Forest-RK, et parfois très inférieure

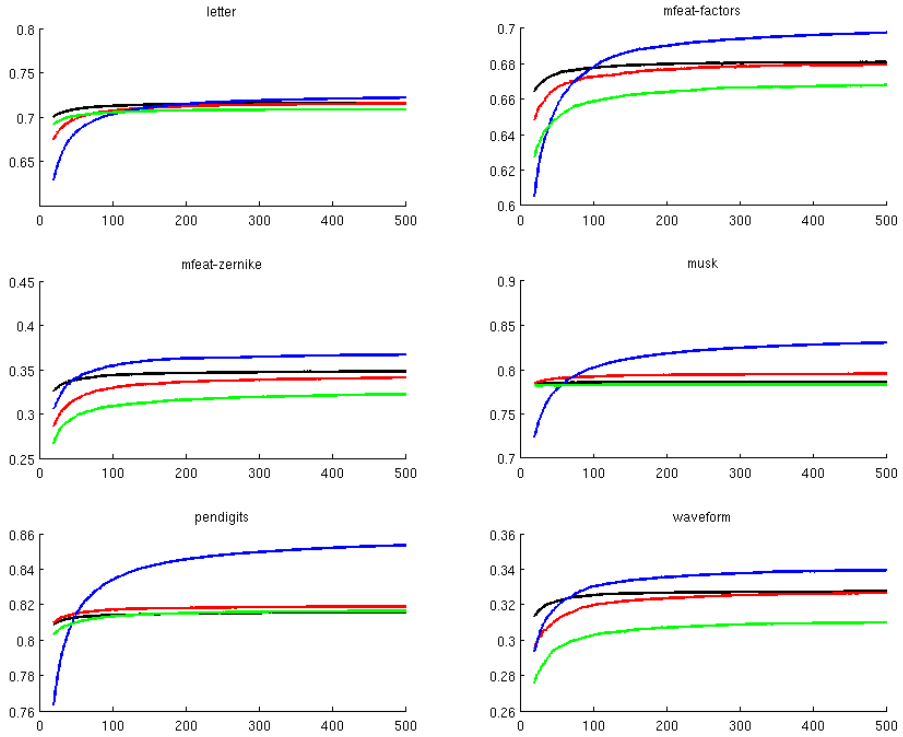

FIG. 5.6: Évolution de la *force* en fonction du nombre d'arbres dans les forêts. Les courbes de Forest-RK sont représentées en noir. Les taux de DRF/W_1^{inv} correspondent aux courbes rouges ; ceux de DRF/W_5^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

(dans 3 cas sur 20 pour DRF/W_{inv}^5, et dans 1 cas pour DRF/W_{exp}^{10}).

Ensuite, nous pouvons analyser les mesures de *corrélation* obtenues au cours du processus d'induction. On se rend compte ici encore que les meilleurs résultats obtenus avec les DRF l'ont été avec DRF/W_1^{pol}, et ce pour la totalité des bases de données. Si le processus d'induction dynamique réussit globalement à diminuer la *corrélation* dans les forêts, comme l'illustrent les courbes de la figure 5.7, c'est toujours la variante DRF/W_1^{pol} qui y parvient le mieux.

Logiquement donc, lorsque l'on regarde la figure 5.8, on se rend compte que DRF/W_{pol}^1 est l'algorithme d'induction dynamique qui améliore le mieux le rapport $\frac{\bar{\rho}}{s^2}$, parfois même de façon très importante. Les différences de valeurs pour ce rapport sont pour certaines bases très grandes entre cette variante et les deux autres. Ces écarts semblent d'ailleurs plus importants ici qu'il ne le sont pour les performances correspondantes. Par exemple, pour la base Mfeat-Zernike la différence entre la courbe de DRF/W_{pol}^1 et les courbes de DRF/W_{inv}^5 et DRF/W_{pol}^{10} sur la figure 5.8 est grande, alors que les différences de performances sur la figure 5.4 sont presque inexistantes. C'est un constat que l'on retrouve par ailleurs sur plusieurs des 20 bases de données et Mfeat-Zernike ne semble pas être un cas

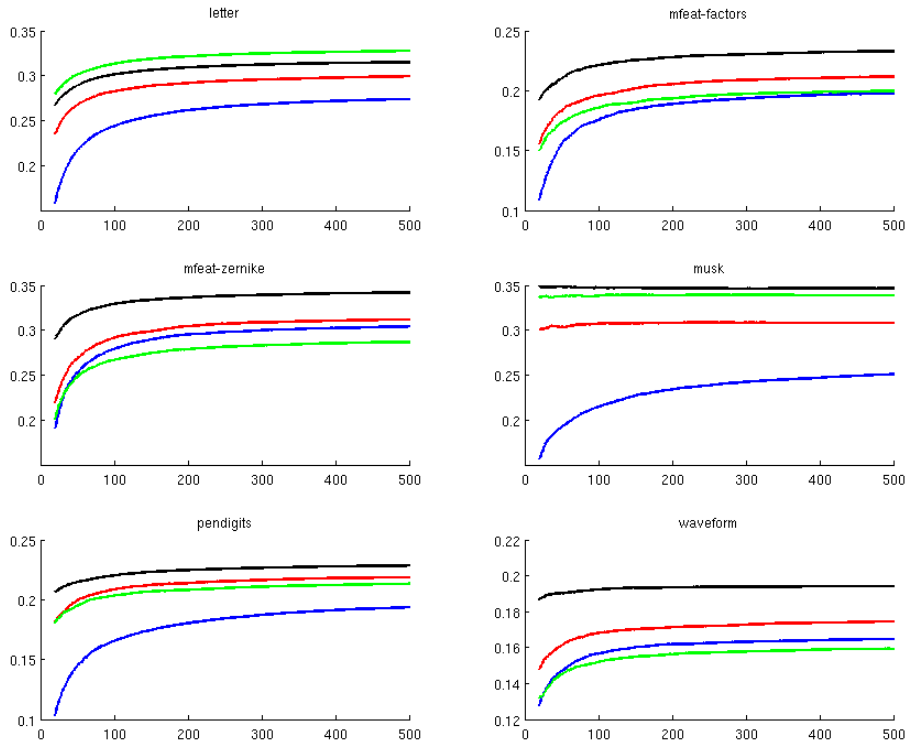

FIG. 5.7: Évolution de la *corrélation* en fonction du nombre d'arbres dans les forêts. Les courbes de Forest-RK sont représentées en noir. Les taux de DRF/W_1^{inv} correspondent aux courbes rouges ; ceux de DRF/W_5^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

atypique. Cela indique qu'il est possible d'obtenir une amélioration conséquente du rapport $\frac{\overline{\rho}}{s^2}$ sans pour autant qu'il en découle un gain de performances. Par conséquent, si ce rapport $\frac{\overline{\rho}}{s^2}$ est intéressant en ce sens qu'il présente une corrélation globale avec les performances en généralisation, il ne suffit pas à expliquer les variations de performances des DRF. L'effet du rééchantillonnage adaptatif sur les performances n'est pas uniquement explicable à l'aide des propriétés de *force* et de *corrélation*.

En conclusion donc, ces résultats mettent en évidence que la pondération qui permet le mieux d'optimiser les propriétés de *force* et de *corrélation* est la pondération qui attribue des poids linéairement décroissants pour des $c(\boldsymbol{x}, y)$ croissants. En d'autre termes, par rapport aux deux autres variantes de DRF, c'est la logique de pondération qui établit le moins de différences entre les poids des données bien classées et les données mal classées. En outre, les données bien classées sont beaucoup moins négligées dans l'induction du prochain arbre, lorsque la fonction polynomiale de degré 1 est utilisée. Cependant si cette fonction permet d'obtenir de meilleurs résultats que les autres algorithmes de DRF sur la minimisation du rapport $\frac{\overline{\rho}}{s^2}$, les

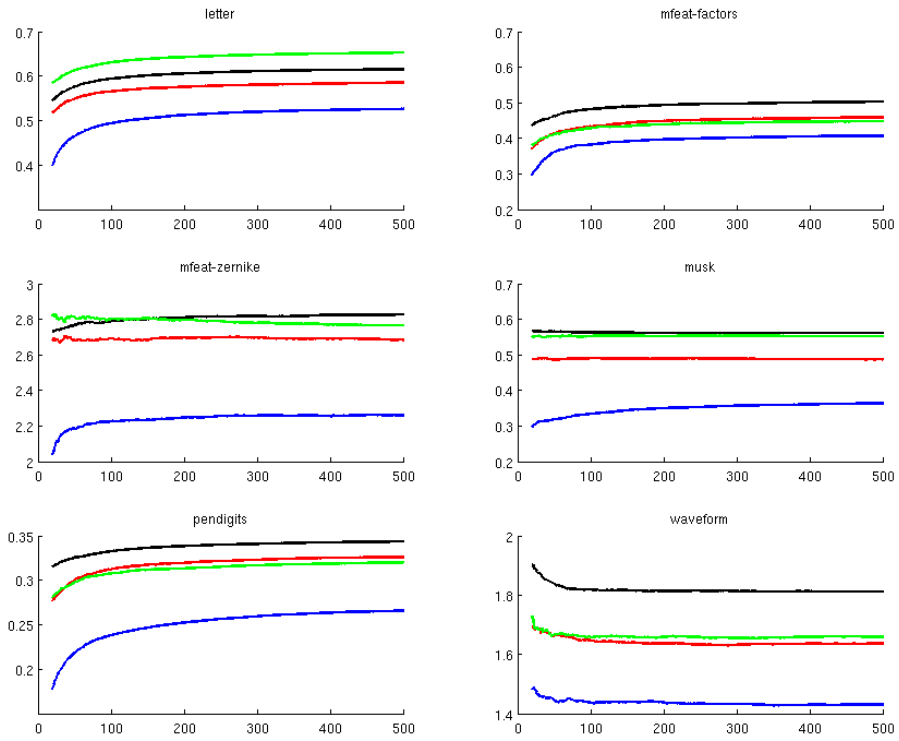

FIG. 5.8: Évolution du rapport $\frac{\overline{\rho}}{s^2}$ en fonction du nombre d'arbres dans les forêts. Les courbes de Forest-RK sont représentées en noir. Les taux de DRF/W_1^{inv} correspondent aux courbes rouges ; ceux de DRF/W_5^{pol} aux courbes bleues ; et ceux de DRF/W_{10}^{exp} aux courbes vertes.

gains de performances correspondants ne semblent pas toujours suivre. Les écarts des taux d'erreur semblent notamment parfois bien faibles proportionnellement aux gains obtenus sur $\frac{\overline{\rho}}{s^2}$.

Il serait alors intéressant d'approfondir l'analyse de l'évolution de ce rapport et de son impact sur l'erreur en généralisation pour tenter d'identifier les situations pour lesquelles le gain de performances obtenus n'est pas aussi importants que le gain de valeurs de $\frac{\overline{\rho}}{s^2}$. Il nous paraît également évident maintenant que les propriétés de *force* et de *corrélation*, si elle fournissent une borne de l'erreur en généralisation des forêts et si elles expliquent en partie les variations de taux d'erreur d'une forêt à une autre, ne suffisent pas à contrôler ces variations dans le cadre d'un processus d'induction dynamique. Il serait alors intéressant de poursuivre ce travail dans l'optique d'identifier et d'exploiter d'autres propriétés des forêts qui pourraient compléter le rôle de la *force* et de la *corrélation*.

5.6 Conclusion

Dans ce chapitre, nous avons présenté un algorithme original d'induction de forêts aléatoires appelé *Dynamic Random Forest* (DRF). Cet algorithme utilise une procédure séquentielle et dynamique pour construire un ensemble d'arbres aléatoires. L'aspect dynamique de cette procédure est fortement inspiré du procédé de rééchantillonnage adaptatif du principe de Boosting. Dans ce principe chaque classifieur élémentaire de l'ensemble est construit en privilégiant la bonne classification des données précédemment mal classées. L'algorithme de DRF reprend en partie cette idée en la combinant aux procédés de randomisation qui sont utilisés dans les algorithmes classiques d'induction de forêts aléatoires.

L'idée principale des DRF est de guider l'induction de chaque arbre à l'aide des prédictions de l'ensemble des arbres déjà induits, de sorte qu'une fois ajouté à l'ensemble celui-ci compense au mieux leurs erreurs. Pour ce faire, à l'instar des principes de Boosting, nous utilisons une pondération des données d'apprentissage de façon à ce que les données difficiles à classer soient privilégiées dans l'induction de l'arbre, et donc, se voient affecter un poids plus important que les données mieux prédites. Pour le calcul des poids nous utilisons alors une mesure de fiabilité des prédictions de la forêt qui s'appuie sur le nombre de votes attribués à la vraie classe des données $c(\boldsymbol{x}, y)$.

Nous avons proposé dans ce chapitre plusieurs variantes de cet algorithme DRF, qui utilisent chacune une fonction de pondération différente. Ces fonctions de pondération ont un point en commun : elles sont inversement proportionnelles à $c(\boldsymbol{x}, y)$ sur l'intervalle $[0, 1]$. Elles fournissent cependant chacune une logique de pondération différente, permettant ainsi de diminuer ou d'accentuer les écarts entre les poids des données. Nous avons ensuite évalué ces variantes en les confrontant aux algorithmes d'induction classique Forest-RI/$K_{\sqrt{M}}$ et Forest-RK, que nous avons présentées en détails dans le chapitre 3. Nous avons montré tout d'abord que pour chaque base, au moins une de ces variantes permet d'induire des forêts significativement plus performantes que celles induites avec les deux algorithmes d'induction statique. Ensuite, avec les résultats de chacune des fonctions de pondération, nous avons mis en évidence que le bon fonctionnement d'un tel processus d'induction dynamique nécessite de ne pas trop pénaliser les données bien classées pour l'induction du prochain arbre. La fonction de pondération qui réussit le mieux à améliorer les performances des forêts en comparaison avec les algorithmes d'induction statiques par ailleurs, est une fonction polynomiale, décroissante pour des valeurs de $c(\boldsymbol{x}, y)$ croissantes.

Nous avons ensuite examiné plus en détails l'évolution de l'erreur en généralisation en fonction du nombres d'arbres ajoutés au cours de l'induction. Plus particulièrement nous nous sommes intéressé à la convergence de l'erreur en généralisation. Nous avons montré dans un premier temps que malgré le fait que la preuve mathématique de la convergence des taux d'erreur donnée par Breiman

dans [Breiman 2001] (*cf.* section 2.5) ne s'applique pas aux algorithmes de DRF, ces taux d'erreur convergent tout de même pour un nombre croissant d'arbres, et ce de façon très similaire aux algorithmes d'induction statique. Nous avons montré également que la pondération adaptative permet assez efficacement et rapidement d'améliorer les performances des forêts dans les premières itérations du processus. Ces résultats sont importants car, sans cette convergence, il serait beaucoup plus compliqué de mettre en place un critère d'arrêt pour l'algorithme d'induction dynamique. L'intérêt d'un tel algorithme est de pouvoir induire des forêts plus performantes, plus efficacement et plus directement. Une des premières perspectives à sa conception est de mettre en place un processus d'arrêt qui stoppe l'induction lorsque le point de convergence est atteint. Nos résultats montrent que la mise en place d'un tel critère d'arrêt est possible.

Enfin, nous avons appronfondi l'analyse des performances des DRF en étudiant l'évolution des propriétés de *force* et de *corrélation*, ainsi que celle du rapport $\frac{\bar{\rho}}{s^2}$, au cours de l'induction. Nous avons montré alors que la pondération basée sur une fonction polynomiale d'ordre 1 est celle qui permet le mieux de remplir son rôle, en maximisant la *force* à mesure que les arbres sont ajoutés, en diminuant la *corrélation* globale en comparaison avec une induction statique, et en diminuant par conséquent les valeurs de $\frac{\bar{\rho}}{s^2}$.

Cependant cette étude a mis en évidence que les écarts de observés sur le rapport $\frac{\bar{\rho}}{s^2}$ ne sont pas toujours corrélés avec les écarts observés sur l'erreur en généralisation. Dans certains cas les valeurs de $\frac{\bar{\rho}}{s^2}$ sont considérablement diminuées sans pour autant qu'un gain de performances significatif ne soit observé. Cela nous indique que les propriétés de *force* et de *corrélation* ne suffisent pas à l'explication des variations de performances en généralisation, et ne permettent pas à elles seules de bien les contrôler. Aussi, nous pensons qu'il serait intéressant de poursuivre l'analyse des comportements des DRF pour déterminer dans quelle mesure et dans quels cas ce contrôle n'est pas établi. De plus, puisqu'il semble évident à la lumière de nos résultats que les performances en généralisation des forêts ne dépendent pas uniquement des fluctuations de $\frac{\bar{\rho}}{s^2}$, nous pensons qu'il est nécessaire de poursuivre l'étude d'un critère de pondération pour l'induction dynamique, qui ne tiendrait pas uniquement compte de ces propriétés.

Nous avons fait le choix dans ces travaux d'utiliser une pondération des données pour guider l'apprentissage des arbres de la forêt. Il y a cependant plusieurs autres pistes possibles pour cela ([Sicard 2006, Sicard 2008, Elgawi 2008, Saffari 2009]). Par exemple, nous avons amorcé au cours de ces travaux sur les DRF la conception d'un deuxième principe d'induction dynamique qui s'appuie cette fois sur une pondération des caractéristiques, plutôt qu'une pondération des données. Cette pondération prend en compte la capacité des caractéristiques à conduire, au sein des arbres, vers une bonne ou une mauvaise classification des données, lorsqu'elles sont choisies pour la règle de partitionnement. Si une caractéristique est souvent utilisée pour la prédiction d'une donnée, et que cette prédiction est le plus souvent bonne, il est

intéressant de chercher à favoriser cette caractéristique pour effectuer le partitionnement chaque fois que la donnée est utilisée au cours de l'induction des prochains arbres. C'est sur la base de cette idée que nous pensons intéressant de mettre en place une pondération de ces caractéristiques, qui est prise en compte pour le choix des règles de partitionnement.

En combinant un procédé comme celui-ci à un rééchantillonnage adaptif tel que nous le mettons en place dans les *Dynamic Random Forest*, on pourrait espérer mieux maîtriser la diversité dans l'ensemble, tout en renforçant encore plus la confiance des prédictions fournit par la forêt.

Conclusion Générale

Nous nous sommes intéressé dans ces travaux de thèse aux méthodes d'Ensembles de Classifieurs (EoC), et plus particulièrement aux forêts aléatoires. Ce qui rend les EoC populaires en comparaison avec d'autres approches de combinaison de classifieurs, est qu'il existe un grand nombre de méthodes d'induction d'EoC qui permettent de construire des ensembles de classifieurs divers. Bagging, Boosting, ou encore Random Subspaces sont des exemple de principes d'induction d'EoC qui peuvent créer de la diversité dans les ensembles, pour des types de classifieurs élémentaires quelconques. Or, sur ce point, les méthodes de forêts aléatoires font figures d'exception puisqu'elles présentent la particularité d'utiliser exclusivement des classifieurs élémentaires de types arbres de décision. La raison principale est que ces classifieurs sont particulièrement adaptés à une utilisation au sein des EoC, et ce en raison de leur instabilité. Par conséquent, les procédés pour générer de la diversité dans ces ensembles sont parfois spécifiques à l'induction automatique d'arbres de décision.

La principale difficulté que présente les différents principes d'induction d'EoC est que leurs hyperparamètres, qui permettent la plupart du temps de contrôler la diversité dans les ensembles, sont souvent difficiles à régler. Les forêts aléatoires n'échappent pas à cette règle. Par exemple, s'agissant de l'algorithme de référence pour l'induction de forêts aléatoires Forest-RI, deux principaux paramètres permettent de créer de la diversité :

- le nombre de caractéristiques sélectionnées aléatoirement à chaque nœud des arbres
- le nombre d'arbres induits dans la forêt

Or, peu d'étude de la littérature se sont à ce jour intéressés à ces deux paramètres et à leur influence sur les performances, de sorte qu'il est difficile aujourd'hui de les règler *a priori* face à un problème donné. En outre, les mécanismes responsables de cette influence de ces deux paramètres sur l'erreur en généralisation sont très peu connus de la communauté scientifique et beaucoup de questions relatives au fonctionnement des forêts aléatoires restent à ce jour sans réponse.

Nous avons proposé avec ces travaux de thèse une étude de chacun de ces paramètres en suivant la même démarche globale. Nous avons d'abord étudié de façon rigoureuse leur influence sur l'erreur en généralisation des forêts. Nous avons ensuite apporté des éléments d'explication de cette influence. Et enfin nous avons exploité ces éléments pour mettre en place un algorithme d'induction de forêts aléatoires qui apporte des réponses aux problèmes posés par l'un et par l'autre de ces deux paramètres.

Dans un premier temps, nous nous sommes intéressés au principe de Random Feature Selection, et plus particulièrement à son paramètre, noté K, désignant le nombre de caractéristiques sélectionnées aléatoirement à chaque nœud des arbres. Jusqu'ici, des solutions arbitraires étaient traditionnellement utilisées dans la littérature pour en fixer la valeur. Nous avons tout d'abord montré que, si l'une de ces valeurs permet d'obtenir des forêts performantes dans la plupart des

cas, il existe des problèmes pour lesquels elle est, comme les autres valeurs par défaut, clairement sous-optimale. Nous avons ensuite expliqué ce phénomène en démontrant que la "qualité" de l'espace de description du problème influence le comportement des forêts vis à vis de ce paramètre. Pour pallier ce problème, nous avons alors proposé un algorithme original d'induction de forêts aléatoires, que nous avons appelé Forest-RK et qui s'appuie sur ces conclusions en basant les choix de la valeur de K sur cette "qualité" des caractéristiques. Nos résultats ont démontré que ce procédé permet pour l'ensemble des bases de données testées d'atteindre les meilleurs taux d'erreur obtenus avec Forest-RI, toutes valeurs de K confondues. La conclusion principale de cette première contribution est que pour l'induction de forêts aléatoires, l'utilisation de Forest-RK est préférable à celle de Forest-RI, paramétré avec l'une des valeurs par défaut.

Notre deuxième contribution a porté sur l'influence du nombre d'arbres dans les forêts sur les performances en généralisation. Nous avons montré dans un premier temps qu'une proportion, dans certains cas importante, des arbres induits au cours d'un processus d'induction statique de forêts aléatoires ne permet pas de diminuer l'erreur en généralisation, et l'augmente même parfois. Nous avons ensuite expliqué ce phénomène à travers deux propriétés importantes des forêts : la *force* et la *corrélation*. Nous avons montré notamment que les forêts les plus performantes sont globalement celles qui présentent une *force* élevée, et une *corrélation* faible. Avec ces résultats, nous avons mis en évidence les principaux inconvénients des processus d'induction statique de forêts aléatoires tels qu'ils sont mis en œuvre dans les algorithmes Forest-RI et Forest-RK par exemple. La principale conclusion de cela est qu'il semble plus pertinent dès lors de procéder par induction dynamique pour l'apprentissage d'une forêt, en guidant l'ajout des arbres pour qu'ils complètent au mieux les arbres déjà induits. Nous avons également mis en évidence l'intérêt de chercher avec cette induction dynamique à augmenter la *force* tout en diminuant la *corrélation*, au fur et à mesure que les arbres sont ajoutés à l'ensemble.

La troisième et dernière contribution de nos travaux est l'exploitation directe de ces conclusions par la mise en point d'une procédure d'induction dynamique de forêts. Son principe est de guider l'induction des arbres pour qu'ils complètent au mieux la forêt déjà construite. Pour cela, nous avons présenté notre approche qui s'appuie sur un rééchantillonnage adaptatif des données d'apprentissage, à l'image de celui mis en place dans les procédures de Boosting. Nous avons ainsi présenté un algorithme original nommé Dynamic Random Forest, dont nous avons proposé et testé plusieurs variantes. Nous avons alors montré que cette procédure d'induction dynamique remplit son objectif en diminuant significativement et dans une grande majorité des cas, l'erreur en généralisation, en comparaison avec les processus d'induction statique. Nous avons également montré que ce gain de performances se traduit de façon intéressante par une augmentation progressive de la *force* des forêts, et une diminution globale de leur *corrélation*.

Ces travaux, s'ils apportent des éléments de réponses aux différentes problématiques de paramétrisation des forêts aléatoires et de compréhension de leur fonctionnement, ouvrent également la voie à quelques perspectives intéressantes.

Dans un premier temps, nous pensons qu'il serait intéressant de poursuivre l'étude de l'influence du procédé Random Feature Selection sur le comportement des forêts, via les propriétés de *force* et de *corrélation* par exemple. Il serait notamment utile de comprendre dans quelle mesure les valeurs du paramètre K influent sur ces deux propriétés, pour pouvoir par exemple affiner son réglage en fonction des valeurs de *force* et de *corrélation* observées sur l'ensemble d'apprentissage. Ces deux propriétés permettraient de cette façon de définir une règle de paramétrisation pour déterminer *a priori* la valeur de K^*.

Dans ce cadre, nous trouvons qu'il serait aussi intéressant de mettre en place une variante de Forest-RK, qui choisirait la valeur de K aléatoirement pour chaque arbre de la forêt, plutôt que pour chaque nœud. En comparant l'ensemble de ces procédures, on pourrait ainsi extrapoler le rôle de l'aléatoire dans le processus de sélection des règles de partitionnement, en fonction de son degré d'intervention.

En perspective de notre deuxième contribution, nous pensons que d'autres éléments pourraient venir compléter l'analyse de la *force* et de la *corrélation*, dans une optique d'induction dynamique de forêts. Par exemple, l'une des principales forces du Bagging est de proposer des mesures *out-of-bag* pour l'estimation de l'erreur en généralisation. On serait alors tenté d'utiliser ces mesures pour guider l'apprentissage des arbres dans un processus d'induction dynamique de forêt, de sorte qu'ils minimisent au mieux l'erreur en généralisation. Cependant, nous avons vu en introduction du chapitre 4 que la fiabilité de ces mesures en tant qu'estimation de cette erreur en généralisation est parfois remise en cause. Il nous semble donc important de réaliser une analyse du même genre que celle que nous avons menée dans le chapitre 4, en nous focalisant cette fois sur les estimations *out-of-bag*. On pourrait de cette façon percevoir si ces mesures sont effectivement corrélées avec l'erreur en généralisation, et si elles peuvent de ce fait être utilisées dans un processus d'induction dynamique.

Enfin, une perspective de notre troisième contribution concernent l'étude d'un ou plusieurs autres procédés d'induction dynamique de forêt. Nous avons dans un premier temps fait le choix d'utiliser un rééchantillonnage des données pour le guidage de l'apprentissage des arbres. On peut cependant imaginer beaucoup d'autres possibilités pour cela. Par exemple, nous pensons qu'il serait possible de guider ces apprentissages à l'aide d'une pondération des caractéristiques. Cette pondération permettrait d'indiquer à l'algorithme d'induction des arbres quelles sont les caractéristiques dont il faut privilégier le choix comme règle de partitionnement à chaque nœud. De cette façon, on pourrait contraindre les partitionnements successifs des arbres à se focaliser sur certaines caractéristiques pour la prédiction de certaines données en particulier. En combinant un procédé comme celui-ci à un rééchantillonnage adaptif tel que nous le mettons en place dans les Dynamic

Random Forest, on pourrait espérer mieux maîtriser la diversité dans l'ensemble, tout en renforçant encore plus la confiance des prédictions que fournit la forêt.

De façon plus générale, ces travaux offrent également des perspectives à plus long terme. Par exemple, nous avons vu que la particularité des forêts aléatoires est qu'elles utilisent des classifieurs élémentaires de types arbres de décision uniquement. Certaines de nos contributions, comme par exemple celles présentées dans le chapitre 3, concernent plus particulièrement les procédés de randomisation utilisés dans la procédure d'induction d'arbres, et sont donc propres aux forêts. Cependant, pour les autres contributions, des généralisations à tout type de classifieurs élémentaires sont envisageables. C'est particulièrement le cas pour notre processus d'induction dynamique qui ne nécessite pas spécialement que les classifieurs élémentaires soient des arbres. Il suffit pour que ce processus soit applicable que l'apprentissage de ces classifieurs élémentaires puisse prendre en compte la pondération des données, tout comme c'est le cas également pour le Boosting par exemple. Une perspective intéressante serait alors de généraliser et de tester notre principe d'induction dynamique qui s'appuie sur la proportion de votes pour la bonne classe, ainsi que sur une pondération adaptative des données d'apprentissage prenant en compte cette proportion, pour n'importe quel type de classifieurs élémentaires.

Publications de l'Auteur

1 S. Bernard, L. Heutte, S. Adam. *On the Selection of Decision Trees in Random Forests*. International Joint Conference on Neural Networks, Atlanta, USA, June 2009. pp. 302–307, 2009.

2 S. Bernard, L. Heutte, S. Adam. *Influence of Hyperparameters on Random Forest Accuracy*. 8th International Workshop on Multiple Classifier Systems, MCS 2009, Reykjavik, Iceland, Springer LNCS 5519, pp. 171–180, 2009.

3 S. Bernard, L. Heutte, S. Adam. *Towards a Better Understanding of Random Forests Through the Study of Strength and Correlation*. International Conference on Intelligent Computing, ICIC 2009, Ulsan, Korea, Springer LNAI 5755, pp. 536–545, 2009.

4 S. Bernard, L. Heutte, S. Adam. *Une Étude sur la Paramétrisation des Forêts Aléatoires*. XIème Conférence francophone sur l'Apprentissage Artificiel, Hammamet, Tunisie, pp. 81–92, 2009.

5 S. Bernard, L. Heutte, S. Adam. *Forest-RK : A New Random Forest Induction Method*. International Conference on Intelligent Computing, Shanghai, China. Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence, pp. 430–437, 2008.

6 L. Heutte, S. Bernard, S. Adam, E. Oliveira. *De la Sélection d'Arbres dans les Forêts Aléatoires*. Colloque International Francophone sur l'Ecrit et le Document, Rouen, France, pp. 163–168, 2008.

7 S. Bernard, L. Heutte, S. Adam. *Etude de l'Influence des Paramètres sur les Performances des Forêts Aléatoires*. Colloque International Francophone sur l'Ecrit et le Document, Rouen, France, pp. 207–208, 2008.

8 S. Bernard, S. Adam, L. Heutte. *Using Random Forests for Handwritten Digit Recognition*. 9th International Conference on Document Analysis and Recognition, Curitiba, Brazil, IEEE Proceedings, pp. 1043-1047, 2007.

Bibliographie

[Adam 2001] S. Adam. *Interprétation de Documents Techniques : des Outils à leur Intégration dans un Système à Base de Connaissances.* Thèse de Doctorat, Université de Rouen, France, 2001. 126

[Aksela 2003] M. Aksela. *Comparison of Classifier Selection Methods for Improving Committee Performance.* 4th International Workshop on Multiple Classifier System, vol. 2709, page 159, 2003. 111, 113

[Amit 1996] Y. Amit et D. Geman. *Shape Quantization and Recognition with Randomized Trees.* Neural Computation, vol. 9, pages 1545–1588, 1996. 42, 48

[Amit 2000] Y. Amit, G. Blanchard et K. Wilder. *Multiple Randomized Classifiers : MRCL.* Department of Statistics, University of Chicago, 2000. 11, 13, 42

[Asuncion 2007] A. Asuncion et D.J. Newman. *UCI Machine Learning Repository*, 2007. 23, 49, 68, 80

[Bahler 2000] D. Bahler et L. Navarro. *Methods for Combining Heterogeneous Sets of Classifiers.* 17th National Conference on Artificial Intelligence (AAAI 2000), Workshop on New Research Problems for Machine Learning, 2000. 16

[Banfield 2003] R.E. Banfield, L.O. Hall, K.W. Bowyer et W.P. Kegelmeyer. *A new Ensemble Diversity Measure Applied to Thinning Ensembles.* 4th International Workshop on Multiple Classifier Systems, vol. 2709, pages 306–316, 2003. 13, 113

[Banfield 2004] R.E. Banfield, L.O. Hall, K.W. Bowyer, D. Bhadoria, W.P.Kegelmeyer et S. Eschrich. *A comparison of ensemble creation techniques.* In The Fifth International Conference on Multiple Classifier Systems, pages 223–232, 2004. 16, 52, 69

[Banfield 2006] R.E. Banfield, L.O. Hall, K.W. Bowyer et W.P. Kegelmeyer. *A Comparison of Decision Tree Ensemble Creation Techniques.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, no. 1, pages 173–180, 2006. 11, 16, 22, 52, 69, 70, 71, 108, 132

[Bauer 1999] E. Bauer et R. Kohavi. *An Empirical Comparison of Voting Classification Algorithms : Bagging, Boosting, and Variants.* Machine Learning, vol. 36, no. 1-2, pages 105–139, 1999. 11, 42

[Benediktsson 2009] J.A. Benediktsson, J. Kittler et F. Roli (eds.). *8th International Workshop on Multiple Classifier Systems.* volume 5519 of *Lecture Notes in Computer Science*. Springer, 2009. 8, 15

[Boinee 2005a] P. Boinee, A. De Angelis et G.L. Foresti. *Ensembling Classifiers - An Application to Image Data Classification from Cherenkov Telescope Experiment.* World Academy of Science, Engineering and Technology, vol. 12, pages 66–70, 2005. 16, 52, 69

[Boinee 2005b] P. Boinee, A. De Angelis et G.L. Foresti. *Meta Random Forests*. International Journal of Computationnal Intelligence, vol. 2, no. 3, pages 138–147, 2005. 67, 69

[Breiman 1984] L. Breiman, J.H. Friedman, R.A. Olshen et C.J. Stone. Classification and regression trees. Chapman and Hall (Wadsworth, Inc.) : New York, 1984. 34, 36, 39, 41

[Breiman 1996] L. Breiman. *Bagging Predictors*. Machine Learning, vol. 24, no. 2, pages 123–140, 1996. 11, 19, 20, 42

[Breiman 1998] L. Breiman. *Arcing classifiers*. The Annals of Statistics, vol. 26, no. 3, pages 801–849, 1998. 139

[Breiman 2001] L. Breiman. *Random Forests*. Machine Learning, vol. 45, no. 1, pages 5–32, 2001. 30, 44, 45, 46, 49, 50, 51, 52, 54, 59, 62, 63, 68, 69, 72, 76, 103, 106, 107, 110, 120, 121, 136, 147, 164

[Breiman 2004] L. Breiman. *Consistency of random forests and other averaging classifiers*. Technical Report, Department of Statistics, University of California, Berkeley, 2004. 44

[Breitenbach 2002] M. Breitenbach, R. Neilsen et G.Z. Grudic. *Probabilistic Random Forest : Predicting Data Point Specific Misclassification Probabilities*. Technical Report, University of Colorado, 2002. 44, 65

[Breslow 1997] L.A. Breslow et D.W. Aha. *Simplifying Decision Trees : A Survey*. Knowledge Engineering Review, vol. 12, no. 1, pages 1–40, 1997. 32, 34, 35

[Brill 2003] R. Brill, R. Gutierrez et F. Quek. *Attribute Bagging : Improving Accuracy of Classifier Ensembles by Using Random Feature Subsets*. Pattern Recognition, vol. 36, no. 6, pages 1291–1302, 2003. 21

[Brostaux 2005] Y. Brostaux. *Etude du classement par forêts aléatoires d'échantillons perturbés à forte structure d'interaction*. Thèse de Doctorat, Université de Sciences Agronomiques de Gembloux, 2005. 32, 33

[Brown 2005] G. Brown, J. Wyatt, R. Harris et X. Yao. *Diversity Creation Methods : A Survey and Categorisation*. Journal of Information Fusion, vol. 6, no. 1, pages 5–20, 2005. 13

[Buntine 1992] W. Buntine et T. Niblett. *A Further Comparison of Splitting Rules for Decision-Tree Induction*. Machine Learning, vol. 8, pages 75–85, 1992. 36, 40

[Chatelain 2006] C. Chatelain. *Extraction de séquences numériques dans des documents manuscrits quelconques*. Thèse de Doctorat, Université de Rouen, France, 2006. 81, 82

[Chatelain 2007] C. Chatelain, S. Adam, Y. Lecourtier, L. Heutte et T. Paquet. *Multi-Objective Optimization for SVM Model Selection*. International Conference on Document Analysis and Recognition, Curitiba, Brazil, vol. 1, pages 427–431, 2007. 80, 82

[Chen 2004] C. Chen, A. Liaw et L. Breiman. *Using Random Forest to Learn Imbalanced Data*. Rapport Technique, Department of Statistics, University of California, Berkeley, 2004. 60, 61

[Cutler 2001] A. Cutler et G. Zhao. *PERT - Perfect Random Tree Ensembles*. Computing Science and Statistics, vol. 33, 2001. 58, 60

[Dasarathy 2005] B.V. Dasarathy. *Information Fusion, Special Issue on Diversity in Multiple Classifiers Systems*. volume 6. Elsevier Science, 2005. 8

[Dash 1997] M. Dash et H. Liu. *Feature selection for classification*. Intelligent Data Analysis, vol. 1, pages 131–156, 1997. 111, 112

[Dietterich 1991] T. G. Dietterich et G. Bakiri. *Error-Correcting Output Codes : A General Method for Improving Multiclass Inductive Learning Programs*. the ninth AAAI Conference on Artificial Intelligence, pages 572–577, 1991. 23, 41

[Dietterich 1995] T.G. Dietterich et G. Bakiri. *Solving Multiclass Learning Problems via Error-Correcting Output Codes*. Journal of Artificial Intelligence Research, vol. 2, pages 263–286, 1995. 24

[Dietterich 1998a] T.G. Dietterich. *Approximate statistical tests for comparing supervised classification learning algorithms*. Neural Computation, vol. 10, pages 1895–1923, 1998. 84

[Dietterich 1998b] T.G. Dietterich. *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees : Bagging, Boosting, and Randomization*. Machine Learning, vol. 40, pages 139–157, 1998. 42

[Dietterich 2000] T.G. Dietterich. *Ensemble Methods in Machine Learning*. First Workshop on Multiple Classifier Systems, vol. 1857, pages 1–15, 2000. 9, 11, 13, 15, 16, 17

[Duin 2000] R. Duin et D. Tax. *Experiments with Classifier Combining Rules*. First Workshop on Multiple Classifier Systems, vol. 1857, pages 16–29, 2000. 14

[Dumont 2009] M. Dumont, R. Marée, L. Wehenkel et P. Geurts. *Fast Multi-Class Image Annotation with Random Subwindows and Multiple Output Randomized Trees*. International Conference on Computer Vision Theory and Applications, vol. 2, pages 196–203, 2009. 68

[Efron 1993] B. Efron et R ; Tibshirani. An introduction to the bootstrap. Chapman et Hall, 1993. 18, 21

[Elgawi 2008] O. Hassab Elgawi. *Online random forests based on CorrFS and CorrBE*. Workshop on Computer Vision and Pattern Recognition, pages 1–7, 2008. 164

[Esposito 1997] F. Esposito, D. Malerba et G. Semeraro. *A Comparative Analysis of Methods for Pruning Decision Trees*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, pages 476–491, 1997. 34, 35

[Everitt 1977] B.S. Everitt. The analysis of contingency tables. Chapman and Hall, London, 1977. 84

[Fairhurst 1990] M. Fairhurst et F. Rahman. International journal on document analysis and recognition, special issue on multiple classifiers for document analysis applications, volume 6. Springer Berlin / Heidelberg, 1990. 8

[Feng 1993] C. Feng, A. Sutherland, S. King, S. Muggleton et R. Henery. *Comparison of Machine Learning Classifiers to Statistics and Nueral Networks.* International Conference on Artificial Intelligence and Statistics, 1993. 23

[Freund 1990] Y. Freund. *Boosting a Weak Learning Algorithm By Majority.* the third annual workshop on Computational learning theory, pages 202–216, 1990. 26

[Freund 1996] Y. Freund et R.E. Schapire. *Experiments with a New Boosting Algorithm.* International Conference on Machine Learning, pages 148–156, 1996. 26, 42

[Freund 2003] Y. Freund, R. Iyer, R.E. Schapire, Y. Singer et G. Dietterich. *An Efficient Boosting Algorithm for Combining Preferences.* Journal of Machine Learning Research, pages 170–178, 2003. 27

[Friedman 2000] J. Friedman, T. Hastie et R. Tibshirani. *Additive Logistic Regression : a Statistical View of Boosting.* Annals of Statistics, vol. 28, no. 2, pages 337–407, 2000. 27

[Geurts 2006] P. Geurts, D. Ernst et L. Wehenkel. *Extremely Randomized Trees.* Machine Learning, vol. 36, no. 1, pages 3–42, 2006. 52, 55, 56, 57, 58, 69, 77, 78, 93

[Giacinto 2000] G. Giacinto, F. Roli et G. Fumera. *Design of effective neural network ensembles for image classification purposes.* Image Vision and Computing Journal, vol. 19, no. 9, pages 699–707, 2000. 113

[Giacinto 2001] G. Giacinto et F. Roli. *An Approach to the Automatic Design of Multiple Classifier Systems.* Pattern Recognition Letters, vol. 22, pages 25–33, 2001. 8

[Goldberg 1989] D.E. Goldberg. Genetic algorithms in search, optimization and machine learning. Kluwer Academic Publishers, 1989. 124

[Grandvalet 2001] Y. Grandvalet, F. d'Alché Buc et C. Ambroise. *Boosting Mixture Models for Semi-supervised Learning.* 18th International Conference on Artificial Neural Networks, pages 41–48, 2001. 27

[Grandvalet 2004] Y. Grandvalet. *Bagging equalizes influence.* Machine Learning, pages 251–270, 2004. 20

[Guyon 2003] I. Guyon et A. Elisseeff. *An Introduction to Variable and Feature Selection.* Journal of Machine Learning Research, vol. 3, pages 1157–1182, 2003. 94

[Haindl 2007] M. Haindl, J. Kittler et F. Roli (eds.). *7th International Workshop on Multiple Classifier Systems.* volume 4472 of *Lecture Notes in Computer Science.* Springer, 2007. 8, 15

[Hao 2003] H. Hao, C. Liu et H. Sako. *Comparison of Genetic Algorithm and Sequential Search Methods for Classifier Subset Selection.* 7th International Conference on Document Analysis and Recognition, vol. 2, pages 765–769, 2003. 113

[Heutte 1994] L. Heutte. *Reconnaissance de Caractères Manuscrits : Application à la Lecture Automatique des Chèques et des Enveloppes Postales.* Thèse de Doctorat, Université de Rouen, France, 1994. 13

[Heutte 1998] L. Heutte, T. Paquet, J.V. Moreau, Y. Lecourtier et C. Olivier. *A Structural/Statistical Feature Based Vector for Handwritten Character Recognition.* Pattern Recognition Letters, vol. 19, no. 7, pages 629–641, 1998. 80, 81

[Ho 1992] T. Ho. *A Theory of Multiple Classifier Systems and its Application to Visual Word Recognition.* PhD Thesis, Departement of Computer Science, Suny at Buffalo, New York, 1992. 14, 22, 42

[Ho 1995] T.K. Ho. *Random Decision Forest.* Third International Conference on Document Analysis and Recognition, vol. 1, pages 278–282, 1995. 21, 22, 42

[Ho 1998] T.K. Ho. *The Random Subspace Method for Constructing Decision Forests.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pages 832–844, 1998. 17, 21, 22, 23, 42, 53

[Hsu 2002] C. Hsu et C. Lin. *A Comparison of Methods for Multi-class Support Vector Machines.* IEEE Transactions on Neural Networks, vol. 13, pages 415–425, 2002. 10

[Kass 1980] G.V. Kass. *An exploratory technique for investigating large quantities of categorical data.* Applied Statistics, vol. 29, pages 119–127, 1980. 41

[Kim 2000] J. Kim, K. Kim, C. Nadal et C. Suen. *A Methodology of Combining HMM and MLP Classifiers for Cursive Word Recognition.* International Conference Document Analysis and Recognition, vol. 2, pages 319–322, 2000. 14

[Kimura 1994] F. Kimura, S. Tsuruoka, Y. Miyake et M. Shridhar. *A Lexicon Directed Algorithm for Recognition of Unconstrained Handwritten Words.* IEICE Transaction on Information and System, vol. E77-D, no. 7, pages 785–793, 1994. 80

[Kittler 1998] J. Kittler, M. Hatef, R.P.W. Duin et J. Matas. *On combining classifiers.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, pages 226–239, 1998. 16

[Kittler 2000] Josef Kittler et Fabio Roli (eds.). *First International Workshop on Multiple Classifier Systems.* volume 1857 of *Lecture Notes in Computer Science*. Springer, 2000. 8, 15

[Kittler 2001] J. Kittler et F. Roli (eds.). *Second International Workshop on Multiple Classifier Systems.* volume 2096 of *Lecture Notes in Computer Science*. Springer, 2001. 8, 15

[Kittler 2003] J. Kittler, A. Ahmadyfard et D. Windridge. *Serial Multiple Classifier Systems Exploiting a Coarse to Fine Output Coding.* 4th International Workshop on Multiple Classifier Systems, vol. 2709, pages 159–160, 2003. 15

[Kittler 2004] J. Kittler. *On Serial Architectures for Multiple Classifier Systems.* Sheffield Machine Learning Workshop, 2004. 13, 15

[Kohavi 1997] R. Kohavi et G.H. John. *Wrappers for Feature Subset Selection.* Artificial Intelligence, vol. 97, no. 1-2, pages 273–324, 1997. 111

[Kong 1995] E.B. Kong et T.G. Dietterich. *Error-Correcting Output Coding Corrects Bias and Variance.* the Twelfth International Conference on Machine Learning, pages 313–321, 1995. 24

[Kuncheva 2000] L. Kuncheva, C. Whitaker, C. Shipp et R. Duin. *Is Independence Good for Combining Classifiers.* 15th International Conference on Pattern Recognition, vol. 2, pages 168–171, 2000. 13, 113

[Kuncheva 2001] L. Kuncheva, J. Bezdek et R. Duin. *Decision Templates for Multiple Classifier Fusion : an Experimental Comparison.* Pattern Recognition, vol. 34, no. 2, pages 299–314, 2001. 14

[Kuncheva 2003a] L. Kuncheva et C. Whitaker. *Measures of Diversity in Classifier Ensembles.* Machine Learning, vol. 51, pages 181–207, 2003. 12

[Kuncheva 2003b] L.I. Kuncheva. *That Elusive Diversity in Classifier Ensembles.* Iberian Conference on Pattern Recognition and Image Analysis, pages 1126–1138, 2003. 12, 13

[Kuncheva 2004] L.I. Kuncheva. Combining pattern recognition. methods and algorithms. John Wiley and Sons, 2004. 8, 12, 13, 14, 16

[Lanckriet 2002] G.R.G. Lanckriet, L.E. Ghaoui, C. Bhattacharyya et M.I. Jordan. *A Robust Minimax Approach to Classification.* Journal of Machine Learning Research, vol. 3, pages 555–582, 2002. 66

[Latinne 2000] P. Latinne, O. Debeir et C. Decaestecker. *Mixing Bagging and Multiple Feature Subsets to Improve Classification Accuracy of Decision Tree Combination.* Tenth Belgian-Dutch Conference on Machine Learning, 2000. 22

[Latinne 2001] P. Latinne, O. Debeir et C. Decaestecker. *Limiting the Number of Trees in Random Forests.* 2nd International Workshop on Multiple Classifier Systems, pages 178–187, 2001. 108, 109

[Latinne 2002] P. Latinne, O. Debeir et C. Decaestecker. *Combining Different Methods and Numbers of Weak Decision Trees.* Pattern Analysis and Applications (special issue on Fusion of Multiple Classifiers), vol. 5, no. 2, pages 201–209, 2002. 22

[LeCun 1998] Y. LeCun, L. Bottou, Y. Bengio et P. Haffner. *Gradient-Based Learning Applied to Document Recognition.* Proceedings of the IEEE, vol. 86, no. 11, pages 2278–2324, 1998. 80

[Liu 1994] W.Z. Liu et A.P. White. *The Importance of Attribute Selection Measures in Decision Tree Induction*. Machine Learning, vol. 15, no. 1, pages 25–41, 1994. 40

[Loh 1997] Wei-Yin Loh et Yu-Shan Shih. *Split Selection Methods for Classification Trees*. Statistica Sinica, vol. 7, pages 815–840, 1997. 32

[Malerba 1996] D. Malerba, F. Esposito et G. Semeraro. *A Further Comparison of Simplification Methods for Decision-Tree Induction*. In D. Fisher and H. Lenz (Eds.), Learning, pages 365–374, 1996. 34, 35

[Marée 2007] R. Marée, P. Geurts et L. Wehenkel. *Content-based Image Retrieval by Indexing Random Subwindows with Randomized Trees*. 8th Asian Conference on Computer Vision, vol. 4844, pages 611–620, 2007. 68

[Margineantu 1997] D.D. Margineantu et T.G. Dietterich. *Pruning Adaptive Boosting*. 14th International Conference on Machine Learning, pages 211–218, 1997. 65

[Mingers 1989a] John Mingers. *An Empirical Comparison of Pruning Methods for Decision-Tree Induction*. Machine Learning, vol. 4, pages 227–243, 1989. 34, 35

[Mingers 1989b] John Mingers. *An Empirical Comparison of Selection Measures for Decision-Tree Induction*. Machine Learning, vol. 3, pages 319–342, 1989. 36, 37, 40

[Minsky 1991] M.L. Minsky. *Logical versus analogical or symbolic versus connectionist or neat versus scruffy*. AI Magazine, vol. 12, no. 2, pages 34–51, 1991. 12

[Moobed 1996] B. Moobed. *Combinaison de Classifieurs, une Nouvelle Approche*. Thèse de Doctorat, Université de Paris Sud, UFR Scientifique d'Orsay, France, 1996. 13

[Murthy 1997] S.K. Murthy. *On Growing Better Decision Trees From Data*. 1997. 31, 32

[Murthy 1998] S.K. Murthy. *Automatic Construction of Decision Trees from Data : A Multi-Disciplinary Survey*. Data Mining and Knowledge Discovery, vol. 2, no. 4, pages 345–389, 1998. 32, 36, 37

[Nilsson 1965] N. Nilsson. Learning machines. McGraw-Hill, 1965. 8

[Opitz 1999a] D. Opitz et R. Maclin. *Popular Ensemble Methods : An Empirical Study*. Journal of Artificial Intelligence Research, vol. 11, pages 169–198, 1999. 17

[Opitz 1999b] D. Opitz et J.W. Shavlik. *Actively Searching for an Effective Neural Network Ensemble*. Connection Science, vol. 8, no. 3, pages 337–354, 1999. 127

[Oza 2005] Nikunj C. Oza, Robi Polikar, Josef Kittler et Fabio Roli (eds.). *6th International Workshop on Multiple Classifier Systems*. volume 3541 of *Lecture Notes in Computer Science*. Springer, 2005. 8, 15

[Partridge 1996] D. Partridge et W. Yates. *Engineering Multiversion Neural-Net Systems*. Neural Computation, vol. 8, pages 869–893, 1996. 113

[Prudent 2005] Y. Prudent et A. Ennaji. *A K Nearest Classifier Design*. Electronic Letters on Computer Visions and Image Analysis, vol. 5, no. 2, pages 58–71, 2005. 80

[Quinlan 1986a] J.R. Quinlan. *Induction of decision trees*. Machine Learning, vol. 1, no. 1, pages 81–106, 1986. 36, 37, 41

[Quinlan 1986b] J.R. Quinlan. *Simplifying Decision Tree*. International Journal of Man-Machine Studies, no. 27, 1986. 34, 35

[Quinlan 1993] J.R. Quinlan. C4.5 : Programs for machine learning. Morgan Kaufman, 1993. 31, 32, 34, 41

[Rahman 1999] A. Rahman et M. Fairhurst. *A Study of Some Multi-Expert Recognition Strategies for Industrial Applications : Issues of Processing Speed and Implementability*. International Conference on Vision Interface, 1999. 13

[Rahman 2003] A.F.R. Rahman et M.C. Fairhurst. *Multiple classifier decision combination strategies for character recognition : A review*. International Journal on Document Analysis and Recognition, vol. 5, no. 4, 2003. 13, 14, 15

[Rakotomalala 2005] Ricco Rakotomalala. *Arbres de Décision*. Modulad, vol. 33, 2005. 31, 32

[Robnik-Sikonja 2004] M. Robnik-Sikonja. *Improving Random Forests*. European Conference on Machine Learning, LNAI 3210, Springer, Berlin, pages 359–370, 2004. 44, 62

[Rodriguez 2006] J.J. Rodriguez, L.I. Kuncheva et C.J. Alonso. *Rotation Forest : A New Classifier Ensemble Method*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 10, pages 1619–1630, 2006. 63, 65

[Roli 2001] F. Roli, G. Giacinto et G. Vernazza. *Methods for Designing Multiple Classifier Systems*. 2nd Workshop on Multiple Classifier Systems, pages 78–87, 2001. 13, 15, 113, 114

[Roli 2002a] F. Roli et J. Kittler (eds.). *Third International Workshop on Multiple Classifier Systems*. volume 2364 of *Lecture Notes in Computer Science*. Springer, 2002. 8, 15

[Roli 2002b] F. Roli et J. Kittler, editeurs. Information fusion, special issue on fusion of multiple classifiers, volume 3. Elsevier Science, 2002. 8

[Roli 2004] F. Roli, J. Kittler et T. Windeatt. *5th International Workshop on Multiple Classifier Systems*. volume 3077 of *Lecture Notes in Computer Science*. Springer, 2004. 8, 15

[Ruta 200] D. Ruta et B. Gabrys. *An Overview of Classifier Fusion Methods*. Computing and Information Systems, vol. 7, pages 7–10, 200. 14

[Ruta 2004] D. Ruta et B. Gabrys. *Classifier Selection for Majority Voting*. Information Fusion, vol. 6, no. 1, pages 63–81, 2004. 8, 111

[Saffari 2009] A. Saffari, C. Leistner, J. Santner, M. Godec et H. Bischof. *On-line Random Forests.* 3rd IEEE ICCV Workshop on On-line Computer Vision, 2009. 164

[Santos 2008] E.M. Dos Santos, R. Sabourin et P. Maupin. *A dynamic overproduce-and-choose strategy for the selection of classifier ensembles.* Pattern Recognition, vol. 41, pages 2993–3009, 2008. 127

[Schapire 1990] R. Schapire. *The Strength of Weak Learnability.* Machine Learning, vol. 5, pages 197–227, 1990. 25, 26

[Sebban 2000] M. Sebban, R. Nock, H. Chauchat et R. Rakotomalala. *Impact of Learning Set Quality and Size on Decision Tree Performances.* International Journal of Computers Systems and Signals, vol. 1(1), pages 85–105, 2000. 34

[Sebban 2003] M. Sebban et M. Suchier. *On Boosting Improvement : Error Reduction and Convergence Speed-Up.* Fourteenth European Conference on Machine Learning (ECML 2003), pages 349–360, 2003. 27

[Sejnowski 1987] T.J. Sejnowski et C.R. Rosenberg. *Parallel Networks that Learn to Pronounce English Text.* Complex Systems, pages 145–168, 1987. 24

[Sharkey 2000] A. Sharkey, N. Sharkey, U. Gerecke et G. Chandroth. *The Test and Select Approach to Ensemble Combination.* First Workshop on Multiple Classifier Systems, vol. 1857, pages 30–44, 2000. 113

[Shlien 1990] S. Shlien. *Multiple binary decision tree classifiers.* Pattern Recognition, vol. 23, no. 7, pages 757–763, 1990. 41

[Sicard 2006] R. Sicard, T. Artières et E. Petit. *Patch Learning for Incremental Classifier Design.* 17th European Conference on Artificial Intelligence, pages 807–808, 2006. 164

[Sicard 2008] R. Sicard, T. Artières et E. Petit. *Learning iteratively a classifier with the Bayesian Model Averaging Principle.* Pattern Recognition, vol. 41, no. 3, pages 930–938, 2008. 164

[Skurichina 2001] M. Skurichina. *Stabilizing Weak Classifiers.* 2001. 20

[Tax 2000] D. Tax et R. Duin. *Experiments with classifier combining rules.* First International Workshop on Multiple Classifier Systems, vol. 1857, pages 16–2, 2000. 16

[Tax 2001] D. Tax et R. Duin. *Combining One-Class Classifiers.* 2nd International Workshop on Multiple Classifier Systems, vol. 2096, pages 299–308, 2001. 16

[Torre 2004] F. Torre. *GLOBOOST Combinaisons de moindres généralisés : Produire des moindres généralisés et les combiner au sein d'un vote pondéré.* Conference sur l'Apprentissage Automatique, vol. 19, pages 769–797, 2004. 27

[Tsymbal 2006] A. Tsymbal, M. Pechenizkiy et P. Cunningham. *Dynamic Integration with Random Forest.* 17th European Conference on Machine Learning, vol. 4212, pages 801–808, 2006. 44

[Windeatt 2003] T. Windeatt et F. Roli (eds.). *4th International Workshop on Multiple Classifier Systems*. volume 2709 of *Lecture Notes in Computer Science*. Springer, 2003. 8, 15

[Wolpert 1997] D.H. Wolpert et W.G. Macready. *No free lunch theorems for optimization*. IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pages 67–82, 1997. 12

[Zouari 2002] H. Zouari, L. Heutte, Y. Lecourtier et A. Alimi. *Un panorama des méthodes de combinaison de classifieurs en reconnaissance de formes*. 13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et d'Intelligence Artificielle, vol. 2, pages 499–508, 2002. 14

[Zouari 2004] H. Zouari. *Contribution à l'évaluation des méthodes de combinaison parallèle de classifieurs par simulation*. Thèse de Doctorat, Université de Rouen, France, 2004. 13, 14, 111, 113

Forêts Aléatoires : De l'Analyse des Mécanismes de Fonctionnement à la Construction Dynamique

Résumé : Les travaux de cette thèse se situent dans le domaine de l'apprentissage automatique et concernent plus particulièrement la paramétrisation des forêts aléatoires, une technique d'ensembles de classifieurs utilisant des arbres de décision. Nous nous intéressons à deux paramètres importants pour l'induction de ces forêts : le nombre de caractéristiques choisies aléatoirement à chaque noeud et le nombre d'arbres. Nous montrons d'abord que la valeur du premier paramètre doit être choisie en fonction des propriétés de l'espace de description, et proposons dans ce cadre un nouvel algorithme nommé Forest-RK exploitant ces propriétés. Nous montrons ensuite qu'avec un processus statique d'induction de Forêts, certains arbres provoquent une diminution des performances de l'ensemble, en dégradant le compromis *force/corrélation*. Nous en déduisons un algorithme d'induction dynamique particulièrement performant en comparaison avec les procédures d'induction statique.

Mots clés : Forêts Aléatoires, Ensemble de Classifieurs, Arbre de Décision, Sélection aléatoire de caractéristiques, Bagging, Induction dynamique de forêt

Random Forests : From the Study of Behaviors to Dynamic Induction

Abstract : This research work is related to machine learning and more particularly deals with the parametrization of Random Forests, which are classifier ensemble methods that use decision trees as base classifiers. We focus on two important parameters of the forest induction : the number of features randomly selected at each node and the number of trees. We first show that the number of random features has to be chosen regarding to the feature space properties, and we propose hence a new algorithm called Forest-RK that exploits those properties. We then show that a static induction process implies that some of the trees of the forest make the ensemble generalisation error decrease, by deteriorating the *strength/correlation* compromise. We finaly propose an original random forest dynamic induction algorithm that favorably compares to static induction processes.

Keywords : Random Forests, Ensemble of Classifiers, Decision Trees, Random Feature Selection, Bagging, Dynamic Random Forest Induction