FORÊTS ALÉATOIRES PAC-BAYÉSIENNES

Mémoire

Brice ZIRAKIZA

Maîtrise en informatique
Maître ès sciences (M.Sc.)

Québec, Canada

Résumé

Dans ce mémoire de maîtrise, nous présentons dans un premier temps un algorithme de l'état de l'art appelé Forêts aléatoires introduit par Léo Breiman. Cet algorithme effectue un vote de majorité uniforme d'arbres de décision construits en utilisant l'algorithme CART sans élagage. Par après, nous introduisons l'algorithme que nous avons nommé SORF. L'algorithme SORF s'inspire de l'approche PAC-Bayes, qui pour minimiser le risque du classificateur de Bayes, minimise le risque du classificateur de Gibbs avec un régularisateur. Le risque du classificateur de Gibbs constitue en effet, une fonction convexe bornant supérieurement le risque du classificateur de Bayes. Pour chercher la distribution qui pourrait être optimale, l'algorithme SORF se réduit à être un simple programme quadratique minimisant le risque quadratique de Gibbs pour chercher une distribution Q sur les classificateurs de base qui sont des arbres de la forêt. Les résultasts empiriques montrent que généralement SORF est presqu'aussi bien performant que les forêts aléatoires, et que dans certains cas, il peut même mieux performer que les forêts aléatoires.

Abstract

In this master's thesis, we present at first an algorithm of the state of the art called Random Forests introduced by Léo Breiman. This algorithm construct a uniformly weighted majority vote of decision trees built using the CART algorithm without pruning. Thereafter, we introduce an algorithm that we called SORF. The SORF algorithm is based on the PAC-Bayes approach, which in order to minimize the risk of Bayes classifier, minimizes the risk of the Gibbs classifier with a regularizer. The risk of Gibbs classifier is indeed a convex function which is an upper bound of the risk of Bayes classifier. To find the distribution that would be optimal, the SORF algorithm is reduced to being a simple quadratic program minimizing the quadratic risk of Gibbs classifier to seek a distribution Q of base classifiers which are trees of the forest. Empirical results show that generally SORF is almost as efficient as Random forests, and in some cases, it can even outperform Random forests.

Table des matières

Liste des tableaux

Liste des figures

A toi ma très chère, et regrettée
mère, pour l'affection que tu n'a
jamais cessée de me témoigner,
je dédie ce mémoire.

Avant-propos

Le présent mémoire est le fruit des efforts consentis tout au long de ma formation au programme de Maîtrise en Informatique de l'Université Laval. Je profite de cette opportunité pour remercier tous ceux qui, de près ou de loin, m'ont assisté dans la réalisation de cette oeuvre.

Mes sincères et considérables remerciements aux Professeurs, Mario Marchand qui a accepté de diriger ce travail, ainsi que mon co-directeur de recherche François Laviolette. C'est grâce à leurs conseils techniques, leurs suggestions et connaissances qu'ils m'ont transmis que ce travail a pu être accompli.

Mes remerciements s'adressent également à tous mes collègues du magnifique Groupe de Recherche en Apprentissage Automatique de Laval (GRAAL). Leur amité, leur professionalisme et leur collaboration pendant les activités de recherche sont des qualités que je suis sincèrement reconnaissant d'avoir appris et partagé avec eux. Je remercie particulièrement Pascal Germain qui m'a aidé dans mes expérimentations concernant l'algorithme SORF.

Mes remerciements vont également à ma très chère regrettée mère, feue Léa Girukwishaka et à mon père André Ndikumwami pour leurs sacrifices qu'ils se sont imposés en m'accordant leur soutien tant moral que matériel afin que je sois la personne que je suis aujourd'hui. Je remercie également mes frères, Elvis et Armel, pour leur soutien moral.

Enfin mes sentiments de gratitude s'adressent à mon Seigneur tout puissant qui est ma forteresse, lui qui m'a guidé et inspiré depuis mes premiers pas. Il est mon sauveur et mon berger pour toujours.

Chapitre 1

Introduction

L'informatique est un domaine très vaste avec beacoup de sous-domaines dont l'intelligence artificielle. Ces derniers temps, l'expression "intelligence artificielle" est fréquemment utilisé dans le public car il s'agit d'un domaine en constante évolution notamment grâce aux progrès des technologies informatiques et entre autres grâce aux capacités toujours plus grandes des machines pour effectuer les calculs. La science-fiction y a également une part importante quant à la vulgarisation du terme intelligence artificielle. Depuis ses origines, l'intelligence artificielle avait généralement pour objectif de doter les machines de la capacité à pouvoir effectuer des tâches réputées "intelligentes". Cependant, cet objectif qui pourrait aussi être considéré comme la définition de l'intelligence artificielle s'est rétrouvé vague. Ce que l'on constate, c'est que les recherches actuelles en intelligence artificielle tendent à rendre la machine capable d'acquérir de l'information, de raisonner sur une situation statique ou dynamique, de résoudre des problèmes combinatoires, de faire un diagnostic, de proposer une décision, un plan d'action, d'expliquer et de communiquer les conclusions(13).

C'est dans ce cadre plus ou moins précis qu'apparaît l'apprentissage automatique qui constitue l'un des sous-domaines de l'intelligence artificielle. Nos travaux de recherche se sont focalisés sur le domaine de l'apprentissage automatique dont l'objectif est de créer des algorithmes permettant à des applications ou systèmes informatiques d'avoir une certaine capacité d'"apprendre" en observant un environnement précis. Au cours de ce mémoire, nous allons revenir sur ce que c'est l'apprentissage automatique, et essayer de contribuer à répondre à une problématique (que nous allons poser) grâce aux connaissances acquises dans ce domaine.

Nous allons nous focaliser sur les problèmes de classification qui consistent à catégori-

ser divers éléments d'un environnement à partir d'un modèle construit en observant des exemples de cet environnement. Nous nous intéresserons à quelques techniques permettant la création d'algorithmes d'apprentissage. Dans le domaine de la classification, un algorithme d'apprentissage construit un classificateur. L'objectif de cet algorithme sera donc de construire un classificateur avec une bonne capacité de prédiction. Autrement dit, un classificateur de "faible risque", le risque étant la probabilité qu'un classificateur classifie incorrectement un exemple. Nous allons présenter un algorithme d'apprentissage appelé Forêts aléatoires qui appartient à l'état de l'art. Nous effectuerons une revue de littérature sur différentes notions reliées à cet algorithme. Par après, on va présenter une borne appelée "borne C_Q" et nous reviendrons sur la théorie PAC-Bayes utilisée pour développer des algorithmes d'apprentissage automatique. Nous présenterons d'une manière brève un autre algorithme de l'état de l'art appelé MinCQ(20) qui se base sur la théorie PAC-Bayes pour minimiser la "borne C_Q". Enfin nous allons proposer un algorithme que nous avons nommé SORF qui s'inspire de l'approche PAC-Bayes et qui pourrait concurrencer les forêts aléatoires dans certains cas que nous allons citer.

Ce mémoire fait donc figure de compte-rendu des travaux effectués et résultats obtenus dans notre maitrise avec recherche en apprentissage automatique. Le contenu de ce mémoire est alors ainsi organisé :

- Le chapitre 2 (**Notions de base de l'apprentissage automatique**) revient sur ce que c'est l'apprentissage automatique et sert d'introduction aux différents principes de base de l'apprentissage automatique et de la classification.
- Le chapitre 3 (**Les arbres de décision**) sert d'introduction aux concepts de base sur les arbres de décision. Dans ce chapitre, on revient sur la façon dont sont induits les arbres de décision et sur la façon dont ils procédent pour classifier des exemples. On va également présenter quelques métriques couramment utiliser par les arbres de décision pour déterminer les attributs qui seront utilisés pour le partitionnement dans la création des noeuds des arbres.
- Le chapitre 4 (**Les forêts aléatoires**) présente un algorithme de l'état de l'art nommé Random forests ou forêts aléatoires. Nous présentons d'une manière détaillée cet algorithme et nous effectuons une analyse concernant la façon de faire de cet algorithme tout en montrant ses forces.
- Le chapitre 5 (**Les bornes dérivées de la théorie Pac-Bayes**) présente une borne appelée C_Q qui a été utilisée pour les forêts aléatoires. Nous présentons également la théorie PAC-Bayes ainsi que des bornes qui sont directement dérivées d'une approche basée sur cette théorie. Nous allons également présenter un algorithme de l'état de

l'art appelé MinCQ qui est un résultat de la minimisation de la borne C_Q en suivant une approche basée sur la théorie PAC-Bayes.

- Le chapitre 6 (**SORF : Une forêt aléatoire inspirée de l'approche PAC-Bayesienne**) présente un algorithme qui est le résultat de nos travaux de recherche. Il s'agit d'un algorithme qui se résume à être un programme quadratique qui minimise le risque quadratique de Gibbs.
- Le chapitre 7 (**Résultats empiriques**) présente les résultats empiriques obtenus en exécutant l'algorithme SORF sur quelques ensembles de données. Dans ce chapitre, nous comparons les résultats de l'algorithme SORF avec ceux optenus par les forêts aléatoires et MinCQ.
- Le chapitre 8 (**Conclusion et travaux futurs**) tire une conclusion sur nos travaux de maitrise et à la lumière des résultats obtenus propose des voies que nous pensons valent la peine d'être explorées pour des travaux futurs.

Chapitre 2

Notions de base de l'apprentissage automatique

2.1 Introduction à l'apprentissage automatique

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle qui s'intéresse à la création d'algorithmes qui "apprennent" à effectuer une certaine tâche par l'observation d' "exemples". Ces exemples constituent un échantillon d'un domaine particulier qui a été regroupé puis traité par des experts de ce domaine. Un système d'apprentissage automatique peut permettre par exemple de déterminer parmi les champignons qu'on lui présente lesquels sont vénéneux et lesquels ne le sont pas. Le système apprendra d'abord un modèle à partir d'une banque de données de champignons qui a été constitué par des experts. Ce modèle lui permettra, lors de la confrontation avec un champignon inconnu, de prédire (avec une marge d'erreur explicite) la valeur de la variable "vénéneux" ("oui", ou "non"). La reconnaissance de caractères manuscrits à partir d'images est un autre exemple d'une tâche complexe nécessitant l'apprentissage automatique car deux images de caractères manuscrits similaires ne sont jamais exactement identiques sachant que les deux caractères sont parfois écrits de manière différente par deux ou plusieurs personnes différentes. On peut concevoir un système d'apprentissage automatique qui apprend à reconnaître des caractères en observant des "exemples", c'est-à-dire dans ce cas des images de caractères manuscrits connus . Un algorithme d'apprentissage automatique aura donc pour objectif de tirer des règles générales à partir d'observations particulières ou "exemples". Dans le contexte de la "classification", l'algorithme d'apprentissage sera exécuté sur un ensemble de données contenant une série d'exemples où chaque exemple est caracterisé par une "description"

et une “étiquette” de la classe. L'algorithme d'apprentissage doit généraliser l'information contenue dans l'ensemble de données afin de créer une fonction qui prend en entrée la description d'un exemple et prédit son étiquette. La fonction créée doit donc approcher au mieux la relation existant entre toutes les descriptions et leurs étiquettes. Cependant des difficultés apparaissent car la relation entre les descriptions des exemples et leurs étiquettes est fixe mais inconnue et on ne dispose que d'un ensemble de données comme information partielle sur cette relation. Notre tâche est donc de developper un algorithme d'apprentissage qui produit une fonction ayant de bonnes qualités de généralisation pour approximer de la meilleure façon possible cette relation.

2.2 Apprentissage supervisé

Selon la nature du problème qu'on traite, un problème d'apprentissage automatique est généralement considéré comme appartenant à l'un des quatres sous-domaines suivants à savoir : l'apprentissage supervisé, l'apprentissage semi-supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. L'apprentissage par renforcement concerne les problèmes d'apprentissage de tâches par des agents où le but est de maximiser les récompenses reçues. Rappelons qu'un agent est un système informatique, situé dans un environnement, qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu(33). Dans le cadre de l'apprentissage non-supervisé, on se trouve en face d'un problème où on a des données qui ne sont pas étiquetés et on doit déterminer une structure intéressante à partir de ces données. En apprentissage semi-supervisé, on a des données étiquetées et d'autres qui ne le sont pas. L'étiquetage des données nécessitant une intervention humaine, il apparait parfois qu'on se retrouve avec un ensemble de données qui a été étiqueté d'une manière partielle. Les techniques d'apprentissage semi-supervisé s'avèrent donc adéquates dans ce cas. Dans le cas où on a des données qui sont totalement étiquetées, on utilise les techniques d‘apprentissage supervisé. Un algorithme d'apprentissage supervisé a pour objectif de déterminer la correspondance entre la description d'un exemple et son étiquette de classe (ou catégorie). Par exemple pour un problème de reconnaissance de caractères manuscrits numériques, l'objectif est de déterminer si un caractère manuscrit correspond à une valeur numérique appartenant au domaine (0,...,9).
Plusieurs applications de la vie courante font intervenir des algorithmes d'apprentissage supervisé. Quelques unes de ces applications sont par exemple la catégorisation de textes et de discours, l'étiquetage grammatical (part-of-speech tagging en anglais)

dans le traitement automatique du langage naturel, etc. Ce mémoire concerne l'apprentisssage supervisé, i.e., l'étude des algorithmes produisant une fonction de prédiction (ou une relation) à partir d'un ensemble de données étiquetées. Dépendant de la sortie de notre algorithme, on parlera de classification (les étiquettes appartiennent a priori à un ensemble fini relativement petit), de régression (les étiquettes peuvent être tout nombre réel) ou de prédiction de structures (la sortie de l'algorithme est une structure plus complexe).

2.3 Représentation des données

On désigne par le terme **exemple** une manifestation d'un phénomène et on représente cet exemple par une paire description-étiquette $(\mathbf{x}, y)$. Selon qu'il s'agisse d'une classification binaire ou multiclasse, l'étiquette peut prendre deux valeurs ou plusieurs valeurs respectivement. Il existe, dans la vie de tous les jours, de nombreuses applications nécessitant des algorithmes de classification multiclasse(29). Cependant dans le developpement des algorithmes d'apprentissage automatique, il est souvent plus facile de d'abord mettre au point un algorithme qui fait de la classification binaire et de le généraliser plus tard à plusieurs classes. Certaines méthodes(26) permettent à ce propos de diviser un problème multiclasse en un ensemble de sous-problèmes de classification binaire. Le problème multiclasse est par la suite résolu par une série d'appels à un algorithme d'apprentissage dédié à la classification binaire.
La description d'un exemple consiste en un vecteur d'attributs réels. Suivant le phénomène qu'on étudie, un attribut peut être numérique (par exemple, la taille de la hauteur d'un champignon) ou nominal (par exemple la forme d'un polygone). Dans le cas où on a des attributs nominaux, on effectue une équivalence de telle sorte que par exemple pour la forme d'un polygone, 1 équivaut à «carré», 2 équivaut à «circulaire» et 3 équivaut à «rectangulaire». L'étiquette est soit négative avec comme valeur -1 ou positive avec comme valeur +1 dans le cas binaire :

$$\begin{aligned} \mathbf{x} &\in \mathcal{X} \subseteq \mathbb{R}^n \\ y &\in \mathcal{Y} = \{-1, +1\} \end{aligned}$$

Dans le cas multiclasse, l'étiquette devient :

$$y \in \mathcal{Y} = \{1, 2, ..., l\}$$

avec l étant le nombre de classes.
L'ensemble $\mathcal{X}$ est appelé ensemble d'entrée et l'ensemble $\mathcal{Y}$ est appelé ensemble de

sortie. La description de l'exemple pourrait être réecrite comme $\mathbf{x} = (x_1, x_2, x_3, .., x_n)$ avec n étant le nombre d'**attributs** de l'exemple. Pour illustrer cela, on peut considérer l'exemple des champignons. Les attributs pour décrire un champignon $\mathbf{x}$ seraient la hauteur du pied et le diamètre du chapeau. En fonction de certains critères sur ces attributs, un champigon pourrait être classé comme étant comestible ou vénéneux. On peut donc considérer une étiquette pour chacune des deux classes : +1 pour les champignons comestibles et -1 pour les champignons véneneux.
Un **ensemble de données** est une collection de plusieurs exemples issus de l'observation d'un même phénomène. On pose comme hypothèse que les exemples d'un ensemble de données sont générés de manière indépendante et identique par une distribution de probabilités D sur $\mathcal{X} \times \mathcal{Y}$.

2.4 Classificateur

Un **classificateur** est une fonction qui reçoit en entrée la description d'un exemple et retourne une étiquette. Un classificateur h est donc une fonction $h : \mathcal{X} \to \{-1, +1\}$. La fonction h est parfois utilisée sous le terme "hypothèse".

2.5 Decision stump

Le *decision stump* est parmi les classificateurs les plus simple qui soient. Il s'agit d'un arbre de décision (voir chapitre 3) à un seul niveau. Pour classifier un exemple $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, le decision stump $h_{i,t,d}$ considère la valeur d'un seul attribut x_i. La sortie du classificateur est déterminée par la valeur de seuil $t \in \mathbb{R}$ et la direction $d \in \{-1, +1\}$:

$$h_{i,t,d}(\mathbf{x}) = \begin{cases} +d & \text{si } x_i > t \\ -d & \text{sinon.} \end{cases}$$

2.6 Risque d'un classificateur

La notion de **risque** est une notion primordiale en apprentissage automatique car c'est elle qui nous indique le taux d'erreur du classificateur et nous sert d'indicateur de performance de notre classificateur. Le risque $R(h)$ d'un classificateur consiste en la probabilité qu'il classifie incorrectement un exemple généré par une distribution de

probabilité D sur $\mathcal{X} \times \mathcal{Y}$:

$$R(h) \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} I\Big(h(\mathbf{x}) \neq y\Big),$$

où I est une fonction indicatrice : $I(a) = 1$ si l'énoncé a est vrai et $I(a) = 0$ sinon.

2.7 Risque empirique et surapprentissage

Comme on ne peut pas calculer le **vrai risque** $R(h)$, on l'estime par le risque empirique qu'on calcule sur l'ensemble d'entrainement. Le **risque empirique** $R_S(h)$ d'un classificateur sur l'ensemble de données $S \sim D^m$ comprenant m exemples est donc le ratio d'erreurs sur cet ensemble :

$$R_S(h) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{(\mathbf{x},y)\in S} I\Big(h(\mathbf{x}) \neq y\Big).$$

Le risque empirique $R_S(h)$ d'un classificateur n'est pas nécessairement un bon indicateur de son vrai risque $R(h)$. En effet, il arrive des cas où on est en présence d'un classificateur ayant un risque empirique moins élévé alors que le vrai risque est élévé. On parle de **surapprentissage** (ou **overfitting** en anglais). La situation de surapprentissage pourrait être comparée à un étudiant qui se mettrait à apprendre par coeur les réponses de ses exercices de Mathématiques sans pour autant comprendre les concepts et qui aurait de mauvaises notes lors de ses examens étant donné que ce n'est pas les mêmes exercices qui reviendraient. Le surapprentissage arrive donc lorsqu'un classificateur ne généralise pas bien le phénomène étudié, la fonction de classification se concentre trop sur l'ensemble des données d'entrainement sans pour autant développer ses qualités de généralisation.

2.8 Algorithme d'apprentissage

Un **algorithme d'apprentissage** est un algorithme qui reçoit en entrée un ensemble de données d'entraînement S et fournit en sortie un classificateur h. Il est representé par une fonction A :

$$h = A(S)$$

En plus de l'ensemble d'entrainement S, l'algorithme peut recevoir en entrées des **hyperparamètres** c'est à dire des paramètres supplémentaires spécifiés préalablement à l'exécution de l'algorithme.

Le défi d'un algorithme d'apprentissage est de généraliser l'information contenue dans l'ensemble d'entraînement afin de produire un classificateur de faible risque. Étant donné que la distribution D qui génère les exemples est inconnue dans la plupart des applications, il est impossible de connaître le risque d'un classificateur. Une pratique courante consiste à estimer le risque en calculant le risque empirique sur un **ensemble de données de test** $T \sim D^{|T|}$ qui n'ont pas servi au processus d'entraînement.

2.9 Validation croisée

La *validation croisée k-fois* (traduction de l'anglais k-fold cross validation) est une technique qui est couramment utilisée en apprentissage automatique pour déterminer les meilleurs hyperparamètres d'un algorithme d'apprentissage. On partitionne l'ensemble d'entrainement en k sous ensembles disjoints qui sont géneralement de même taille. La procédure pour déterminer les hyperparamètres consiste à exécuter l'algorithme d'apprentissage k fois et à chaque fois un ensemble de données de test est mis à côté pour déterminer le risque du classificateur généré par l'algorithme d'apprentissage sur les autres données qui ont servies d'entrainement. La figure 2.1 illustre une validation croisée 3-fois.
Considérons un ensemble d'entrainement S, partitionnée en k-sous ensembles disjoints $S_1, S_2, ..., S_k$ et un algorithme d'apprentissage A. Une validation croisée k fois va éxécuter l'algorithme A k fois, et à chaque fois un classificateur h_j est généré :

$$h_j = A(S \backslash S_j)$$

Le risque de validation croisée R_{Val} est déterminé en calculant la moyenne des risques des classificateurs $h_1, ..., h_k$:

$$R_{Val} = \frac{1}{k} \sum_{j=1}^{k} R_{S_j}(h_j)$$

Les meilleurs paramètres qui seront utilisés par l'algorithme d'apprentissage sont les hyperparamètres qui minimisent le risque de validation croisée.

2.10 Vote de majorité

Il existe des algorithmes en apprentissage automatique qui pour avoir de bons classificateurs, combinent des fonctions (que nous appelons parfois votants) ayant la plupart des

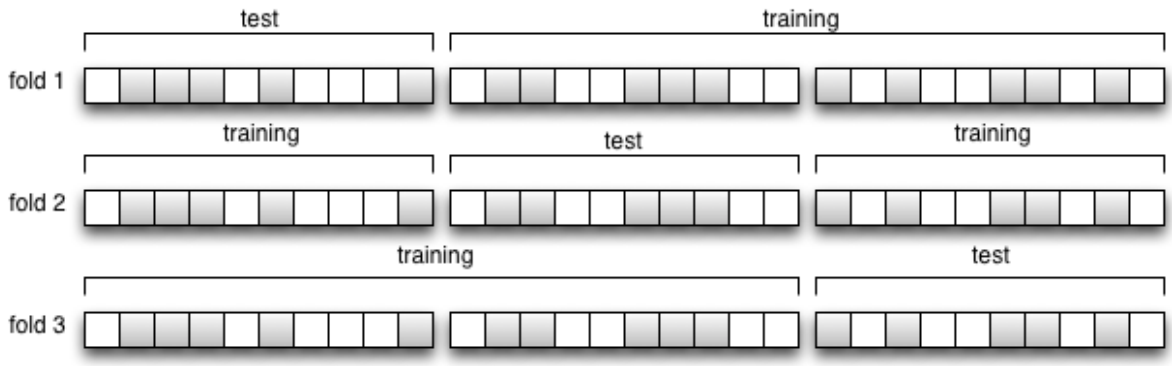


FIGURE 2.1 – Validation croisée 3-fois

fois une mauvaise performance individuelle. Ces algorithmes déterminent une pondération pour chacun de ces votants et un mécanisme de vote s'en suit pour déterminer le résultat final. Dans le cas de la classification, on utilise couramment le terme de **vote de majorité**. Le vote de majorité consiste à chercher la classe la plus majoritaire parmi les classes prédites par les classificateurs de la collection et de retourner cette classe comme la classe du vote de majorité. Si on suppose qu'on a un ensemble $\{h_1, .., h_n\}$ de n classificateurs, la valeur retournée par l'agrégation des classificateurs de la collection est donnée dans le cas binaire par :

$$H(\mathbf{x}) = \text{sgn}\Big(\sum_{i=1}^{n} \alpha_i h_i(\mathbf{x})\Big) \tag{2.1}$$

$$\text{avec } \alpha_i \in [0,1] \text{ et } \forall a \in \mathbb{R},\ \text{sgn}(a) \stackrel{\text{def}}{=} \begin{cases} +1 & \text{si } a > 0 \\ -1 & \text{sinon.} \end{cases}$$

Remarquons que sgn(0) est un cas indéterminé. Nous optons donc pour sgn(0)=-1. Signalons que sgn(0)=+1 est également valide dans notre cas.
Dans le cas de la classification multiclasse, l'expression du vote de majorité est donnée par :

$$H(\mathbf{x}) = \underset{c \in Y}{\text{argmax}} \Big(\sum_{i=1}^{n} \alpha_i I(h_i(\mathbf{x}) = c)\Big) \tag{2.2}$$

$$\text{avec } \alpha_i \in [0,1]$$

Si $\alpha_i = \frac{1}{n}$, on est en présence d'un vote de majorité uniformément pondéré ou vote démocratique, étant donné que les classificateurs h_i ont tous une même pondération $\frac{1}{n}$. Les algorithmes utilisant un vote de majorité sont souvent utilisés comme méthodes de référence pour obtenir de bons classificateurs.

2.11 La marge

Pour définir la marge, considérons un ensemble de classificateurs $\{h_1, h_2, ..., h_T\}$. D'une manière générale (cas multiclasse), la marge du vote de majorité H de ces classificateurs sur un exemple $(\mathbf{x}, y)$ est définie par :

$$\mathcal{M}_H(\mathbf{x}, y) = \frac{\sum_{t=1}^{T} I\Big(h_t(\mathbf{x}) = y\Big)}{T} - \max_{j \neq y} \frac{\sum_{t=1}^{T} I\Big(h_t(\mathbf{x}) = j\Big)}{T} \tag{2.3}$$

Comme le montre l'expression (2.3), la marge sur un exemple $(\mathbf{x}, y)$ est une mesure de la différence entre la moyenne des votes des classificateurs qui ne se trompent pas et ceux qui se trompent en retournant l'autre étiquette apparaissant le plus souvent. C'est donc une fonction comprise dans l'intervalle [-1, 1]. Si la marge est positive, le vote de majorité prédit la classe correcte sinon le vote de majorité prédit une classe incorrecte.

L'expression de marge exprimée par (2.3) peut être ramenée au cas binaire par :

$$\begin{aligned}
\mathcal{M}_H(\mathbf{x}, y) = & \quad \frac{\sum_{t=1}^{T} I\Big(h_t(\mathbf{x}) = y\Big)}{T} - \max_{j \neq y} \frac{\sum_{t=1}^{T} I\Big(h_t(\mathbf{x}) = j\Big)}{T} \\
= & \quad \frac{\sum_{t=1}^{T} I\Big(h_t(\mathbf{x}) = y\Big)}{T} - \frac{\sum_{t=1}^{T} I\Big(h_t(\mathbf{x}) \neq y\Big)}{T} \\
= & \quad \frac{\sum_{t=1}^{T} \Big[I\Big(h_t(\mathbf{x}) = y\Big) - I\Big(h_t(\mathbf{x}) \neq y\Big)\Big]}{T} \\
= & \quad \frac{\sum_{t=1}^{T} \Big(y\, h_t(\mathbf{x})\Big)}{T} \\
= & \quad \frac{y \sum_{t=1}^{T} h_t(\mathbf{x})}{T} \\
= & \quad y\Big(\frac{1}{T} \sum_{t=1}^{T} h_t(\mathbf{x})\Big)
\end{aligned}$$

Ramenée à l'expression de la dernière ligne, on peut facilement remarquer que le résultat de la classification n'est correct que si $\mathcal{M}_H(\mathbf{x}, y) > 0$. Pour éviter des cas de marge nulle causant l'indétermination dans le vote, on choisit souvent un nombre d'abres T impair. Quelques fois, on parle aussi de marge non signée. Celle-ci est exprimée par $\Big(\frac{1}{T} \sum_{t=1}^{T} h_t(\mathbf{x})\Big)$ dans notre exemple. Elle sert d'indicateur de confiance qu'on peut avoir dans un vote de majorité car si elle est élevée (positivement ou négativement), on est plus confiant dans les résultats du vote de majorité.

2.12 Méthodes d'ensemble

Les méthodes d'ensemble(15) sont des algorithmes d'apprentissage qui construisent une collection de classificateurs et effectuent une agrégation de leurs prédictions. Une approche de la vie courante qui s'apparente aux méthodes d'ensemble est le fameux dicton "l'union fait la force". Au lieu d'essayer d'améliorer un seul classificateur, les méthodes d'ensemble génèrent plusieurs classificateurs et mettent en commun leurs prédictions. L'idée derrière les méthodes d'ensemble est qu'en génèrant un ensemble de classificateurs au lieu d'un seul classificateur, on va explorer largement l'espace des solutions et la mise en commun de leurs prédictions ne pourra que mieux généraliser la relation entre les descriptions des exemples et leurs classes et de ce coup être meilleur que chacun des classificateurs de base. D'une manière générale, une méthode d'ensemble est caractérisé par :

– un espace d'hypothèses $\mathcal{H}$
– un algorithme d'apprentissage A produisant les hypothèses $h_1, ..., h_t \in \mathcal{H}$
– l'aggrégation des h_n par vote de majorité (section 2.10) pour la classification

Plusieurs auteurs (9; 16) ont déjà démontré l'utilité des méthodes d'ensemble dans l'amélioration des performances des classificateurs qui au départ donnaient de mauvaises perfomances. Dans les sections suivantes, on va introduire quelques méthodes d'ensemble couramment utilisées en apprentissage automatique : le Bagging(9) et le Boosting(32). Le tableau 2.1 contient un récapitulatif de la comparaison de ces deux méthodes.

Algorithme 1 Bagging(S, T)

Entrée : $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, l'ensemble d'entrainement.
Entrée : T, le nombre d'itérations.

```
pour t = 1, ..., T faire
    S_t ← ∅
    pour i = 1, ..., m faire
        tirer un exemple au hasard avec remise dans S et l'ajouter dans S_t.
    fin pour
    h_t = A(S_t)
fin pour
```

Sortie : $H(\mathbf{x}) = sgn\Big(\sum_{t=1}^{T} h_t(\mathbf{x})\Big)$.

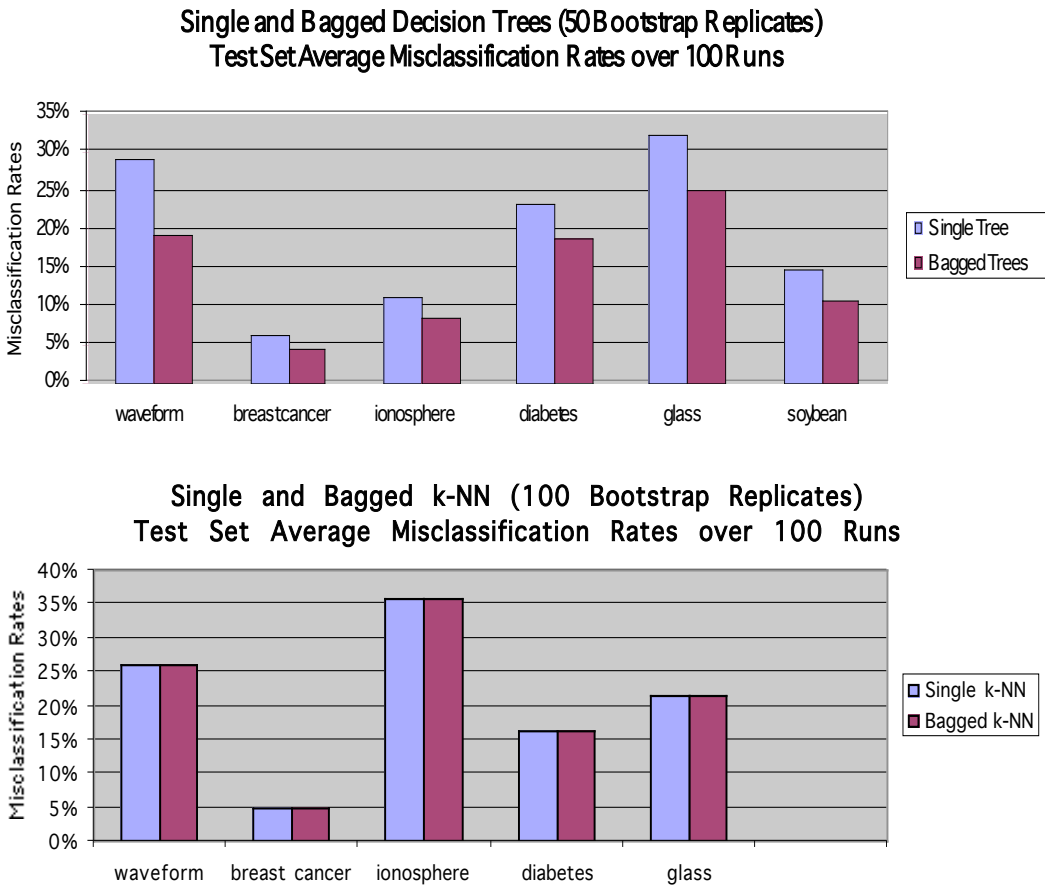


FIGURE 2.2 – Illustration du Bagging appliqué sur CART et k-NN. Breiman(1996)

2.12.1 Bagging

Le Bagging(9) est une méthode d'ensemble dont l'expression provient de la concaténation de "Bootstrap Aggregation". Elle utilise le rééchantillonage pour générer une collection de classificateurs qui participent par la suite à un vote de majorité. Pour illustrer la technique du Bagging, considérons l'algorithme 1 et supposons un ensemble d'entrainement $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$ composé de m tuples avec y_i étant une étiquette de classe pour la description $\mathbf{x}_i$ de l'exemple i, et un algorithme A permettant de génerer un classificateur h. Le principe du Bagging est de génèrer plusieurs échantillons aléatoires S_t couramment appelés "bootstrap". Chaque S_t contient m tuples tirés d'une manière aléatoire avec remise dans S. On applique par la suite l'algorithme A sur chaque S_t pour obtenir à la fin une collection de classificateurs h_t sur lesquels on applique un vote de majorité .

Il est démontré dans (9) que le facteur clé de la bonne performance du Bagging, c'est

l'**instabilité**. Un algorithme d'apprentissage est dit "instable" lorsqu'un changement mineur dans les données d'entrainement utilisé provoque un changement assez important dans la sortie de cet algorithme. Quelques algorithmes sont déjà connus pour leur instabilité notamment CART(7) (Classification and Regression Trees) qui est un algorithme générant des classificateurs de type arbre de décision qui sont détaillés dans le chapitre 3.
La figure 2.2 illustre l'application du Bagging sur deux algorithmes : un algorithme connu pour son instabilité à savoir CART, et un algorithme qui est stable : k-NN (k nearest neighbours) (8) ou règle des k plus proches voisins. Les résultats montrent que le Bagging fournit de bien meilleurs gains en performance de généralisation lorsqu'il est utilisé avec des algorithmes instables tandis qu'aucune amélioration n'est obtenue lorsqu'il est utilisé sur des algorithmes stables. Cependant, il a été également démontré (9) que le Bagging peut aussi marcher sur des algorithmes stables sur lesquels on introduit de l'instabilité en utilisant des méthodes aléatoires qui vont perturber les données.
La conclusion en est que l'instabilité joue un rôle important pour avoir une bonne performance du bagging.

2.12.2 Boosting

Le Boosting est une méthode d'ensemble qui a été introduite par Schapire(32). Comme le Bagging, il est utilisé pour améliorer la performance d'un algorithme d'apprentissage (diminuer le risque du classificateur issu de cet algorithme). Théoriquement, le Boosting peut améliorer significativement les performances de tout algorithme d'apprentissage, caractérisé comme étant **faible** c'est à dire un algorithme qui n'est assuré que de retourner un classificateur de risque élevé mais inférieur à 50%.
Le principe du Boosting est de combiner plusieurs classificateurs obtenus par un algorithme d'apprentissage faible en un classificateur de faible risque. Les algorithmes de type "boosting" effectuent une adaptation itérative des poids sur les classificateurs utilisés. Une illustration claire du Boosting est donnée par Adaboost, un algorithme d'usage très courant en apprentissage automatique, dû à sa simplicité et à ses grandes performances et dont le pseudo-code est détaillé par l'algorithme 2. L'expression "Adaboost" est une concaténation de "Adaptative Boosting". Il construit un vote de majorité itérativement de manière adaptative. A la fin de l'algorithme, chaque classificateur h_t est pondéré par une valeur α_t calculée dans la boucle de l'algorithme. La classification d'un nouvel exemple se fait en utilisant un vote de majorité pondéré.

Algorithme 2 AdaBoost(S, T)

Entrée : $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, l'ensemble d'entrainement.
Entrée : T, le nombre d'itérations.

Initialiser $W_1(i) \leftarrow 1/m$ pour $i = 1, \ldots, m$.
pour $t = 1, \ldots, T$ **faire**
Entrainer l'algorithme d'apprentissage faible : $h_t \leftarrow A_{\text{weak}}(S, W_t)$.
Calculer le risque empirique pondéré :

$$r_t \leftarrow \sum_{i=1}^{m} W_t(i)\ I(h_t(\mathbf{x}_i) \neq y_i)\,. \tag{2.4}$$

Calculer le poids [1]du classificateur dans le vote de majorité :

$$\alpha_t \leftarrow \frac{1}{2}\ \ln\Big(\frac{1 - r_t}{r_t}\Big). \tag{2.5}$$

Mettre à jour les poids des exemples (Z_t est une constante de normalisation pour que W_{t+1} soit une distribution sur les exemples) :

$$W_{t+1}(i) \leftarrow \frac{W_t(i) \exp\Big(-\alpha_t y_i h_t(\mathbf{x}_i)\Big)}{Z_t}\ . \tag{2.6}$$

fin pour

Sortie : $H(\mathbf{x}) = sgn\Big(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\Big)$.

Bagging	Boosting
Aléatoire	Adaptatif et généralement déterministe
Utilise des échantillons Bootstrap	Utilise échantillon initial au complet
Les modèles ont le même poids	Les modèles pondérés selon leur qualité d'ajustement

TABLE 2.1 – Bagging vs Boosting

2.12.3 Analyse sur l'efficacité d'Adaboost

Adaboost comme d'autres algorithmes de type Boosting permettent d'améliorer la performance d'un algorithme faible. Une analyse d'Adaboost montre qu'à l'étape t

1. si r_t=0, $\frac{1}{2}\ln\left(\frac{1-r_t}{r_t}\right)$ devient plus grand que α_{max}, on choisit alors $\alpha_t = \alpha_{max}$. Dans ce cas, le poids maximal permis α_{max} devient un hyperparamètre de l'algorithme.

($1 \leq t \leq T$ où T est un hyperparamètre spécifiant le nombre d'itérations), une nouvelle distribution sur les exemples W_t est calculée en fonction des résultats fournis par l'algorithme à l'étape précedente. A chaque étape t, Adaboost appelle l'algorithme d'apprentissage faible qui nous fournit en sortie un classificateur h_t. La performance de ce classificateur est calculée par le risque empirique pondéré r_t exprimé par (2.4). Chaque classificateur h_t est pondéré par α_t dans le vote de majorité H. α_t sera positif si $r_t \leq \frac{1}{2}$. Si $r_t > \frac{1}{2}$, α_t devient alors négatif dans ce cas c'est le classificateur $-h_t$ (classificateur inverse à h_t) qui est utilisé . Plus r_t est faible, plus h_t aura une pondération α_t élevée dans le vote de majorité H. L'expression (2.6) montre qu'Adaboost maintient une distribution des poids sur les exemples d'entrainement de manière à augmenter le poids des exemples qui sont mal classifiés et diminuer le poids des exemples bien classifiés. Si on considère le risque empirique r_t de h_t comme $r_t = \frac{1}{2} - \gamma_t$ où γ_t mesure l'amélioration apportée par h_t par rapport à l'erreur de base $\frac{1}{2}$ (les deux classes sont équiprobables et l'erreur d'une décision aléatoire est $\frac{1}{2}$), il est démontré dans (32) que :

$$R_S(H) \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp(-2\sum_{t=1}^{T} \gamma_t^2); \tag{2.7}$$

$$\text{et } R(H) = R_S(H) + \mathcal{O}(\sqrt{\frac{T.d_{\mathcal{H}}}{m}}) \tag{2.8}$$

où $d_{\mathcal{H}}$ est la dimension de Vapnik-Chervonenkis (6)(mesure de la compléxité de l'espace des hypothèses $\mathcal{H}$), ce qui signifie que le risque empirique du vote de majorité H a une borne supérieure constitué par la partie droite de l'expression (2.7). Ainsi si chaque hypothèse h_t est légèrement meilleur que le hasard ($\gamma_t \geq \gamma \geq 0$), alors le risque empirique $R_S(H)$ diminue exponentionellement rapidement avec t. Le vrai risque $R(H)$ est borné par une expression faisant intervenir le risque empirique $R_S(H)$, le nombre d'exemples m dans l'ensemble d'entrainement et la dimension de Vapnik-Chervonenkis $d_{\mathcal{H}}$ de l'espace d'hypothèses $\mathcal{H}$ ainsi que le nombre d'itérations T d'Adaboost. La borne suggère que le Boosting devrait tendre à surapprendre lorsque le nombre d'itérations T augmente puisque le deuxième terme de l'expression devient très grand, mais il a été démontré empiriquement que cela se manifeste après un nombre T très élévé d'itérations.

Chapitre 3

Les arbres de décision

3.1 Introduction

Les arbres de décision sont des classificateurs très couramment utilisés en apprentissage automatique. Leur popularité repose essentiellement sur plusieurs facteurs : ils sont simples, efficaces, facilement interpretables et possèdent une capacité à capturer des relations non linéaires entre les entrées et les sorties.
Leur efficacité a poussé la communauté de l'apprentissage automatique à développer plusieurs algorithmes qui génerent ces classificateurs dont le procédé de classification peut être structuré sous la forme d'un arbre. Parmi ces algorithmes, on peut citer quelques uns qu'on rencontre fréquemment dans la littérature comme CART(7), C4.5(31), qui est une version modifiée de ID3(30).

3.2 Construction d'un arbre de décision

Pour illustrer les arbres de décision, nous allons nous baser sur les arbres de décision binaire. Le principe général de ces derniers repose sur le partitionnement récursif de l'espace d'entrée $\mathcal{X}$ de façon binaire en cherchant la sous-partition optimale pour la classification. L'algorithme associe initialement la racine de l'arbre à tout l'espace d'entrée $\mathcal{X}$. A chaque étape du partitionnement, la partie de l'espace associée au noeud courant est découpée en deux sous-parties.
Pour la racine par exemple, après le premier partitionnement, elle possède deux noeuds fils, chacun correspondant à une sous-partie de l'espace d'entrée $\mathcal{X}$. Pour éclaircir le découpage, considérons un espace d'entrée $\mathcal{X}$ tel que $\mathcal{X} \subseteq \mathbb{R}^n$, et n étant le nombre

d'attributs. Considérons également le critère suivant :

$$\{x_j \leq t\} \cup \{x_j > t\}\,, \tag{3.1}$$

$$\text{avec } 1 \leq j \leq n \text{ et } t \in \mathbb{R}.$$

Pour construire l'arbre, l'algorithme commence par la racine en y appliquant le critère (3.1) qui signifie que la racine est partitionnée en deux noeuds : un noeud fils de gauche associé au sous-ensemble de $\mathcal{X}$ contenant toutes les instances de $\mathcal{X}$ dont l'attribut j est inférieur ou égal à un certain réel t et un noeud fils de droite correspondant au sous-ensemble de $\mathcal{X}$ contenant les instances supérieures à t. Ce procédé est répété pour chaque noeud de l'arbre jusqu'à un certain critère d'arrêt. Le critère d'arrêt peut varier. Ci-suivant, on présente deux différentes conditions couramment utilisées pour spécifier un critère d'arrêt. On arrête le développement de l'arbre :

– si tous les instances de l'ensemble d'entrainement associé à un noeud appartiennent à une même classe ;
– si une certaine profondeur maximum fixée d'avance a été atteinte.

D'une manière générale, le premier critère d'arrêt cité est celui qu'on rencontre couramment dans la littérature et le plus utilisé. Une fois qu'on a atteint le critère d'arrêt, on ne peut plus partitionner l'arbre. Celui-ci contient donc au bout, des noeuds terminaux. Ces noeuds terminaux sont appelés *feuilles*. En considérant le premier critère d'arrêt, un noeud devient terminal lorsque tous les exemples de ce noeud appartiennent à une même classe. On est donc en présence d'une "*feuille pure*". C'est cette classe qui sera retournée lors de la classification pour cette feuille. Cependant, s'il arrive qu'un autre critère soit utilisé, on attribue au noeud terminal la classe majoritaire : lorsque plusieurs classes sont en concurrences, on peut choisir la classe la plus représentée dans l'ensemble de l'échantillon. C'est cette classe majoritaire qui sera utilisée lors de la classification. Le procédé qui détermine la construction de l'arbre doit donc nécessairement assigner à chaque feuille une seule classe. Pour classifier un instance $\mathbf{x}$ quelconque, la sortie $h(\mathbf{x})$ d'un arbre de décision h sera déterminée par la classe de la feuille où aboutira l'exemple $\mathbf{x}$.

La figure 3.1 montre un partionnement d'un espace à deux dimensions et sa représentation sous la forme d'un arbre de décision. Les noeuds r_1, r_2, r_3, r_4 et r_5 correspondent aux feuilles de l'arbre.

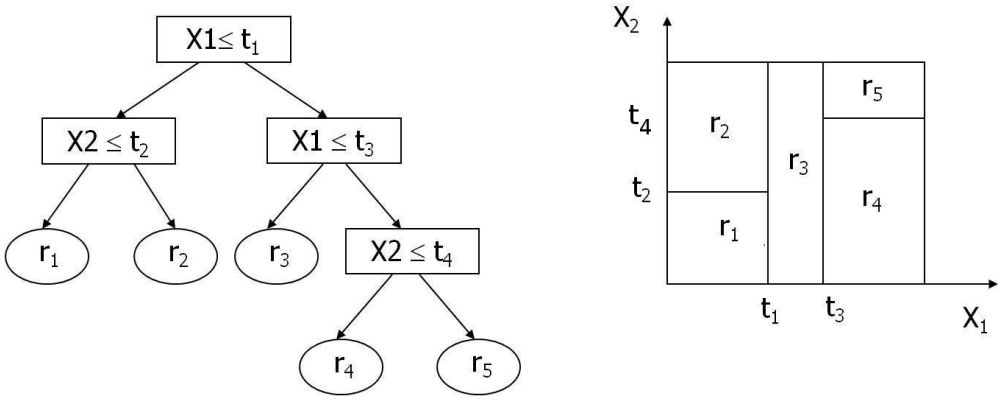


FIGURE 3.1 – Exemple d'un Arbre de décision associé à un espace à 2 dimensions

3.3 Algorithme d'induction et de classification

L'algorithme d'induction de l'arbre doit répondre principalement à deux questions :

1. **Comment effectuer le partitionnement de l'ensemble d'entrainement** : Chaque étape récursive du développement de l'arbre doit sélectionner les attributs sur lesquels effectuer le partitionnement. La meilleure collection d'attributs est celle qui est constituée par les attributs permettant d'induire l'arbre le plus petit (arbre ayant la plus petite profondeur à partir de la racine). L'algorithme doit donc spécifier une méthode pour déterminer sur quels attributs on doit effectuer le partitionnement ainsi qu'une mesure objective pour évaluer la qualité de cette méthode. Les mesures pour la détermination de l'attribut de partitionnement sont discutés à la section 3.4
2. **Quelle est la condition d'arrêt de la procédure de partitionnement** : Une condition d'arrêt est nécessaire pour signifier la fin du développement de l'arbre. La condition d'arrêt la plus souvent utilisée est que tous les exemples d'un noeud terminal doivent posséder une même classe.

L'algorithme 3 donne le pseudo-code de l'algorithme d'induction utilisé par un arbre de décision. L'entrée de l'algorithme est un ensemble d'entrainement S. L'algorithme construit les noeuds de l'arbre en sélectionnant d'une manière récursive l'attribut de partitionnement. La sortie de l'algorithme est un classificateur h. Pour classifier une nouvelle instance $\mathbf{x}$, h utilise l'algorithme 4. Dans le pseudo-code de l'algorithme 3, on fait appel aux fonctions suivantes :

– condition_arret(S) est une fonction qui détermine pour un ensemble S passé en paramètre si on a déjà atteint le critère d'arrêt (déterminer entre autres si les exemples de S appartiennent à une même classe).

– creerFeuille(S) est une fonction qui crée une structure de type feuille pour l'arbre. Elle recoit en paramètre un ensemble S dont les exemples appartiennent à la même classe et la feuille qui est construite, sauvegarde cette classe comme attribut de cette structure feuille. C'est cette classe qui sera retournée comme résultat de la classification de l'arbre pour un exemple qui aboutit dans cette feuille.
– creerNoeud($j, t, noeud_g, noeud_d$) est une fonction qui crée une structure de type noeud pour l'arbre. Elle reçoit en paramètres quatre arguments : j qui est l'indice de l'attribut de partitionnement pour le noeud qui sera créé, t la valeur du seuil pour le partitionnement de l'attribut d'indice j, $noeud_g$ et $noeud_d$ qui sont des pointeurs vers les noeuds fils de gauche et de droite respectivement. Pour un arbre ou sous-arbre h, on pourra accéder à ces attributs en utilisant la notation $h.j, h.t, h.noeud_g, h.noeud_d$ respectivement.

Dans le pseudo-code de l'algorithme 4, les fonctions suivantes ont été utilisées :
– estFeuille(h) est une fonction qui vérifie si l'arbre h passé en paramètre est une feuille. Si tel est le cas, pour classifier un nouvel exemple, la feuille retournera la valeur de son attribut *classe* qui détermine la classe de la feuille. Nous utilisons la notation $h.classe$ pour désigner l'attribut classe d'une feuille h.
– estNoeud(h) est une fonction qui vérifie si la racine de l'arbre h est un noeud qui n'est pas une feuille.

Algorithme 3 ArbreDecision(S)

Entrée : $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, l'ensemble d'entrainement.
Sortie : Un arbre de décision h

si condition_arret(S) **alors**
  retourner creerFeuille(S)
sinon
  Choisir une condition de test pour déterminer l'attribut j de partitionnement et la valeur t de partitionnement sur l'attribut j
  $S_g \leftarrow \{(\mathbf{x}, y) \in S : x_j <= t\}$
  $S_d \leftarrow \{(\mathbf{x}, y) \in S : x_j > t\}$
  $noeud_g \leftarrow$ ArbreDecision(S_g)
  $noeud_d \leftarrow$ ArbreDecision(S_d)
  retourner creerNoeud($j, t, noeud_g, noeud_d$)
fin si

Algorithme 4 classifier($\mathbf{x}, h$)

Entrée : $\mathbf{x}$, un nouvel instance à déterminer la classe.
Entrée : h, l'arbre de décision.

```
si estFeuille(h) alors
  retourner h.classe
sinon
  assert (estNoeud(h))
  x_j ← x[h.j]
  si x_j <= h.t alors
    retourner classifier(x, h.noeud_g)
  sinon
    retourner classifier(x, h.noeud_d)
  fin si
fin si
```

3.4 Mesures pour déterminer l'attribut de partitionnement

Il existe plusieurs mesures pour quantifier la qualité de la façon dont s'est effectuée le partitionnement de l'ensemble d'entrainement dans la création des noeuds de l'arbre. Ces mesures sont définies par rapport à la distribution des classes des exemples avant et après le partitionnement. Rappelons qu'un arbre de classification est d'autant meilleur que lorsque les instances de ses feuilles sont de classe homogène (les instances de chaque feuille ont la même classe). L'objectif de l'algorithme d'induction sera donc de maximiser l'homogéneité des feuilles de l'arbre tout en obtenant un arbre de la plus petite taille possible. En d'autres termes, l'algorithme doit chercher à avoir un arbre ayant le plus de feuilles "pures" (des feuilles avec le plus d'instances qui appartiennent à une même classe). Trois mesures sont couramment utilisées pour calculer le degré d'impureté d'un noeud : **l'indice de Gini, l'entropie**, et **l'erreur de classification**. Pour définir ces mesures, considérons les classes $1, ..., K$, avec K étant le nombre total de classes différentes qu'on trouve dans les m' instances présents dans un noeud N. Le noeud N est construit à partir d'un échantillon que nous notons S_N (avec $|S_N| = m'$), qui est un sous-ensemble de l'ensemble d'entrainement S.
La fréquence p_k relative à une classe k dans le noeud N ainsi que la classe k^* (classe majoritaire) qui maximise p_k sont définies par :

$$p_k = \frac{1}{m'} \sum_{(\mathbf{x},y)\in S_N} I(y = k)$$

$$k^* = \text{argmax}_k \ (\ p_k \ )$$

Pour un noeud N (qui peut être une feuille), l'index de Gini, l'entropie et l'erreur de classification sont alors définies comme suit :

$$\textbf{index de Gini : } g(N) = \sum_{k=1}^{K} \sum_{k \neq k'} p_k \ p_{k'}$$
$$= \sum_{k=1}^{K} p_k \ (1 - p_k)$$

$$\textbf{Entropie}\,^{1} \textbf{ : } e(N) = -\sum_{k=1}^{K} p_k \ln p_k$$

$$\textbf{Erreur de classification : } t(N) = \frac{1}{m} \sum_{(\mathbf{x},y) \in S_N} I(y \neq k^*)$$
$$= 1 - p_{k^*}$$

Les algorithmes ID3(30) et C4.5(31) utilisent l'entropie comme métrique pour déterminer l'attribut de partitionnement, tandis que CART(7) utilisent l'index de Gini. Dans le cas de la classification binaire, si p est la fréquence relative d'une classe, ces mesures se simplifient de la manière suivante :

$$g(N) = 2p \ (1 - p)$$
$$e(N) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$
$$t(N) = 1 - \max(p, 1 - p)$$

La figure 3.2 compare les trois mesures d'impureté pour la classification binaire. Les trois mesures atteignent leur maximum lorsque la distribution des classes est uniforme (lorsque la fréquence est $p = 0.5$). On observe le minimum lorsque les instances appartiennent à une même classe ($p = 0$ ou $p = 1$). Plus ces mesures sont petites pour un noeud, plus son degré d'impureté est faible. L'illustration de ces trois mesures est donnée par l'exemple du tableau 3.1.

Dans cette illustration, on propose un exemple où on a trois noeuds N_1, N_2, et N_3. Le noeud N_1 possède 8 exemples d'une même classe, et 0 exemple de l'autre classe. Le noeud N_2 possède une répartition de 2 exemples pour une classe et 6 exemples

1. ln correspond à la fonction de logartithme naturel et dans le calcul de l'entropie, $0 \ln_2 0 = 0$

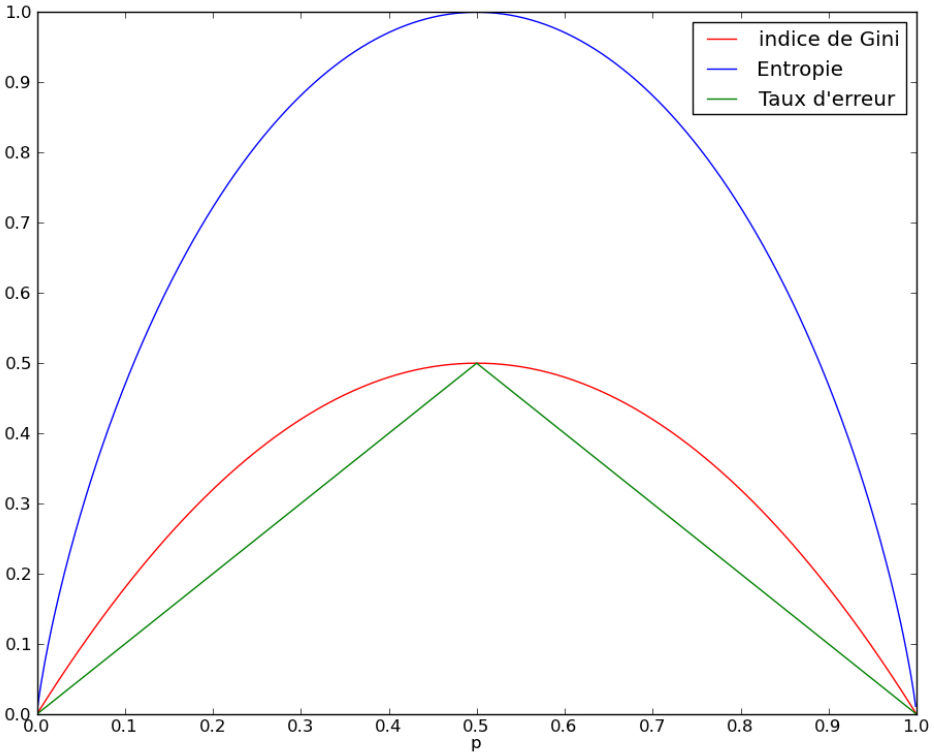


FIGURE 3.2 – Comparaison des mesures d'impureté pour la classification binaire

TABLE 3.1 – Répartition des classes pour les noeuds N_1, N_2, N_3

Noeud N_1	#Exemples
Classe=+1	0
Classe=-1	8

Noeud N_2	#Exemples
Classe=+1	2
Classe=-1	6

Noeud N_3	#Exemples
Classe=+1	4
Classe=-1	4

pour l'autre, tandis que le noeud N_3 possède 4 exemples pour chacune des classes. Le tableau 3.2 montre les résultats qu'on obtient en appliquant les trois mesures du degré d'impureté sur chacun de ces trois noeuds. On remarque donc que selon ces résultats le noeud N_1 possède le plus petit degré d'impureté, suivi de N_2 puis par N_3.

TABLE 3.2 – Mesures d'impureté des noeuds N_1, N_2, N_3

Noeud N_1	Résultat
$g(N_1)$	0
$e(N_1)$	0
$t(N_1)$	0

Noeud N_2	Résultat
$g(N_2)$	0.375
$e(N_2)$	0.811
$t(N_2)$	0.25

Noeud N_3	Résultat
$g(N_3)$	0.5
$e(N_3)$	1.0
$t(N_3)$	0.5

Lors de la construction de l'arbre par l'algorithme 3, l'attribut j de partitionnement pour un noeud N doit être choisi en tenant compte de la *pureté* que cela apportera à ses fils. L'algorithme doit maximiser, pour chaque noeud de l'arbre, un certain critère que nous notons Δ (se lit delta), qui est défini comme suit :

$$\Delta = f(N) - \sum_{t=1}^{n} \frac{c(N_t)}{c(N)} f(N_t) \tag{3.2}$$

où :
- f est la fonction (Gini, entropie ou erreur de classification) qui détermine la mesure $f(N)$ du degré d'impureté du noeud N,
- n correspond au nombre de noeuds fils (noeuds enfants) obtenus après le partitionnement sur chaque noeud parent. Selon n, on a l'habitude de dire qu'on a des arbres n-aires. Ainsi, avec $n = 2$, on a des arbres binaires,
- c est une fonction qui retourne le nombre d'instances qui sont associés à un noeud passé en paramètre,
- N_t représente le t-ème noeud fils obtenu après le partitionnement du noeud N.

Soit $G(N) = \sum_{t=1}^{n} \frac{c(N_t)}{c(N)} f(N_t)$. Le degré d'impureté du noeud parent N déterminé par $f(N)$ étant fixe, maximiser Δ revient à minimiser l'expression $G(N)$. Cela revient également à déterminer l'attribut de partitionnement qui permet de minimiser $G(N)$. Pour illustrer cela, considérons alors l'exemple d'un noeud N dont la répartition des classes est donnée par le tableau 3.3. Dans cet exemple on va considérer l'index de Gini comme mesure. On a donc $f = g$.

TABLE 3.3 – Répartition des classes pour un noeud parent N

Noeud N	#Exemples
Classe=+1	6
Classe=-1	6
$g(N) = 0.5$	

L'objectif consiste à déterminer parmi deux attributs A_1 et A_2 [2], lequel sera utilisé pour le partitionnement du noeud N. En partitionnant le noeud N, selon l'attribut A_1 on obtient les noeuds fils N_1 et N_2 et avec l'attribut A_2 on obtient les noeuds N_3 et N_4.

2. La notation A_j est utilisée pour déterminer la variable désignant l'attribut d'indice j

TABLE 3.4 – Répartition des classes pour les noeuds fils N_1, N_2

Classe	N_1	N_2
+1	4	2
-1	3	3
$G_1(N)$= 0.486		

Classe	N_3	N_4
+1	1	5
-1	4	2
$G_2(N)$ = 0.375		

Le tableau 3.4 montre les résultats qu'on obtient en considérant les partitionnements selon l'attribut A_1 pour le premier sous-tableau (gauche) et l'attribut A_2 pour le deuxième sous-tableau (droite). Nous utilisons aussi la notation $G_j(N)$ pour désigner la moyenne des indices de Gini des noeuds fils de N obtenus après le partitionnement selon l'attribut A_j. Dans le calcul de $G_j(N)$, l'indice de Gini de chacun des noeuds fils de N est pondéré par la proportion des exemples le constituant sur le nombre total des exemples du noeud parent. Nous détaillons les calculs effectués pour arriver aux résultats obtenus dans le tableau 3.4. En considérant l'attribut A_1, on obtient :

- $g(N_1) = 2 \cdot \frac{4}{7}(1 - \frac{4}{7}) = 0.4898$
- $g(N_2) = 2 \cdot \frac{2}{5}(1 - \frac{2}{5}) = 0.48$
- $G_1(N) = \frac{7}{12} \cdot 0.4898 + \frac{5}{12} \cdot 0.48 = 0.486$

De même, si on applique le même principe pour l'attribut A_2, on obtient $G_2(N) = 0.375$. Si on avait donc à choisir, l'attribut A_2 est donc celui qui serait choisi étant donné que c'est celui qui minimise le degré d'impureté.

3.5 Détermination du seuil de partitionnement pour un attribut

Dans la section précédente, on a décrit comment on choisit l'attribut de partitionnement. Cependant, rappelons que dans l'algorithme 3, on doit déterminer un seuil t de partitionnement sur l'attribut qui a été choisi. Une première approche consiste à utiliser une stratégie par force brute pour choisir chaque valeur unique de l'attribut de partitionnement dans les m exemples comme candidat à la valeur t du seuil de partitionnement. Pour chaque candidat, on calcule l'indice de Gini et le candidat qui donne la plus petite valeur de l'indice de Gini est choisi comme valeur de seuil t de partitionnement. Ainsi, s'il y a un nombre n d'attributs et si chacun des n attributs possède k valeurs possibles cela donne nk partitionnements possibles. Cela nécessite donc $O(nk)$ opérations pour

calculer l'indice de Gini pour chaque candidat et comme il y aurait nk candidats possibles cela impliquerait une complexité de $O(n^2k^2)$. Cette approche peut être efficace dans le cas où la majorité des attributs sont binaires donc avec $k = 2$. Cependant, dans certains cas, si on est en présence d'attributs qui ne sont pas binaires, généralement des attributs avec des valeurs numériques, le nombre k de valeurs uniques peut s'avérer éléver. Dans ce cas, pour réduire cette complexité l'approche qui est couramment utilisée consiste à trier d'abord les valeurs uniques de l'attribut de partitionnement parmi les k valeurs de cet attribut. Les candidats pour la valeur du seuil t de partitionnement sont détérminés par les milieux obtenus en prenant les valeurs uniques adjacentes deux à deux de l'attribut qu'on a après le tri.

3.6 Performance et/ou complexité des arbres

Dans la littérature, il existe beaucoup de sortes d'arbres de décision, les uns diffèrent des autres principalement par les critères d'arrêt et de partitionnement. Il a été empiriquement démontré (12) que le critère de partitionnement n'influence pas significativement la capacité de prédiction de l'arbre mais plutôt le contrôle de sa complexité. En effet, les arbres complexes (très larges) ont tendance à surapprendre. Une mesure de complexité pour un arbre de décision serait par exemple de considérer le nombre de ses feuilles. Une méthode générale pour contrôler la complexité d'un arbre consiste à attribuer une penalité en fonction du nombre de feuilles de l'arbre. Pour minimiser cette pénalité, on utilise souvent **l'élagage** qui consiste à chercher le sous-arbre qui minimise l'erreur de généralisation. Ce sous-arbre est obtenu en réduisant certaines branches de l'arbre de manière à augmenter la capacité de généraliser ses prédictions à de nouveaux exemples. Cependant, la question de l'élagage mérite une attention particulière. En effet, dans la littérature, certains auteurs conseillent d'effectuer l'élagage et d'autres suggèrent de ne pas élaguer. Ceux qui proposent de ne pas élaguer les arbres, le justifient souvent par le fait que l'élagage n'apporte pas d'amélioration considérable sur la capacité de généraliser les prédictions à de nouveaux exemples et augmente significativement le temps de calcul. A l'opposé, ceux qui proposent l'élagage suggèrent qu'élaguer les arbres peut s'avérer intéressant (5; 4; 3). Certains algorithmes de l'état de l'art tels que CART utilisent l'élagage et font un compromis lors de la procédure d'induction de l'arbre entre la performance (l'erreur de prédiction de l'arbre sur l'ensemble d'entrainement) et sa complexité. Dans la suite de ce mémoire, les arbres de décision qui seront utilisés ne seront pas élagués. Dans un premier temps, ils seront induits suivant l'algorithme CART jusqu'à la profondeur maximale sans élaguage. Pour des objectifs d'expérimentations,

toujours en suivant CART, on limitera peut être leur profondeur, mais l'élaguage ne sera pas encore une fois effectué. En effet, l'algorithme des fôrets aléatoires que nous allons introduire dans le chapitre suivant, utilise des arbres non élagués tel que son auteur l'a préconisé. L'élagage permettant de réduire le surapprentissage, cet algorithme utilise d'autres stratégies pour ne pas surapprendre sans toutefois recourir à l'élagage car cela lui permet d'être plus résistant lorsqu'on traite des données bruitées.

Chapitre 4

Les Forêts Aléatoires

4.1 Introduction

Les méthodes d'ensemble constituent une famille ou ensemble d'algorithmes qui génèrent une collection de classificateurs pour par la suite les combiner en aggrégeant leurs prédictions. L'efficacité de la combinaison des classificateurs repose principalement sur leur capacité à tirer les complémentarités des classificateurs individuels dans le but d'améliorer autant que possible les performances (capacité de prédiction) en généralisation de l'ensemble. Une explication de ce lien entre complémentarité et performance est donnée par la notion de *diversité*. Bien que la littérature ne fournisse pas une définition formelle sur laquelle tout le monde s'accorde de ce qu'est la diversité (21), ce concept est reconnu comme étant l'une des plus importantes caractéristiques pour l'amélioration des performances en généralisation d'un ensemble de classificateurs (22). D'une manière usuelle, la diversité peut être définie comme la capacité des classificateurs individuels d'un ensemble à être en accord sur les bonnes prédictions et en désaccord sur les erreurs de prédiction.
Le bagging détaillé à la section 2.12.1 est une méthode d'ensemble qui, à partir des "bootstraps", génère des classificateurs et un vote de majorité s'en suit pour déterminer le résultat de la classification. Le bagging s'appuie sur l'aléatoire dans la génération des bootstraps dans le but de construire des classificateurs de base différents les uns des autres et d'ainsi introduire la diversité dans l'ensemble.
Les forêts aléatoires qui font l'objet de ce chapitre constituent l'un des plus récents algorithmes de la famille des méthodes d'ensembles. Introduites par Leo Breiman (11), elles se basent aussi sur l'aléatoire ou la randomisation. Elles effectuent une modification substantielle du bagging (pour introduire la diversité) afin de produire une collection

d'arbres de décision "*décorrelés*" et un vote de majorité s'en suit pour déterminer la classe majoritaire .

4.2 Définition des forêts aléatoires

Considérons un ensemble d'entrainement $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$, a le nombre d'attributs des exemples de $\mathcal{X}$. Considérons également S_t un bootstrap (voir Section 2.12.1) contenant m instances obtenus par rééchantillonage avec remplacement de S. Soit $\{h_1, \ldots, h_T\}$ un ensemble de T arbres de décision. Chaque arbre h_t est construit à partir de S_t. Pour chaque noeud de l'arbre, l'attribut de partitionnement est choisi en considérant un nombre f ($f << a$) d'attributs choisis aléatoirement (parmi les a attributs). Pour classifier une nouvelle instance $\mathbf{x}$, le classificateur des forêts aléatoires effectue un vote de majorité uniformément pondéré des classificateurs de cet ensemble pour l'instance $\mathbf{x}$. L'algorithme 5 illustre cette définition.

Algorithme 5 RandomForest(S, T)

Entrée : $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, l'ensemble d'entrainement.
Entrée : T, le nombre d'arbres de décision de la forêt aléatoire.

pour $t = 1, \ldots, T$ **faire**

1. Générer un échantillon bootstrap S_t de taille m à partir de S
2. Construire un arbre de décision h_t à partir de S_t en répetant d'une manière récursive pour chaque noeud de l'arbre les étapes suivantes :
   a) Sélectionner aléatoirement f attributs parmi les a attributs
   b) Choisir l'attribut de partitionnement parmi les f
   c) Partitionner le noeud en deux noeuds fils

fin pour

Sortie : H, le classificateur de la forêt aléatoire

.

En considérant le classificateur H fourni par l'algorithme 5, la classification d'un nouvel exemple $(\mathbf{x}, y)$ s'effectue suivant l'expression suivante :

$$H(\mathbf{x}) = \underset{c \in Y}{\operatorname{argmax}} \Big(\sum_{t=1}^{T} \frac{1}{T} I(h_t(\mathbf{x}) = c) \Big) \tag{4.1}$$

Tout comme pour le bagging, chaque arbre de décision h_t est construit à partir d'un échantillon bootstrap S_t. Cependant une modification substantielle du bagging est effectuée par les forêts aléatoires. En effet, lors du partitionnement d'un noeud, le meilleur

attribut de partitionnement est choisi parmi un sous-ensemble f ($f << a$) d'attributs choisis aléatoirement à chaque noeud parmi les a attributs possibles. La différence avec le bagging s'observe donc à ce point puisque le bagging considère tous les a attributs possibles pour en choisir le meilleur.
Les arbres qui sont construits par les forêts aléatoires ne sont pas élagués. Ainsi, l'algorithme résultant est plus robuste face à des données entâchées de bruit. Pour la classification, un vote de majorité démocratique est effectué pour déterminer la classe majoritaire comme c'est décrit par l'expression (4.1).

4.3 Convergence d'une forêt aléatoire

En considérant une distribution de probabilités D sur $\mathcal{X} \times \mathcal{Y}$ qui génère les exemples, le vrai risque d'un classificateur h tel qu'il est défini dans la section 2.6 est donné par :

$$R(h) = \underset{(\mathbf{x},y)\sim D}{\mathbf{E}} I\Big(h(\mathbf{x}) \neq y\Big)$$

En faisant intervenir la notion de marge (voir section 2.11), le vrai risque peut être réecrit par :

$$R(h) = \underset{(\mathbf{x},y)\sim D}{\mathbf{E}} I\Big(\mathcal{M}_h(\mathbf{x},y) \leq 0\Big)$$

Dans le cas des forêts aléatoires, considérons alors un espace de classificateurs (arbres de décision) $\mathcal{H}$ et **une distribution uniforme** Q sur l'espace $\mathcal{H}$. Pour un nombre T d'arbres et H le classificateur de la forêt aléatoire obtenu par vote de majorité de ces T arbres tel que décrit par l'algorithme 5, on a alors :

$$R(H) = \underset{(\mathbf{x},y)\sim D}{\mathbf{E}} I\Big(\mathcal{M}_H(\mathbf{x},y) \leq 0\Big)$$

Il a été démontré dans (11) que lorsque le nombre d'arbres d'une forêt aléatoire tend vers l'infini, l'erreur de généralisation de la forêt tend vers l'erreur de la forêt continue ou forêt infinie (forêt avec un nombre d'arbres infini). Ceci peut être représenté par l'expression suivante :

$$R(H) \rightarrow \underset{(\mathbf{x},y)\sim D}{\mathbf{E}} \underset{h\sim Q}{\mathbf{E}} I\Big(\mathcal{M}_h(\mathbf{x},y) \leq 0\Big) \tag{4.2}$$

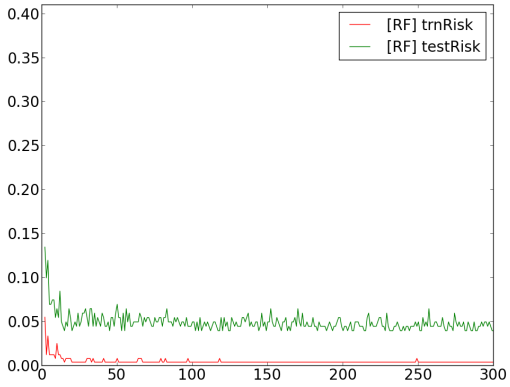
Il s'agit de la propriété de *convergence* des forêts aléatoires. Cela explique pourquoi les forêts aléatoires ne font pas de surapprentissage lorsque le nombre d'arbres de la forêt augmente mais plutôt convergent vers une valeur limite de l'erreur de généralisation.

Pour cette raison, Breiman a déterminé une borne supérieure pour l'erreur de généralisation des forêts aléatoires. Nous illustrons le calcul de cette borne dans la section 4.4. La figure 4.1 montre les résultats d'une exécution empirique de l'algorithme des forêts aléatoires sur quelques ensembles de données UCI (Université de California à Irvine). Les détails sur ces ensembles de données sont spécifiés dans l'annexe A. Les colonnes $|S|$ et $|T|$ dans l'annexe A représentent les cardinalités de l'ensemble des données d'entrainement et de l'ensemble des données test respectivement. Pour exécuter l'algorithme des forêts aléatoires, on a utilisé un nombre f d'attributs, tel que $f = \lfloor \log_2 a + 1 \rfloor$ avec a étant le nombre d'attributs de l'ensemble des données. La figure 4.1 illustre donc les résultats de cette exécution. Les axes des abscisses indiquent le nombre d'arbres de la forêt aléatoire et les axes des ordonnées indiquent le taux d'erreur (risque empirique). Pour chaque point de l'axe des abscisses qui correspond au nombre d'arbres de la fôret, le résultat est obtenu en calculant la moyenne des risques empiriques obtenu après avoir roulé 100 fois l'algorithme des forêts aléatoires. Les courbes vertes correspondent au risque sur l'ensemble test et les rouges au risque sur l'ensemble d'entrainement. On remarque que pour la majorité des ensembles de données même quand le risque sur l'ensemble d'entrainement a atteint 0, le risque sur l'ensemble test a tendance à se stabiliser à partir d'un certain nombre d'arbres et que même lorsqu'on dépasse ce nombre, le risque sur l'ensemble test ne monte pas pour autant mais plutôt reste stable.

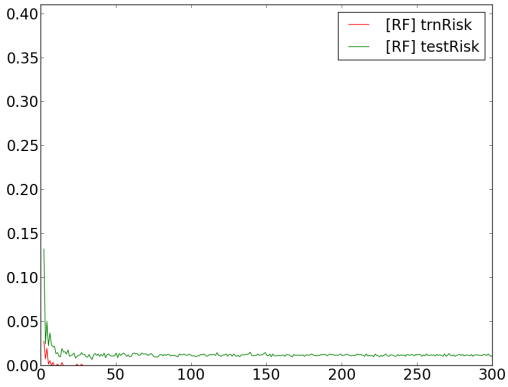
4.4 Une borne pour le risque des forêts aléatoires

Avec l'algorithme forêts aléatoires, Breiman a introduit deux notions importantes : *la force* et *la corrélation*. Pour les définir, il a utilisé une notation générale qui s'applique pour la classification multiclasse et la classification binaire. Dans notre cas, on va effectuer quelques modifications mineures par rapport à sa façon de faire : premièrement on va modifier sa notation pour standardiser notre notation avec les définitions faites dans les chapitres et sections précédents (ainsi que les chapitres suivants) et deuxièmement, pour cette section nous allons nous restreindre à la classification binaire. Le lecteur interessé pourra retrouver la démonstration (généralement faite dans le contexte multiclasse) et la notation de Breiman dans (11).
Considérons un espace de classificateurs (arbres de décision) $\mathcal{H}$ et **une distribution uniforme** Q sur l'espace $\mathcal{H}$. La marge[1] suivant la distribution uniforme Q sur un

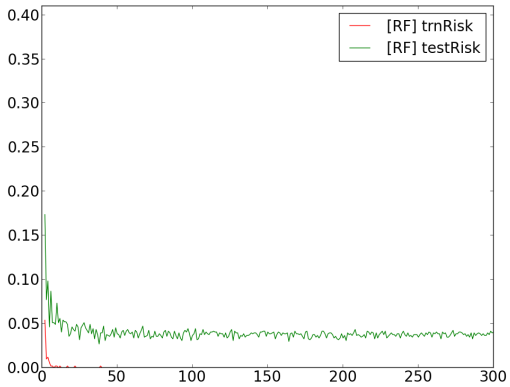
1. Cette définition est aussi valable lorsque Q n'est pas une distribution uniforme.



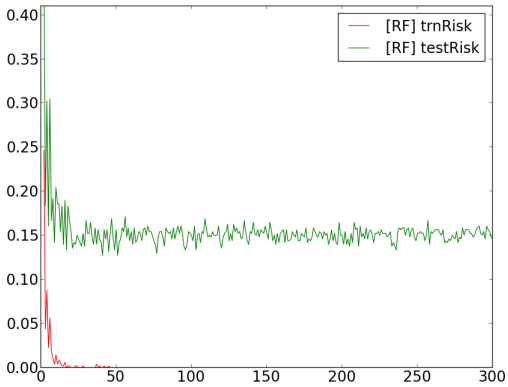
(a) USvotes



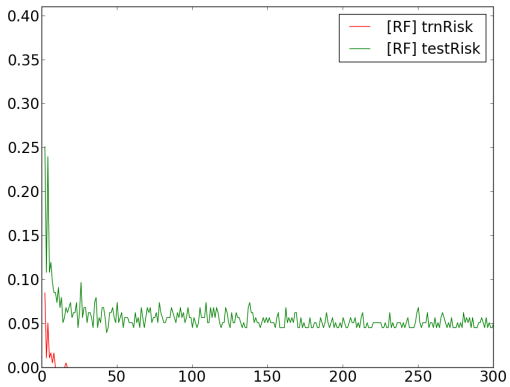
(b) Mnist_17



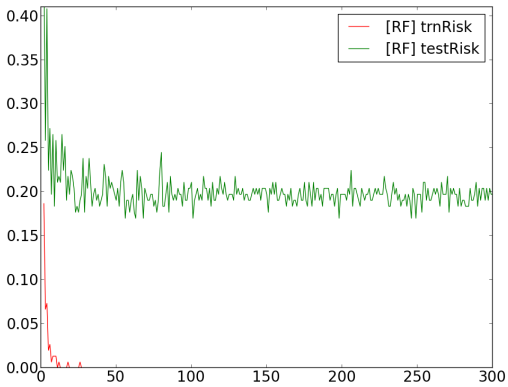
(c) Letter_OQ



(d) Tic-Tac-Toe



(e) Ionosphere



(f) Heart

FIGURE 4.1 – Risque sur l'ensemble test (testRisk) et risque sur l'ensemble d'entrainement (trnRisk) en fonction du nombre d'arbres dans une forêt

exemple $(\mathbf{x}, y)$ est définie comme suit :

$$\mathcal{M}_Q(\mathbf{x}, y) \stackrel{\text{def}}{=} y \cdot \mathbf{E}_{h\sim Q}\, h(\mathbf{x})\,.$$

La force [2] $\mathcal{M}_Q^D$ ou *premier moment de la marge* (20) est définie comme suit :

$$\mathcal{M}_Q^D \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} \mathcal{M}_Q(\mathbf{x}, y) = \mathop{\mathbf{E}}_{h\sim Q} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} y\, h(\mathbf{x})\,. \tag{4.3}$$

La force $\mathcal{M}_Q^D$ telle que Breiman la définit est donc l'espérance de la marge du classificateur des forêts aléatoires selon la distribution D qui génère les exemples. Considérons aussi le *deuxième moment de la marge* $\mathcal{M}_{Q^2}^D$:

$$\mathcal{M}_{Q^2}^D \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} \Big(\mathcal{M}_Q(\mathbf{x}, y)\Big)^2 = \mathop{\mathbf{E}}_{(h,h')\sim Q^2} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} h(\mathbf{x})\, h'(\mathbf{x})\,.$$

Remarquez que comme $y^2 = 1$ (classification binaire), il n y a pas d'étiquette y présente dans la dernière équation.

Breiman se sert aussi de l'inégalité de Tchebychev. Pour la définir, évoquons d'abord l'inégalité de Markov [3] :

Lemme 1 (Inégalité de Markov)**.** *Pour toute variable aléatoire non négative X dont l'espérance est μ et pour tout $a > 0$, on a :*

$$\Pr\ (X \geq a) \leq \frac{\mu}{a}\ .$$

Evoquons aussi l'inégalité de Tchebychev (qui sera aussi utilisée) qui s'énonce alors comme suit :

Lemme 2 (Inégalité de Tchebychev)**.** *Soit X une variable aléatoire d'espérance μ et de variance σ^2. Soit $a \geq 0$. Alors*

$$\Pr\left(|(X - \mu)| \geq a\right) \leq \frac{\sigma^2}{a^2}.$$

2. force est une traduction de *strength* qui est le terme utilisé par Breiman et il utilise pour cela la notation s

3. nous nous contentons de l'énoncer, sa démonstration peut être trouvée sur ce lien `http://fr.wikipedia.org/wiki/In%C3%A9galit%C3%A9_de_Markov`

Démonstration.

$$\begin{aligned}\Pr\left(|(X-\mu)| \geq a\right) &= \Pr\left((X-\mu)^2 \geq a^2\right) \\ &\leq \frac{\mathbf{E}\left((X-\mu)^2\right)}{a^2} \\ &= \frac{\mathbf{Var}\ X}{a^2} \\ &= \frac{\sigma^2}{a^2}\end{aligned}$$

□

Dans la démonstration ci-dessus, le passage de la première ligne à la deuxième ligne s'effectue grâce à l'inégalité de Markov énoncée par le lemme 1.
En utilisant l'inégalité de Tchebychev, Breiman(11) détermine une borne supérieure sur le vrai risque du classificateur H (vote de majorité des arbres) de la forêt aléatoire. En effet, en supposant que $\mathcal{M}_Q^D \geq 0$ et en remplaçant X par la variable aléatoire $\mathcal{M}_Q(\mathbf{x}, y)$ et a par $\mathcal{M}_Q^D$ dans l'inégalité de Tchebychev, on a :

$$\Pr\left((\mathcal{M}_Q(\mathbf{x}, y) - \mathcal{M}_Q^D) \leq -\mathcal{M}_Q^D\right) \leq \frac{\mathbf{Var}_{(\mathbf{x},y)\sim D}\left(\mathcal{M}_Q(\mathbf{x}, y)\right)}{(\mathcal{M}_Q^D)^2} \tag{4.4}$$

Or :

$$\Pr\left((\mathcal{M}_Q(\mathbf{x}, y) - \mathcal{M}_Q^D) \leq -\mathcal{M}_Q^D\right) = \Pr\left(\mathcal{M}_Q(\mathbf{x}, y) \leq 0\right) \tag{4.5}$$

$$= R(H) \tag{4.6}$$

De (4.4) et (4.6), on a :

$$R(H) \leq \frac{\mathbf{Var}_{(\mathbf{x},y)\sim D}\left(\mathcal{M}_Q(\mathbf{x}, y)\right)}{(\mathcal{M}_Q^D)^2} \tag{4.7}$$

Rappelons que la marge $\mathcal{M}_Q(\mathbf{x}, y)$ sur l'exemple $(\mathbf{x}, y)$ peut être réécrite comme suit :

$$\mathcal{M}_Q(\mathbf{x}, y) = \mathop{\mathbf{E}}_{h\sim Q}\left[I\Big(h(\mathbf{x}) = y\Big) - I\Big(h(\mathbf{x}) \neq y\Big)\right]$$

Considérons alors la définition suivante :

$$\mathcal{B}_h(\mathbf{x}, y) \stackrel{\text{def}}{=} I\Big(h(\mathbf{x}) = y\Big) - I\Big(h(\mathbf{x}) \neq y\Big)$$

On a alors :

$$\mathcal{M}_Q(\mathbf{x}, y) = \mathop{\mathbf{E}}_{h\sim Q}\Big(\mathcal{B}_h(\mathbf{x}, y)\Big)$$

De cela, il en résulte que :

$$\mathcal{M}_Q(\mathbf{x}, y)^2 = \mathop{\mathbf{E}}_{(h,h')\sim Q^2}\Big(\mathcal{B}_h(\mathbf{x}, y)\mathcal{B}_{h'}(\mathbf{x}, y)\Big)$$

Considérons alors la notion d'écart-type σ_x d'une variable x. Considérons également les notions de covariance et de coefficient de corrélation entre les variables x et y notées respectivement par $\mathrm{cov}\,(x, y)$ et $\mathrm{corr}\,(x, y)$ tel que :

$$\begin{aligned}
\sigma_x &= \sqrt{\big(x - \mathrm{E}(x)\big)} \\
\mathrm{cov}\,(x, y) &= \mathrm{E}\big[\big(x - \mathrm{E}(x)\big)\big(y - \mathrm{E}(y)\big)\big] \\
\mathrm{corr}\,(x, y) &= \frac{\mathrm{cov}\,(x, y)}{\sigma_x \cdot \sigma_y}
\end{aligned}$$

On a donc :

$$\begin{aligned}
\mathbf{Var}_{(\mathbf{x},y)\sim D}\Big(\mathcal{M}_Q(\mathbf{x}, y)\Big) &= \mathop{\mathbf{E}}_{(h,h')\sim Q^2}\ \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D}\Big[\mathrm{cov}\Big(\mathcal{B}_h(\mathbf{x}, y), \mathcal{B}_{h'}(\mathbf{x}, y)\Big)\Big] \\
&= \mathop{\mathbf{E}}_{(h,h')\sim Q^2}\ \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D}\Big[\mathrm{corr}\Big(\mathcal{B}_h(\mathbf{x}, y), \mathcal{B}_{h'}(\mathbf{x}, y)\Big)\sigma_h\sigma_{h'}\Big]
\end{aligned}$$

avec σ_h et $\sigma_{h'}$ correspondant à l'écart type entre les variables $\mathcal{B}_h(\mathbf{x}, y)$ et $\mathcal{B}_{h'}(\mathbf{x}, y)$ respectivement.
On a donc :

$$\begin{aligned}
\mathbf{Var}_{(\mathbf{x},y)\sim D}\Big(\mathcal{M}_Q(\mathbf{x}, y)\Big) &= \mathrm{Corr}^{\,D}_{Q^2}(h, h')\Big(\mathop{\mathbf{E}}_{h\sim Q}\ \sigma_h\Big)^2 && (4.8)\\
&\leq \mathrm{Corr}^{\,D}_{Q^2}(h, h')\Big[\mathop{\mathbf{E}}_{h\sim Q}\ \mathbf{Var}_{(\mathbf{x},y)\sim D}\Big(\mathcal{B}_h(\mathbf{x}, y)\Big)\Big] && (4.9)
\end{aligned}$$

où :

$$\mathrm{Corr}^{\,D}_{Q^2}(h, h') \stackrel{\text{def}}{=} \frac{\mathop{\mathbf{E}}_{(h,h')\sim Q^2}\ \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D}\Big[\mathrm{corr}\Big(\mathcal{B}_h(\mathbf{x}, y), \mathcal{B}_{h'}(\mathbf{x}, y)\Big)\sigma_h\sigma_{h'}\Big]}{\mathop{\mathbf{E}}_{(h,h')\sim Q^2}\Big(\sigma_h\sigma_{h'}\Big)}$$

Sachant que :

$$\begin{aligned}
\mathop{\mathbf{E}}_{h\sim Q}\ \mathbf{Var}_{(\mathbf{x},y)\sim D}\Big(\mathcal{B}_h(\mathbf{x}, y)\Big) &\leq \mathop{\mathbf{E}}_{h\sim Q}\Big[\mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D}\Big(\mathcal{B}_h(\mathbf{x}, y)\Big)\Big]^2 - (\mathcal{M}_Q^D)^2 && (4.10)\\
&\leq 1 - (\mathcal{M}_Q^D)^2 && (4.11)
\end{aligned}$$

En combinant les équations (4.7), (4.9) et (4.11), nous avons :

$$R(H) \leq \frac{\text{Corr}\,_{Q^2}^{D}(h,h')\Big(1-(\mathcal{M}_Q^D)^2\Big)}{(\mathcal{M}_Q^D)^2} \tag{4.12}$$

L'expression (4.12) correspond à la borne du risque du classificateur des forêts aléatoires telle que Breiman l'a déterminée. Cette borne dépend de la distribution D (et non des valeurs empiriques) et elle permet de déterminer l'enjeu dans l'induction des forêts aléatoires. Elle signale que pour avoir une forêt avec un bon risque, la forêt devra avoir des arbres les plus performants possibles d'une manière individuelle et les moins corrélés possibles en termes de prédictions.
Cependant, Breiman signale que cette borne semble être assez peu précise mais qu'elle fournit les éléments nécessaires pour la compréhension des fôrets aléatoires. Pour faire face à la tendance de la borne à être lâche, dans (20) et (23) [4], au lieu d'utiliser l'inégalité de Chybechev, ils utilisent l'inégalité de Cantelli-Chebychev (14) qui permet d'avoir une borne supérieure encore plus serrée. En effet, il a été démontré dans (2) que la borne de Cantelli-Chebychev est la borne supérieure la plus serrée possible pour une variable aléatoire basée sur son espérance et sa variance.

Théorème 3 (Inégalité de Cantelli-Tchebychev)**.** *Soit X une variable aléatoire d'espérance μ et de variance σ^2. Soit $a \geq 0$. Alors*

$$\Pr\left(X \geq a+\mu\right) \leq \frac{\sigma^2}{\sigma^2+a^2}.$$

4. dans cette source, il s'agit de bornes PAC (voir chapitre 5) qui sont utilisées

Démonstration.

$$\begin{aligned}
\Pr\left(X \geq a+\mu\right) &= \Pr\left(X-\mu+\frac{\sigma^2}{a} \geq a+\frac{\sigma^2}{a}\right) \\
&\leq \Pr\left(\left(X-\mu+\frac{\sigma^2}{a}\right)^2 \geq \left(a+\frac{\sigma^2}{a}\right)^2\right) \\
&\leq \frac{\mathbf{E}\left(X-\mu+\frac{\sigma^2}{a}\right)^2}{\left(a+\frac{\sigma^2}{a}\right)^2} \\
&= \frac{\mathbf{E}\ X^2+\mu^2+\frac{\sigma^4}{a^2}-2\mu\mathbf{E}\ X+2\frac{\sigma^2}{a}\mathbf{E}\ X-2\mu\frac{\sigma^2}{a}}{a^2+2\sigma^2+\frac{\sigma^4}{a^2}} \\
&= \frac{\mathbf{E}\ X^2-\mu^2+\frac{\sigma^4}{a^2}}{(1+\frac{\sigma^2}{a^2})(\sigma^2+a^2)} \\
&= \frac{\sigma^2+\frac{\sigma^4}{a^2}}{(1+\frac{\sigma^2}{a^2})(\sigma^2+a^2)} \\
&= \frac{\sigma^2(1+\frac{\sigma^2}{a^2})}{(1+\frac{\sigma^2}{a^2})(\sigma^2+a^2)} \\
&= \frac{\sigma^2}{\sigma^2+a^2}.
\end{aligned}$$

□

La borne qui résulte de l'inégalité de Cantelli-Chebychev, si on considère toujours que $\mathcal{M}_Q^D \geq 0$, et en remplaçant X par la variable aléatoire $\mathcal{M}_Q(\mathbf{x}, y)$ et a par $\mathcal{M}_Q^D$ dans la nouvelle inégalité, est la suivante :

$$R(H) \ \leq \ \frac{\mathbf{Var}_{(\mathbf{x},y)\sim D}\left(\mathcal{M}_Q(\mathbf{x},y)\right)}{\mathbf{Var}_{(\mathbf{x},y)\sim D}\left(\mathcal{M}_Q(\mathbf{x},y)\right) \ + \ (\mathcal{M}_Q^D)^2} \tag{4.13}$$

Rappelons que :

$$\begin{aligned}
\underset{(\mathbf{x},y)\sim D}{\mathbf{Var}}\left(\mathcal{M}_Q(\mathbf{x},y)\right) &= \underset{(\mathbf{x},y)\sim D}{\mathbf{E}}\left(\mathcal{M}_Q(\mathbf{x},y)\right)^2 \ - \left(\underset{(\mathbf{x},y)\sim D}{\mathbf{E}}\ \mathcal{M}_Q(\mathbf{x},y)\right)^2 \\
&= \mathcal{M}_{Q^2}^D - (\mathcal{M}_Q^D)^2.
\end{aligned} \tag{4.14}$$

En tenant compte de (4.14), l'expression (4.13) devient :

$$R(H) \ \leq \ 1 - \frac{(\mathcal{M}_Q^D)^2}{\mathcal{M}_{Q^2}^D} \tag{4.15}$$

Si on retourne à l'expression (4.12) qui correspond à la borne classique de Breiman, celle-ci suggère que pour minimiser le risque du classificateur des forêts aléatoires, cela revient à minimiser la *corrélation* entre les arbres de la forêt tout en maximisant leur *force*. Ce même constat peut être tiré pour la borne de l'equation (4.15) puisque cela revient à maximiser l'expression $\frac{(\mathcal{M}_Q^D)^2}{\mathcal{M}_{Q^2}^D}$. Cependant maximiser cette expression peut s'avérer difficile lorsqu'on a le numérateur et le dénominateur qui sont près de 0. Une étude détaillée pour ce cas a été proposée dans (20) et nous y reviendrons dans le chapitre suivant.

4.5 Utilisation des données Out-of-bag

Algorithme 6 OOBclassification(S, T)

Entrée : $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, l'ensemble d'entrainement.
Entrée : T, le nombre d'itérations.

$Y_{OOB} \leftarrow \emptyset$
pour $t = 1, \ldots, T$ **faire**
Soit S_t le t-ème échantillon bootstrap et h_t le t-ème arbre obtenu par S_t
fin pour
pour $i = 1, \ldots, m$ **faire**
Déterminer les indices des arbres où l'observation i est out-of-bag :

$$\mathcal{S}_i \leftarrow \{t : (\mathbf{x}_i, y_i) \notin S_t\} \tag{4.16}$$

Classifier l'exemple :

$$H_{OOB}(\mathbf{x}_i) = \underset{c \in Y}{\operatorname{argmax}} \Big(\sum_{t \in \mathcal{S}_i} I(h_t(\mathbf{x}_i) = c) \Big) \tag{4.17}$$

$$Y_{OOB} \leftarrow Y_{OOB} \cup \{H_{OOB}(\mathbf{x}_i)\} \tag{4.18}$$

fin pour
Sortie : Y_{OOB}, le résultat de la classification des exemples de S par H_{OOB}
.

Un autre élément important que Breiman a introduit pour les forêts aléatoires est le concept $out - of - bag$. Le *risque out-of-bag* du classificateur d'une forêt aléatoire sur un ensemble d'entrainement S est déterminé en calculant le risque empirique du classificateur de la forêt, tel que chaque exemple $z_i = (\mathbf{x}_i, y_i)$ appartenant à S est classifié en effectuant un vote de majorité des prédictions des arbres de la forêt dans lesquels

l'exemple z_i n'appartient pas aux échantillons bootstrap utilisés pour développer ces arbres. L'algorithme 6 illustre une classification binaire par le principe des out-of-bag [5]. En se basant sur l'algorithme 6, le risque empirique out-of-bag noté $R_S(H_{OOB})$, est le risque empirique du classificateur out-of-bag et il est défini comme suit :

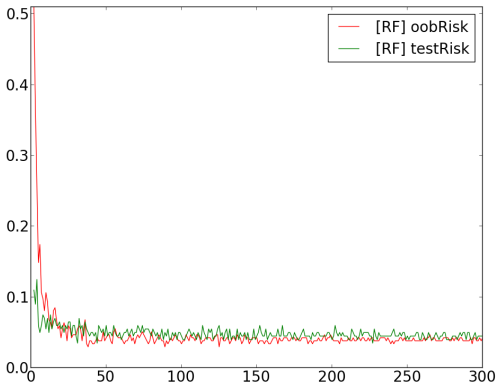
$$R_S(H_{OOB}) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{(\mathbf{x},y)\in S} I\Big(H_{OOB}(\mathbf{x}) \neq y\Big)\,.$$

Comme c'est mentionné dans (11), le risque out-of-bag calculé en utilisant les échantillons out-of-bag permet d'estimer l'erreur de généralisation du classificateur de la forêt aléatoire. La figure 4.2 illustre la comparaison entre le risque out-of-bag basé sur les données d'entraînements et le risque sur l'ensemble test correspondant en utilisant quelques ensembles de données UCI (pour plus de détails voir l'annexe A). Les deux risques (risque out-of-bag et rique sur l'ensemble test) L'étude effectuée dans (9) donne des preuves empiriques que l'estimation du risque par le risque out-of-bag est aussi précis que l'utilisation du risque sur l'ensemble test lorsque la taille de ce dernier est la même que celle de l'ensemble d'entrainement. Cependant, dans quelques rare fois, ça ne fonctionne pas toujours très bien comme le montre nos résultats par la figure 4.2d, où avec 300 arbres, on remarque une petite différence entre les deux risques empiriques. Néanmoins, d'une manière générale, l'utilisation des échantillons out-of-bag apporte donc un avantage qui consiste à nous épargner la nécessité de mettre à côté un ensemble test. Le risque out-of-bag peut aussi être comparé au risque de validation croisée (Section 2.9) sauf que le calcul de ce dernier est plus coûteux. En effet, il est bien connu que l'utilisation de la validation croisée nécessite beaucoup de temps de calcul dû aux découpages qu'on doit effectuer sur l'ensemble d'entrainement. Pour cette raison, au lieu d'effectuer une validation croisée, le risque empirique obtenu grâce aux échantillons out-of-bag offre une meilleure alternative permettant une réduction considérable du temps de calcul. Dans ce cas, le choix des hyperparamètres est effectué en choisissant ceux qui minimisent le risque out-of-bag.

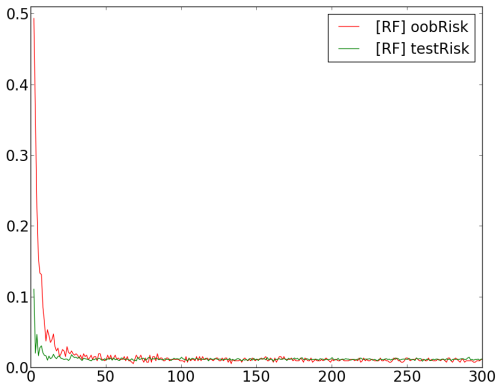
4.6 Mesure de l'importance des variables

Dans cette section, nous utiliserons le terme de variable pour désigner un attribut. Breiman a détérminé une méthode pour les forêts aléatoires qui permet de déterminer la pertinence de chaque variable de l'ensemble des données. Lors de l'induction des

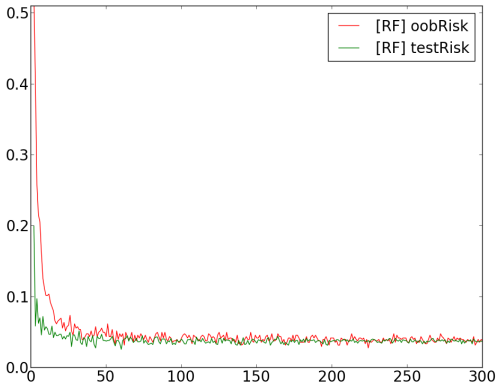
5. la définition du classificateur H_{OOB} est donné à l'intérieur du pseudo-code définissant l'algorithme 6



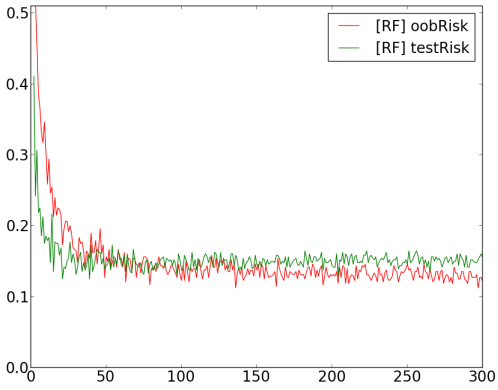
(a) USvotes



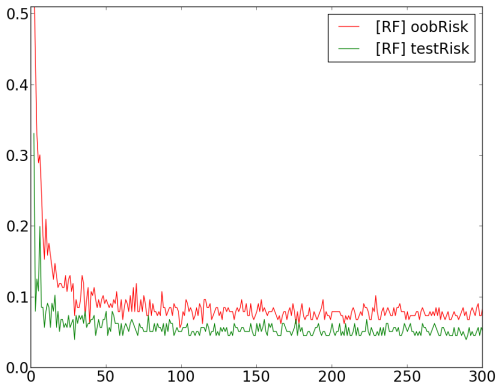
(b) Mnist_17



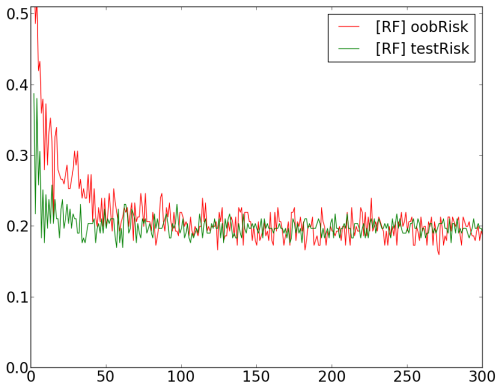
(c) Letter_OQ



(d) Tic-Tac-Toe



(e) Ionosphere



(f) Heart

FIGURE 4.2 – Risque out-of-bag (oobRisk) comparé avec le risque sur l'ensemble test (testRisk) calculés sur six ensembles de données

arbres de la forêt, pour chaque arbre dévéloppé à partir d'un bootstrap S_t, on effectue une permutation aléatoire des valeurs de la i-ème variable dans les exemples appartenant à l'ensemble S_t. A la fin, nous calculons le risque out-of-bag de la forêt avec les arbres dont les valeurs de la i-ème variable ont été permuté aléatoirement dans les bootstraps. L'importance de la i-ème variable est calculée en comparant les risques d'une forêt dont un ensemble de données normal a été utilisé et d'une forêt dont les valeurs de la i-ème variable ont été permuté aléatoirement dans les bootstraps. Pour un ensemble d'entrainement S, soit H_{OOB} le classificateur out-of-bag des forêts aléatoires et H^i_{OOB} le classificateur des forêts aléatoires dont les valeurs de la i-ème variable ont été permutés aléatoirement dans les bootstraps. L'importance de la variable i, nommée M_i est calculée comme suit :

$$M_i = \max\left(0, R_S(H^i_{OOB}) - R_S(H_{OOB})\right) \tag{4.19}$$

Chapitre 5

Les bornes dérivées de la théorie Pac-Bayes

5.1 Introduction

Dans le domaine de la classification, l'objectif d'un algorithme d'apprentissage est de construire un bon classificateur. Par bon classificateur, il faut surtout comprendre un classificateur de faible risque. L'approche classique qu'on rencontre souvent, consiste à développer des algorithmes d'apprentissage en se basant souvent sur des heuristiques. Une démonstration des propriétés de l'algorithme s'en suit par après, en déterminant par exemple des bornes théoriques sur le risque des classificateurs générés. Pour le cas des forêts aléatoires vus dans le chapitre précédent(voir Chapitre 4) par exemple, Breiman a proposé l'algorithme des forêts aléatoires et par la suite il a dérivé une borne (Borne 4.12) sur le risque du classificateur des forêts aléatoires qui dépend de la distribution D qui génère les exemples. On retrouve une borne semblable (voir les similarités et différences dans la section 4.4) dans la littérature sous l'appelation de la borne C_Q (23) où elle est généralisée pour l'utiliser avec un vote de majorité pondéré qui n'est pas nécessairement uniforme. Dans ce chapitre, on va revenir brièvement sur cette borne.

On va également parler de la théorie PAC-Bayes qui est largement utilisée pour développer des bornes. À partir de ces bornes, on y dérive des algorithmes d'apprentissage dont l'objectif consiste à minimiser ces bornes. L'approche PAC-Bayes a été introduite par David McAllester dans (28). L'objectif de cette approche est de fournir des garanties PAC (Probablement Approximativement Corrects) en proposant des bornes supérieures sur le risque d'un ensemble de classificateurs. Les bornes de type PAC-Bayes permettent

de borner à l'aide d'un ensemble d'entrainement, le risque de tout classificateur issu d'un vote de majorité de classificateurs d'un ensemble $\mathcal{H}$. Dans ce chapitre on va revenir sur quelques bornes dérivées de la théorie PAC-Bayes.

Pour conclure ce chapitre, on va présenter un algorithme de l'état de l'art appelé MinCq (20) qui est le résultat de la combinaison de la borne C_Q (23) et des connaissances apportées par la théorie PAC-Bayes. En effet, cet algorithme minimise la borne C_Q et trouve sa justification entre autre en utilisant une approche PAC-Bayes.

5.2 Classificateur de Bayes et de Gibbs

Cette section est inspirée par (27), un chapitre du cours d'Apprentissage automatique. Un *classificateur de Bayes* est un *classificateur* obtenu par un *vote de majorité* de plusieurs classificateurs distincts regroupés dans un ensemble $\mathcal{H}$. Chaque classificateur $h \in \mathcal{H}$ est un *classificateur de base* du vote de majorité. Le nombre de classificateurs de base de l'ensemble $\mathcal{H}$ peut être fini ou infini. Dans le vote de majorité effectué pour obtenir un classificateur de Bayes, chaque classificateur de base h est pondéré par un poids $Q(h)$ détérminé par une distribution Q sur l'ensemble $\mathcal{H}$. Le poids $Q(h)$ associé au classificateur h détermine le pouvoir de décision de h dans le vote de majorité. Pour classer un exemple $\mathbf{x}$, le classificateur de Bayes B_Q considère le vote de chacun des classificateurs de base et retient l'opinion majoritaire suivant la pondération Q. Ainsi, la prédiction $B_Q(\mathbf{x})$ est donnée par :

$$B_Q(\mathbf{x}) \stackrel{\text{def}}{=} \underset{y \in Y}{\operatorname{argmax}} \Big(\underset{h \sim Q}{\mathbf{E}} I(h(\mathbf{x}) = y) \Big).$$

Dans le cas où l'ensemble $\mathcal{H}$ des classificateurs de base est dénombrable, cette définition revient à :

$$B_Q(\mathbf{x}) \stackrel{\text{def}}{=} \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \Big(\sum_{h \in \mathcal{H}} Q(h) I(h(\mathbf{x}) = y) \Big).$$

Le *risque de Bayes* d'une distribution Q sur un ensemble de classificateur $\mathcal{H}$ est le risque du classificateur de Bayes associé à Q. Le vrai risque de Bayes $R(B_Q)$ et le risque empirique de Bayes $R_S(B_Q)$ sont donnés par :

$$R(B_Q) \stackrel{\text{def}}{=} \underset{(\mathbf{x},y) \sim D}{\mathbf{E}} I\Big(B_Q(\mathbf{x}) \neq y \Big),$$

$$R_S(B_Q) \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^{m} I\Big(B_Q(\mathbf{x}_i) \neq y_i \Big).$$

Le *classificateur de Gibbs* à son tour est un classificateur obtenu en pigeant aléatoirement un classificateur h selon la distribution Q sur l'ensemble $\mathcal{H}$ des classificateurs de base. Comparativement au classificateur de Bayes qui est déterministe, le classificateur de Gibbs est stochastique. Pour classifier un exemple $\mathbf{x}$, il pige aléatoirement un classificateur h selon Q et retourne $h(\mathbf{x})$.

Le vrai risque de Gibbs et le risque empirique de Gibbs sont donnés par :

$$R(G_Q) \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{h\sim Q} R(h) = \mathop{\mathbf{E}}_{h\sim Q} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} I(h(\mathbf{x}) \neq y)\,, \tag{5.1}$$

$$R_S(G_Q) \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{h\sim Q} R_S(h) = \frac{1}{m}\sum_{i=1}^{m} \mathop{\mathbf{E}}_{h\sim Q} I(h(\mathbf{x}_i) \neq y_i)\,. \tag{5.2}$$

Dans le cas où l'ensemble $\mathcal{H}$ des classificateurs de base est dénombrable, ces définitions reviennent à :

$$R(G_Q) \stackrel{\text{def}}{=} \sum_{h\in\mathcal{H}} Q(h)R(h) = \sum_{h\in\mathcal{H}} \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} Q(h)I(h(\mathbf{x}) \neq y)\,,$$

$$R_S(G_Q) \stackrel{\text{def}}{=} \sum_{h\in\mathcal{H}} Q(h)R_S(h) = \frac{1}{m}\sum_{h\in\mathcal{H}}\sum_{i=1}^{m} Q(h)I(h(\mathbf{x}_i) \neq y_i)\,.$$

La relation entre le risque du classificateur par vote de majorité (classificateur de Bayes) et le risque du classificateur de Gibbs est donnée par le lemme suivant :

Lemme 4. *(27) Pour toute distribution Q sur un espace de classificateurs $\mathcal{H}$ caractérisant un vote de majorité, on a :*

$$R(B_Q) \leq 2R(G_Q)\,.$$

Démonstration. Supposons d'abord que le classificateur de Bayes B_Q classifie incorrectement l'exemple $(\mathbf{x}, y)$. Donc, au moins la moitié du poids du vote de majorité est assignée à des classificateurs de base qui classifient incorrectement $(\mathbf{x}, y)$.

$$\begin{aligned} B_Q(\mathbf{x}) \neq y &\Rightarrow \Pr_{h\sim Q}(h(\mathbf{x}) \neq y) \geq 1/2 \\ &\Rightarrow \mathop{\mathbf{E}}_{h\sim Q} I\left(h(\mathbf{x}) \neq y\right) \geq 1/2 \end{aligned}$$

Soit $R_{(\mathbf{x},y)}(B_Q)$ et $R_{(\mathbf{x},y)}(G_Q)$ le risque de B_Q et de G_Q sur l'exemple $(\mathbf{x}, y)$. Le risque de Gibbs pour $(\mathbf{x}, y)$ est d'au moins 50% :

$$R_{(\mathbf{x},y)}(B_Q) = 1 \;\Rightarrow\; R_{(\mathbf{x},y)}(G_Q) \geq 1/2\ .$$

Donc, peu importe si B_Q classe l'exemple $(\mathbf{x}, y)$ correctement (auquel cas $R_{(\mathbf{x},y)}(B_Q) = 0$) ou s'il classe l'exemple $(\mathbf{x}, y)$ incorrectement (auquel cas $R_{(\mathbf{x},y)}(B_Q) = 1$), on remarque que :

$$R_{(\mathbf{x},y)}(B_Q) \;\leq\; 2R_{(\mathbf{x},y)}(G_Q)\,,$$

ce qui implique :

$$\mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} R_{(\mathbf{x},y)}(B_Q) \;\leq\; 2 \mathop{\mathbf{E}}_{(\mathbf{x},y)\sim D} R_{(\mathbf{x},y)}(G_Q)\,,$$

et conséquemment :

$$R(B_Q) \;\leq\; 2R(G_Q)\,.$$

□

En se basant sur l'expression du lemme 4, on remarque que le risque du classificateur de Bayes est borné supérieurement par le double du risque de Gibbs. Cependant, ceci ne signifie pas que le classificateur de Gibbs est préférable au classificateur de Bayes. Bien qu'il existe des exemples où la borne est atteinte (c'est-à-dire que $R(B_Q) \approx 2R(G_Q)$), il est fréquent que le risque de Gibbs d'un vote de majorité soit beaucoup supérieur au risque de Bayes.

5.3 Bornes PAC-Bayes

Dans le chapitre 4, nous avons introduit l'algorithme des forêts aléatoires qui construisent un classificateur par vote de majorité uniforme qu'on a noté H. Dans la section 5.2, on a introduit le classificateur de Bayes. Il s'agit d'un classificateur déterminé par un vote de majorité pondéré par une distribution Q sur les classificateurs de base (la distribution Q n'est pas nécessairement uniforme). Pour standardiser notre notation, on va dorénavant prendre dans le reste de ce mémoire le classificateur des forêts aléatoires comme étant un classificateur de Bayes dont la distribution Q sur les arbres de décision (classificateurs de base) est uniforme. Autrement dit, pour les forêts aléatoires on aura un classificateur de Bayes dont le poids associé à chaque arbre de décision dans la forêt aléatoire qui est le même ($Q(h) = \frac{1}{n}$ pour le cas d'un ensemble $\mathcal{H}$ fini de n classificateurs de base). Dans la section 4.4, nous avons présenté la borne que Breiman a développée pour le risque des forêts aléatoires. On retrouve une borne semblable dans (20) sous l'appelation de borne $C_Q(23)$. Breiman utilise, pour développer sa borne, l'inégalité

de Tchebychev tandis que dans (20) l'inégalité de Cantelli-Tchebychev est utilisée. La borne de Breiman pour les forêts aléatoires n'est cependant pas une borne PAC (Probablement approximativement Correct) car elle dépend de la distribution D qui génère les exemples. Une borne PAC doit montrer ce qu'il y a à optimiser sur les données. Elle dépend donc d'un ensemble d'entrainement. Dans cette section, nous revenons sur l'approche PAC-Bayes pour borner le risque d'un classificateur. Celle-ci a été introduite par David McAllester dans (28). Cette approche ne permet pas de borner directement le risque du classificateur de Bayes mais plutôt le classificateur de Gibbs. Cependant comme indiqué dans la section 5.2 par le lemme 4, il existe une relation entre le classificateur de Bayes et le classificateur de Gibbs. En effet, le risque du classificateur d'un vote de majorité (classificateur de Bayes) est borné supérieurement par le double du risque de Gibbs associé au même vote de majorité. A partir de cette relation, on peut donc utiliser l'approche PAC-Bayes pour borner d'une manière indirecte le risque du classificateur de Bayes.

Soit un classificateur de Gibbs G_Q caractérisé par une distribution Q sur un espace $\mathcal{H}$ de classificateurs de base. Pour introduire le théorème général sur lequel est basée la théorie PAC-Bayes, considérons :

- Le risque empirique du classificateur de Gibbs $R_S(G_Q)$, mesuré sur l'ensemble d'entraînement S contenant m exemples ;
- La distribution P sur l'espace $\mathcal{H}$, qui représente les connaissances qu'on a du problème *a priori*. Le choix de la distribution P ne doit pas être influencé par les exemples de l'ensemble d'entraînement S.
- La distribution Q sur l'espace $\mathcal{H}$, représentant le classificateur obtenu par l'apprentissage. Celui-ci correspond aux connaissances qu'on a du problème *a posteriori*. Pour déterminer la distribution Q, l'algorithme d'apprentissage prend en considération cette fois-ci les exemples d'entrainement de l'ensemble S.
- Un paramètre de confiance δ, qui détermine avec quelle probabilité la borne calculée sera invalide.
- la *divergence de Kullback-Leibler* entre les distributions P et Q :

$$\mathrm{KL}(Q\|P) \stackrel{\text{def}}{=} \mathop{\mathbf{E}}_{h\sim Q} \ln \frac{Q(h)}{P(h)} \,. \tag{5.3}$$

  La divergence de Kullback-Leibler est une mesure de similarité entre les connaissances qu'on a d'un problème d'apprentissage *a priori* et *a posteriori* (intuitivement considérée comme mesure de distance entre les distributions P et Q). Elle est nulle lorsque $Q \equiv P$ et sa valeur croît lorsque la distribution Q s'éloigne de la distribution P.

5.3.1 Théorème PAC-Bayes général

Le théorème PAC-Bayes général suivant est tiré de (17) et s'énonce comme suit :

Théorème 5. *(17) Pour toute distribution D, pour tout ensemble $\mathcal{H}$ de classificateurs, pour toute distribution P sur $\mathcal{H}$, pour tout $\delta \in]0,1]$ et pour toute fonction convexe $\mathcal{D} : [0,1] \times [0,1] \to \mathbb{R}$, on a :*

$$\Pr_{S\sim D^m}\left(\begin{array}{l}\forall\, Q \text{ sur } \mathcal{H}: \\ \mathcal{D}(R_S(G_Q), R(G_Q)) \leq \frac{1}{m}\left[\mathrm{KL}(Q\|P) + \ln\left(\frac{1}{\delta}\mathop{\mathbf{E}}_{S\sim D^m}\mathop{\mathbf{E}}_{h\sim P} e^{m\mathcal{D}(R_S(h),R(h))}\right)\right]\end{array}\right) \geq 1-\delta\,,$$

où $\mathrm{KL}(Q\|P)$ est la divergence de Kullback-Leibler donnée par l'équation ((5.3)).

Le lecteur interessé pourra retrouver la démonstration du théorème précédent dans (17).

5.3.2 Quelques bornes PAC-Bayes

A partir du théorème 5 présenté dans la section précédente, il est possible de dériver plusieurs bornes PAC-Bayes. En effet, comme signalé dans (20) et (24), il suffit d'insérer dans l'expression du théorème PAC-Bayes général une fonction convexe $\mathcal{D} : [0,1] \times [0,1] \to \mathbb{R}$ adéquate. Les bornes qu'on va donc présenter dans cette section proviennent des travaux effectués antérieurement. C'est ainsi qu'on va nommer chaque borne en la désignant selon son auteur. Alexandre Laccasse dans sa thèse (24) ainsi que Germain et al. dans (17) y sont revenus en montrant comment on peut obtenir ces bornes en partant de la borne présentée par le théorème 5. C'est ainsi que nous nous contenterons d'énoncer ces bornes, l'approche utilisée pour les obtenir ainsi que leurs démonstrations peuvent être consultées dans (17) et (24).

Corollaire 6 (Borne PAC-Bayes version Langford-Seeger)**.** *Pour toute distribution D, pour tout ensemble $\mathcal{H}$ de classificateurs, pour toute distribution P sur $\mathcal{H}$ et pour tout $\delta \in]0,1]$, on a :*

$$\Pr_{S\sim D^m}\left(\forall Q \text{ sur } \mathcal{H}: \mathrm{kl}\Big(R_S(G_Q)\Big\|R(G_Q)\Big) \leq \frac{\mathrm{KL}(Q\|P) + \ln\frac{\xi(m)}{\delta}}{m}\right) \geq 1-\delta\,,$$

où $\mathrm{kl}(q\|p)$ est la divergence de Kullback-Leibler entre deux distributions de Bernoulli de probabilité de succès p et q définie comme suit :

$$\mathrm{kl}(q\|p) \stackrel{\text{def}}{=} q\ln\frac{q}{p} + (1-q)\ln\frac{1-q}{1-p}\,. \tag{5.4}$$

et $\mathrm{KL}(Q\|P)$ la divergence de Kullback-Leibler donnée par l'équation ((5.3)) et où :

$$\xi(m) \stackrel{\text{def}}{=} \sum_{k=0}^{m}\binom{m}{k}\left(\frac{k}{m}\right)^k\left(1-\frac{k}{m}\right)^{m-k}\,. \tag{5.5}$$

A partir du théorème PAC-Bayes général, en optant pour la fonction de pseudo-distance donnée par $\mathcal{D}(q,p) = 2(q-p)^2$, nous obtenons une autre version du théorème PAC-Bayes général exprimée par le corollaire suivant.

Corollaire 7 (Borne PAC-Bayes version McAllester)**.** *Soit D une distribution, $\mathcal{H}$ un ensemble de classificateurs et P une distribution à priori sur $\mathcal{H}$ et soit $\delta \in (0,1]$. Alors*

$$\Pr_{S\sim D^m}\left(\forall Q \text{ sur} \mathcal{H}\colon \ R(G_Q) \leq R_S(G_Q) + \sqrt{\frac{1}{2m}\left[\mathrm{KL}(Q\|P) + \ln\frac{\xi(m)}{\delta}\right]}\right) \geq 1-\delta\,.$$

Pour obtenir la prochaine version de la borne, on applique le théorème 5 avec une fonction $\mathcal{D}(q,p)$ qui est linéaire en q (le risque empirique). On cherche alors une fonction de la forme $D(q\|p,C) = \mathcal{F}(p) - C\cdot q$, où l'influence de q est déterminée par un hyper-paramètre $C \in \mathbb{R}^+$. On souhaite que la fonction $\mathcal{F}$ soit convexe et indépendante de la variable q.

Corollaire 8 (Borne PAC-Bayes version Catoni)**.** *Pour toute distribution D, pour toute classe $\mathcal{H}$ de classificateurs, pour toute distribution P sur $\mathcal{H}$, pour toute valeur réelle positive C et pour tout $\delta \in]0,1]$, on a :*

$$\Pr_{S\sim D^m}\left(\begin{array}{l}\forall Q \text{ sur } \mathcal{H}\colon \\ R(G_Q) \leq \frac{1}{1-e^{-C}}\left\{1-\exp\left[-\left(C\cdot R_S(G_Q) + \frac{1}{m}\left[\mathrm{KL}(Q\|P) + \ln\frac{1}{\delta}\right]\right)\right]\right\}\end{array}\right) \geq 1-\delta\,.$$

où $\mathrm{KL}(Q\|P)$ est la divergence de Kullback-Leibler donnée par l'équation ((5.3)).

Comme dernier exemple de théorème, nous présentons la borne PAC-Bayes version d'Audibert dont la preuve se trouve dans (1).

Corollaire 9 (Borne PAC-Bayes version Audibert)**.** *Soit D une distribution, $\mathcal{H}$ un ensemble de classificateurs et P une distribution à priori sur $\mathcal{H}$ et soit $\delta \in (0,1]$.*
On a :

$$\Pr_{S\sim D^m}\left(\begin{array}{l}\forall Q \text{ sur } \mathcal{H}\colon \\ |R(G_Q) - R_S(G_Q)| \leq \sqrt{\frac{2R_S(G_Q)\left[1-R_S(G_Q)\right]\left[\mathrm{KL}(Q\|P)+\ln\frac{2\sqrt{m}}{\delta}\right]}{m}} \\ +\frac{4}{3m}\left[\mathrm{KL}(Q\|P) + \ln\frac{2\sqrt{m}}{\delta}\right]\end{array}\right) \geq 1-\delta\,.$$

5.4 MinCq : un algorithme minimisant la borne C_Q

Dans la section 4.4 du chapitre 4, nous avons présenté la borne C_Q (4.15). Nous avons également introduit, dans les sections précédentes de ce chapitre, la théorie PAC-Bayes

ainsi que quelques bornes dérivées de cette théorie. Dans cette section, on présente un algorithme de l'état de l'art appelé MinCq. Cet algorithme a été proposé dans (20) et nous nous contentons donc de le décrire d'une manière brève. Le lecteur interessé pourra retrouver toute la documentation nécessaire autour de MinCq dans le papier présenté par les auteurs de cet algorithme dans (20).
L'algorithme MinCq minimise la borne C_Q, qui est une borne sur le risque des votes de majorité. Il a été conçu pour deux cadres : le cadre inductif supervisé et le cadre transductif (ou semi-supervisé) (pour plus de détails sur l'induction et la transduction voir (20)). Les deux versions peuvent être exprimées comme des programmes quadratiques sur des matrices positives semi-définies. MinCq retourne donc un vote de majorité Q-pondéré sur des fonctions (appelées votants), qui donnent une mauvaise performance individuelle. Ces votants sont souvent appelés des "weak learners". La décision de chaque vote est donc basée sur une petite majorité. Cependant minimiser la valeur empirique de la borne C_Q a tendance à surapprendre les données. Pour surmonter ce problème, les auteurs de MinCq utilisent une distribution Q de votants restreint à être *quasi-uniforme*. En effet, pour l'algorithme MinCq, un espace de votants $\mathcal{H}$ qui est *auto-complémenté* est utilisé, ce qui veut dire que

$$h_{i+n}(\mathbf{x}) \;=\; -h_i(\mathbf{x}) \text{ pour tout } \mathbf{x} \in \mathcal{X} \text{ et tout } i \in \{1, \ldots, n\}\,.$$

et pour tout $\mathcal{H}$ auto-complémenté, il considère juste les distributions *quasi-uniformes*, c'est à dire les distributions Q telles que

$$Q(h_i) + Q(h_{i+n}) \;=\; 1/n \qquad \text{pour tout } i \in \{1, \ldots, n\}\,.$$

Pour cette distribution Q de votants quasi-uniforme, MinCq se restreint également à une marge de la combinaison Q-convexe sur l'ensemble d'entraînement qui est fixée à une valeur précise $\mu > 0$. Cette stratégie d'apprentissage a été justifiée dans (20) par une borne PAC-Bayes dédiée aux distributions a posteriori quasi-uniformes, qui ne contient pas de KL-divergence entre la distribution a priori uniforme et la distribution a posteriori quasi-uniforme. Cette borne PAC-Bayes est exprimée par le théorème suivant :

Théorème 10. *Pour toute distribution D, pour tout $m \geq 8$, pour toute famille $\mathcal{H}$ auto-complémentée de fonctions à valeur réelle B-bornée, et pour tout $\delta \in (0, 1]$, on a*

$$\Pr_{S\sim D^m}\left(\begin{array}{l} \textit{Pour toute distribution quasi-uniforme } Q \textit{ sur } \mathcal{H}, \textit{ on a :} \\ \mathcal{M}_Q^S - \frac{2B\sqrt{\ln\frac{2\sqrt{m}}{\delta}}}{\sqrt{2m}} \leq \mathcal{M}_Q^D \leq \mathcal{M}_Q^S + \frac{2B\sqrt{\ln\frac{2\sqrt{m}}{\delta}}}{\sqrt{2m}} \\ \quad \textit{et} \\ \mathcal{M}_{Q^2}^S - \frac{2B^2\sqrt{\ln\frac{2\sqrt{m}}{\delta}}}{\sqrt{2m}} \leq \mathcal{M}_{Q^2}^D \leq \mathcal{M}_{Q^2}^S + \frac{2B^2\sqrt{\ln\frac{2\sqrt{m}}{\delta}}}{\sqrt{2m}} \end{array}\right) \geq 1 - \delta\,.$$

Le Théorème 10 est limité aux votants B-bornés (c'est à dire, les votants h tels que $-B \leq h(\mathbf{x}) \leq +B$ pour tout $\mathbf{x} \in \mathcal{X}$). La preuve de ce théorème se retrouve également dans (20).

Pour conclure avec l'algorithme MinCq, considérons aussi la définition suivante. On dit qu'une valeur μ est *D-réalisable* s'il existe une distribution quasi-uniforme Q telle que $\mathcal{M}_Q^D = \mu$.

L'algorithme MinCq, est finalement défini dans (20) comme suit :

– **l'algorithme MinCq.** Soit un ensemble $\mathcal{H}$ de votants, un ensemble d'entraînement S et un $\mu > 0$ S-réalisable (s'il existe une distribution quasi-uniforme Q telle que $\mathcal{M}_Q^S = \mu$). Parmi toutes les distributions quasi-uniformes Q de marge empirique $\mathcal{M}_Q^S$ exactement égale à μ, l'algorithme MinCq consiste à trouver celle qui minimise $\mathcal{M}_{Q^2}^S$.

Chapitre 6

SORF : Une forêt aléatoire inspirée de l'approche PAC-Bayesienne

6.1 Introduction

Dans le chapitre 4, on a présenté l'algorithme des forêts aléatoires, algorithme qui a été introduit par Leo Breiman dans (11). L'algorithme des forêts aléatoires utilise une distribution uniforme sur les arbres de la forêt dans le vote de majorité. La question à laquelle on peut se demander est de savoir s'il n y aurait pas une distribution optimale qui permettrait d'avoir une forêt plus performante en terme de minimisation du risque. Ce chapitre essaie donc de répondre en quelques sortes à cette question en proposant un algorithme qui essaie de chercher la distribution optimale sur les arbres de la forêt. Dans le chapitre précédent, on a introduit la théorie PAC-Bayes et quelques bornes qui sont dérivées de cette approche. Notre objectif étant de chercher une distribution à postériori Q optimale sur l'ensemble des classificateurs de base, nous devons chercher une stratégie nous permettant d'obtenir cette distribution Q sans pour autant surapprendre. Référons-nous donc à la section 5.3 sur la théorie PAC-Bayes. Dans cette section, les bornes PAC-Bayes suggérent généralement de minimiser $R_S(G_Q)$ et $\mathrm{KL}(Q\|P)$ parmi les variables qui dépendent de la distribution à postériori Q. Rappelons que le $\mathrm{KL}(Q\|P)$ intervient comme régularisateur pour qu'il n'y ait pas de surapprentissage. Rappelons aussi que pour un ensemble de n classificateurs, $\mathrm{KL}(Q\|P)$ est au plus $\ln(n)$. Nous pouvons donc ici borner $\mathrm{KL}(Q\|P)$ par une constante et dans ce cas, minimiser la borne PAC-Bayes revient à minimiser seulement $R_S(G_Q)$. Dans ce chapitre, nous adoptons donc une première approche permettant de trouver une distribution à postériori Q par la minimisation de $R_S(G_Q)$. Ceci aboutit à un algorithme d'apprentissage automatique

que nous avons nommé SORF pour *"Structured Output Random Forest"*. Ce chapitre consiste donc à introduire l'algorithme SORF . Pour cela, on va d'abord commencer par introduire ce que c'est le *"structured output prediction"* qu'on pourrait traduire par "prédiction dont la sortie est une structure"(donc $\mathcal{Y}$ est l'ensemble des structures possibles). Après, on va revenir principalement sur l'algorithme SORF. Nous verrons que l'algorithme SORF se résume à être un simple programme d'optimisation quadratique avec des contraintes linéaires qui s'expriment sous forme matricielle.

6.2 Introduction au Structured Output Prediction

En apprentissage supervisé, l'algorithme d'apprentissage reçoit en paramètre un ensemble d'entrainement $S \stackrel{\text{def}}{=} \{(x_1, y_1), \ldots, (x_m, y_m)\}$ contenant m exemples où chaque exemple est constitué d'une paire description-étiquette $(x, y) \in \mathcal{X} \times \mathcal{Y}$.
L'espace des entrées $\mathcal{X}$ et l'espace des sorties $\mathcal{Y}$ sont arbitraires mais nous supposons l'existence d'une fonction de l'espace des entrées $\mathcal{X}$ vers un espace $\mathcal{H}_\mathcal{X}$ des caractéristiques de $\mathcal{X}$, $X : \mathcal{X} \to \mathcal{H}_\mathcal{X}$ et une fonction de l'espace des sorties $\mathcal{Y}$ vers un espace $\mathcal{H}_\mathcal{Y}$ des caractéristiques de $\mathcal{Y}$, $Y : \mathcal{Y} \to \mathcal{H}_\mathcal{Y}$, où $\mathcal{H}_\mathcal{X}$ et $\mathcal{H}_\mathcal{Y}$ sont des espaces vectoriels de grande dimensionalité.

L'objectif de l'algorithme d'apprentissage est de construire un prédicteur (fonction de prédiction) h tel que la sortie pour un nouvel instance x est une étiquette y avec $y = h(x)$.

Dans le cas du Structured Output Prediction, l'algorithme d'apprentissage reçoit en entrée l'ensemble des données d'apprentissage S, et il a comme objectif de construire un prédicteur dont la sortie est une structure. Cette structure peut être complexe et elle est différente de l'étiquette de classe standard dont on a par exemple l'habitude de rencontrer en classification. Cette structure fournie comme sortie par notre prédicteur variera donc en fonction du domaine de sortie $\mathcal{Y}$. Par exemple, en traitement du langage naturel, on pourrait s'attendre à avoir comme sortie "une phrase" ou par exemple "un arbre d'analyse syntaxique". En robotique, on pourrait avoir comme sortie une "séquence d'actions", etc.

Pour représenter ce prédicteur, nous adoptons la notation introduite dans (19) et nous utilisons un opérateur linéaire $\mathbf{W}$ qui transforme les vecteurs de l'espace $\mathcal{H}_\mathcal{X}$ en vecteurs de l'espace $\mathcal{H}_\mathcal{Y}$. Pour chaque entrée $x \in \mathcal{X}$ et un prédicteur $\mathbf{W}$, la sortie $y_\mathbf{w}(x)$ prédite

par $\mathbf{W}$ est donnée par :

$$y_{\mathbf{w}}(x) \stackrel{\text{def}}{=} \underset{y \in \mathcal{Y}}{\operatorname{argmin}} \ \|Y(y) - \mathbf{W}X(x)\| \ , \tag{6.1}$$

où $\|\cdot\|$ dénote la norme euclidienne dans l'espace $\mathcal{H}_{\mathcal{Y}}$.

Si l'on note le produit scalaire dans $\mathcal{H}_{\mathcal{Y}}$ par $\langle \cdot, \cdot \rangle$, on a que $\|Y(y) - \mathbf{W}X(x)\|^2 = \|Y(y)\|^2 + \|\mathbf{W}X(x)\|^2 - 2\langle Y(y), \mathbf{W}X(x)\rangle$. Ainsi, si $\|Y(y)\|$ est le même pour tout $y \in \mathcal{Y}$—ce que nous considérons être le cas, on a $y_{\mathbf{w}}(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \ \langle Y(y), \mathbf{W}X(x)\rangle$.

6.3 Notation et représentation mathématique

Considérons un ensemble d'entrainement S contenant m exemples tel que $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \subset \mathcal{X} \times \mathcal{Y}$. Nous supposons aussi que les exemples $(\mathbf{x}, y)$ sont indépendants et identiquement distribués suivant une distribution D inconnue. On a donc $S \sim D^m$.

On a aussi :
- $\mathcal{Y} = \{-1, 1\}$ dans le cas de la **classification binaire** ;
- $\mathcal{Y} = \{1, 2, \ldots, l\}$ dans le cas de la **classification multiclasse**.
- $\mathcal{Y}$ est un ensemble de structure dans le cas de la prédiction de structures
- $\mathcal{H} = \{h_1, \ldots, h_n\}$, un ensemble de n arbres de décisions construits à partir des données S.
- $Q = \{Q_1, \ldots, Q_n\}$, une distribution sur $\mathcal{H}$.

Dans nos travaux, nous abordons d'abord une première approche pour le cas binaire et pour le cas multiclasse. A la lumière des résultats qu'on aura, on pourra envisager une géneralisation de l'algorithme pour d'autres structures autre que binaire et multiclasse.

En tenant compte des notions introduites dans la section 6.2 sur la prédiction de structures, nous représentons chaque classe $y \in \mathcal{Y}$ par $\boldsymbol{\phi}(y)$, un vecteur unitaire de dimension $l = |\mathcal{Y}|$, tel que :
- dans le cas **binaire** : $\boldsymbol{\phi}(-1) = (1, 0)$ et $\boldsymbol{\phi}(+1) = (0, 1)$;
- dans le cas **multiclasse** : $\boldsymbol{\phi}(y) = (i_1, \ldots, i_l)$ $\langle$avec $i_p = I(p = y) \quad \forall p \in \{1, \ldots, l\}\rangle$ Ainsi par exemple, pour l=3, $\mathcal{Y} = \{1, 2, 3\}$ et y=2, on a $\boldsymbol{\phi}(y) = (0, 1, 0)$.

Le classificateur de Bayes B_Q obtenu par un vote de majorité Q-pondéré des classificateurs de l'ensemble $\mathcal{H}$ est donné par :

$$B_Q \;=\; \underset{y\in Y}{\text{argmax}} \Big(\sum_{i=1}^{n} Q_i I(h_i(\mathbf{x}) = y) \Big)$$

Notons l'expression $\Big(\sum_{i=1}^{n} Q_i I(h_i(\mathbf{x}) = y) \Big)$ par $Q(\mathbf{x}, y)$. Dans ce cas, on a :

$$B_Q \;\stackrel{\text{def}}{=}\; \underset{y\in Y}{\text{argmax}}\; Q(\mathbf{x}, y)$$

La sortie prédite $y_Q(\mathbf{x})$ sur $\mathbf{x}$ par Q est définie par :

$$y_Q(\mathbf{x}) \stackrel{\text{def}}{=} \underset{y\in Y}{\text{argmin}} \;||\; \boldsymbol{\phi}(y) - \sum_{i=1}^{n} Q_i \boldsymbol{\phi}(h_i(\mathbf{x})) \;||^2 \tag{6.2}$$

Notons l'expression $\sum_{i=1}^{n} Q_i \boldsymbol{\phi}(h_i(\mathbf{x}))$ par $\boldsymbol{\phi}_Q(\mathbf{x})$ [1], on a alors :

$$y_Q(\mathbf{x}) \stackrel{\text{def}}{=} \underset{y\in Y}{\text{argmin}} \;||\; \boldsymbol{\phi}(y) - \boldsymbol{\phi}_Q(\mathbf{x}) \;||^2 \tag{6.3}$$

Le théorème suivant démontre que pour tout $\mathbf{x}$, la sortie prédite $y_Q(\mathbf{x})$ sur $\mathbf{x}$ par Q est la même que celle du classificateur de Bayes B_Q donnée par $B_Q(\mathbf{x})$. En effet, cette égalité est prouvée par la démonstration du théorème suivant :

Théorème 11. *$\forall \mathbf{x} \in X$, on a :*

$$y_Q(\mathbf{x}) = B_Q(\mathbf{x})$$

Démonstration.

$$\begin{aligned} y_Q(\mathbf{x}) &= \underset{y\in Y}{\text{argmin}} \;||\; \boldsymbol{\phi}(y) - \sum_{i=1}^{n} Q_i \boldsymbol{\phi}(h_i(\mathbf{x})) \;||^2 \\ &= \underset{y\in Y}{\text{argmin}} \;||\; \boldsymbol{\phi}(y) - \sum_{y'} Q(\mathbf{x}, y') \boldsymbol{\phi}(y') \;||^2 \\ &= \underset{y\in Y}{\text{argmin}} \;||\; \Big(1 - Q(\mathbf{x}, y)\Big) \boldsymbol{\phi}(y) - \sum_{y'\neq y} Q(\mathbf{x}, y') \boldsymbol{\phi}(y') \;||^2 \end{aligned}$$

Sachant que dans un plan donné, la norme d'un vecteur $\vec{u}$ de cordonnées (x, y) est donnée par $||\; \vec{u} \;|| = \sqrt{x^2 + y^2}$. Par conséquent, $||\; \vec{u} \;||^2 = x^2 + y^2$. Si on considère le plan

1. Remarquez que $\boldsymbol{\phi}_Q(\mathbf{x}) \neq \boldsymbol{\phi}\Big(y_Q(\mathbf{x})\Big)$

constitué par les vecteurs $\boldsymbol{\phi}(y)$ et $\boldsymbol{\phi}(y')$, on a alors :

$$\begin{aligned} y_Q(\mathbf{x}) &= \underset{y\in Y}{\operatorname{argmin}} \Big[\Big(1 - Q(\mathbf{x}, y)\Big)^2 + \sum_{y'\neq y} Q^2(\mathbf{x}, y')\Big] \\ &= \underset{y\in Y}{\operatorname{argmax}}\, Q(\mathbf{x}, y) \\ &= B_Q(\mathbf{x}) \end{aligned}$$

Le passage à l'avant dernière ligne de cette démonstration n'est qu'une simple question de logique. En effet, l'expression à droite de l'addition dans la ligne qui la précède ne dépendant pas de y, l'argument y qui minimise toute l'expression ne peut être obtenu qu'en minimisant $\Big(1-Q(\mathbf{x}, y)\Big)^2$. Or, comme il s'agit d'une soustraction, la minimisation de $\Big(1 - Q(\mathbf{x}, y)\Big)^2$ s'avère à être tout simplement la maximisation de $Q(\mathbf{x}, y)$. □

Dans le chapitre 5, on a introduit la relation entre le risque du classificateur de Bayes $R(B_Q)$ et le risque du classificateur de Gibbs noté $R(G_Q)$. D'une manière générale, cette relation signale que $R(B_Q)$ est borné supérieurement par le double de $R(G_Q)$. Nous allons donc montrer que cette relation reste maintenue dans notre nouvel espace vectoriel. Considérons d'abord le risque de Gibbs sur un exemple $(\mathbf{x}, y)$ exprimé par :

$$R_{(\mathbf{x},y)}(G_Q) = || \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) ||^2 \tag{6.4}$$

où $c > 0$. Pour comprendre la raison de cette égalité, voir l'équation 6.7.
L'objectif est de montrer que la relation $R_{(\mathbf{x},y)}(B_Q) \leq 2\, R_{(\mathbf{x},y)}(G_Q)$ est toujours vrai dans le nouvel espace vectoriel. Par simple équivalence, cela revient donc à démontrer que l'inégalité $R_{(\mathbf{x},y)}(B_Q) \leq 2 \,||\, \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) \,||^2$ est vraie. La preuve de cette inégalité est fournie par la démonstration du théorème suivant :

Théorème 12. *$\forall(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ et $\forall c > 0$, on a :*

$$R_{(\mathbf{x},y)}(B_Q) \leq 2 \,||\, \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) \,||^2$$

Démonstration.

$$\begin{aligned} R_{(\mathbf{x},y)}(B_Q) &= I\Big(y_Q(\mathbf{x}) \neq y\Big) \\ &= I\Big[\boldsymbol{\phi}\Big(y_Q(\mathbf{x})\Big) \neq \boldsymbol{\phi}(y)\Big] \\ &= \frac{1}{2}\Big[2 - 2\,\boldsymbol{\phi}(y) \cdot \boldsymbol{\phi}\Big(y_Q(\mathbf{x})\Big)\Big] \\ &= \frac{1}{2} \,||\, \boldsymbol{\phi}(y) - \boldsymbol{\phi}\Big(y_Q(\mathbf{x})\Big) \,||^2 \end{aligned}$$

Sachant que :

$$\begin{aligned} || \boldsymbol{\phi}(y) - \boldsymbol{\phi}(y_Q(\mathbf{x})) || &\leq || \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) || + || \boldsymbol{\phi}(y_Q(\mathbf{x})) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) || \quad (6.5) \\ &\leq 2 \; || \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) || \quad (6.6) \end{aligned}$$

L'équation 6.5 est justifiée par le fait que $|| \boldsymbol{\phi}(y) - \boldsymbol{\phi}(y_Q(\mathbf{x})) || \leq || \boldsymbol{\phi}(y) + \boldsymbol{\phi}(y_Q(\mathbf{x})) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) ||$ et que pour tout vecteur $\vec{u}$ et $\vec{v}$, on a $|| \vec{u} + \vec{v} || \leq || \vec{u} || + || \vec{v} ||$.
Le passage de l'equation 6.5 à l'équation 6.6 est obtenu grâce à l'équation 6.3 valable pour tout c.

Des équations 6.5 et 6.6, on déduit que :

$$R_{(\mathbf{x},y)}(B_Q) \leq 2 \; || \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) ||^2$$

□

Rappelons que $R(G_Q)$ peut être réécrit comme $\underset{(\mathbf{x},y)}{\mathbf{E}} R_{(\mathbf{x},y)}(G_Q)$ avec :

$$\begin{aligned} R_{(\mathbf{x},y)}(G_Q) &= || \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) ||^2 \\ &= || \boldsymbol{\phi}(y) - c \cdot \Big(\sum_{i=1}^{n} Q_i \boldsymbol{\phi}(h_i(\mathbf{x})) \Big) ||^2 \\ &= || Q_i \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x})) \Big] ||^2 \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j L_{i,j}(\mathbf{x}, y) \quad (6.7) \end{aligned}$$

où

$$L_{i,j}(\mathbf{x}, y) \stackrel{\text{def}}{=} \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x})) \Big] \cdot \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x})) \Big]$$

Cette dérivation justifie l'équation 6.4. En effet, avec l'équation 6.7, on remarque que $|| \boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}_Q(\mathbf{x}) ||^2$ se ramène à être un risque de Gibbs car il s'agit d'une espérance de perte sur des paires de prédicteurs (voir l'approche sample compress dans (18)).

6.4 Problème d'optimisation

Comme on l'a signalé dans l'introduction de ce chapitre, la borne PAC-Bayes dépend de $R_S(G_Q)$ et de $\mathrm{KL}(Q\|P)$. Or ici $\mathrm{KL}(Q\|P)$ est au plus $\ln(n)$ pour une distribution

sur n classificateurs. Nous pouvons donc borner ici le $\mathrm{KL}(Q\|P)$ par une constante et dans ce cas, minimiser la borne PAC-Bayes revient à minimiser seulement $R_S(G_Q)$. Si la cardinalité de S est m, notre problème d'optimisation se réduit donc à la minimisation de l'équation suivante :

$$R_S(G_Q) \;=\; \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_i Q_j \cdot L_{i,j}(\mathbf{x}_k, y_k) \tag{6.8}$$

où :

$$L_{i,j}(\mathbf{x}, y) \;=\; \Big[\boldsymbol{\phi}(y) - c\cdot\boldsymbol{\phi}(h_i(\mathbf{x}))\Big]\cdot\Big[\boldsymbol{\phi}(y) - c\cdot\boldsymbol{\phi}(h_j(\mathbf{x}))\Big]$$

et $c > 0$ est un hyperparamètre.

6.5 Le cas binaire

6.5.1 Simplification des équations

Dans le cas de la classification binaire, la valeur de $L_{i,j}(\mathbf{x}, y)$ peut prendre seulement trois valeurs différentes.

Cas 1 : Les classificateurs h_i et h_j classifient correctement l'exemple $(\mathbf{x}, y)$

$$\begin{aligned} L_1 &= \Big[\boldsymbol{\phi}(1) - c\cdot\boldsymbol{\phi}(1)\Big]\cdot\Big[\boldsymbol{\phi}(1) - c\cdot\boldsymbol{\phi}(1)\Big] \\ &= \Big[(0,1) - c\cdot(0,1)\Big]\cdot\Big[(0,1) - c\cdot(0,1)\Big] \\ &= (0, 1-c)\cdot(0, 1-c) \\ &= (1-c)^2 \;=\; c^2 - 2c + 1\,. \end{aligned}$$

Cas 2 : Les classificateurs h_i et h_j classifient incorrectement l'exemple $(\mathbf{x}, y)$

$$\begin{aligned} L_2 &= \Big[\boldsymbol{\phi}(1) - c\cdot\boldsymbol{\phi}(-1)\Big]\cdot\Big[\boldsymbol{\phi}(1) - c\cdot\boldsymbol{\phi}(-1)\Big] \\ &= \Big[(0,1) - c\cdot(1,0)\Big]\cdot\Big[(0,1) - c\cdot(1,0)\Big] \\ &= (-c, 1)\cdot(-c, 1) \\ &= c^2 + 1\,. \end{aligned}$$

Cas 3 : Les classificateurs h_i et h_j sont en désaccord sur l'exemple $(\mathbf{x}, y)$

$$\begin{aligned} L_3 &= \Big[\boldsymbol{\phi}(1) - c\cdot\boldsymbol{\phi}(1)\Big]\cdot\Big[\boldsymbol{\phi}(1) - c\cdot\boldsymbol{\phi}(-1)\Big] \\ &= \Big[(0,1) - c\cdot(0,1)\Big]\cdot\Big[(0,1) - c\cdot(1,0)\Big] \\ &= (0, 1-c)\cdot(-c, 1) \\ &= 1 - c\cdot\,. \end{aligned}$$

En résumé :

$$L_{i,j}(\mathbf{x}, y) \;=\; \begin{cases} c^2 - 2c + 1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) = y, \\ c^2 + 1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) \neq y, \\ 1 - c & \text{si } h_i(\mathbf{x}) \neq h_j(\mathbf{x}). \end{cases} \tag{6.9}$$

Reprenons le problème d'optimisation original (equation (6.8)) et appliquons une transformation linaire à tous les éléments de la matrice L (équation (6.9)), de tel sorte que le problème de minimisation soit optimal pour la même distribution Q :

$$R_S(G_Q) = \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j \cdot L'_{i,j}(\mathbf{x}_k, y_k) \tag{6.10}$$

où :

$$L'_{i,j}(\mathbf{x}, y) \;\stackrel{\text{def}}{=}\; \frac{L_{i,j}(\mathbf{x}, y) - (1 - c)}{c} \;=\; \begin{cases} c - 1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) = y, \\ c + 1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) \neq y, \\ 0 & \text{si } h_i(\mathbf{x}) \neq h_j(\mathbf{x}). \end{cases}$$

Réécrivons l'équation (6.10) :

$$\begin{aligned} R_S(G_Q) \;&=\; \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j \cdot \Big[(c-1) \cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) = y_k\big) \\ &\qquad\qquad + (c+1) \cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k\big) \Big] \\ &=\; \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j \cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big) \cdot \Big[(c-1) \cdot I(h_i(\mathbf{x}_k) = y_k) \\ &\qquad\qquad + (c+1) \cdot I(h_i(\mathbf{x}_k) \neq y_k) \Big] \\ &=\; \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j \cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big) \cdot \Big[c - I(h_i(\mathbf{x}_k) = y_k) + I(h_i(\mathbf{x}_k) \neq y_k) \Big] \\ &=\; \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j \cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big) \cdot \Big[c - y_k \cdot h_i(\mathbf{x}_k) \Big]. \end{aligned}$$

6.5.2 Ecriture sous la forme d'un problème d'optimisation quadratique

Considérons désormais le vecteur de poids $\mathbf{Q} \stackrel{\text{def}}{=} \Big(Q(h_1) \cdots Q(h_n) \Big)^T$ et une matrice $\mathbf{L}'$ de taille $n \times n$ où :.

$$\mathbf{L}'_{(i,j)} \stackrel{\text{def}}{=} \sum_{k=1}^{m} L'_{i,j}(\mathbf{x}_k, y_k) \;=\; \sum_{k=1}^{m} I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big) \cdot \Big[c - y_k \cdot h_i(\mathbf{x}_k)\Big] \,.$$

Le problème d'optimisation est donc formulé ainsi :

$$\begin{aligned} \textbf{Minimiser} \;&:\; \mathbf{Q}^T \, \mathbf{L}' \, \mathbf{Q} \\ \textbf{sous les contraintes} \;&:\; \mathbf{Q}_{(i)} \geq 0 \quad \forall i \in \{1, \ldots, n\} \\ &\quad\; \sum_{i=1}^{n} \mathbf{Q}_{(i)} = 1 \,. \end{aligned}$$

6.6 Le cas multiclasse

6.6.1 Simplification des équations

Dans le cas de la classification multiclasse, la valeur de $L_{i,j}(\mathbf{x}, y)$ peut prendre quatre valeurs différentes. Rappelons que dans le cas multiclasse , on a $\boldsymbol{\phi}(y) = (0, \ldots, 1, \ldots, 0)$, le "1" étant à la position y du vecteur. On a donc les 4 cas suivants :

Cas 1 : Les classificateurs h_i et h_j classifient correctement l'exemple $(\mathbf{x}, y)$

$$\begin{aligned} L_1 &= \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x}))\Big] \cdot \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x}))\Big] \\ &= \Big[(0, \ldots, 1, \ldots, 0) - c \cdot (0, \ldots, 1, \ldots, 0)\Big] \cdot \Big[(0, \ldots, 1, \ldots, 0) - c \cdot (0, \ldots, 1, \ldots, 0)\Big] \\ &= (0, \ldots, 1-c, \ldots, 0) \cdot (0, \ldots, 1-c, \ldots, 0) \\ &= (1-c)^2 \;=\; c^2 - 2c + 1 \,. \end{aligned}$$

Cas 2 : Les classificateurs h_i et h_j se trompent sur l'exemple $(\mathbf{x}, y)$ et $h_i(\mathbf{x}) = h_j(\mathbf{x})$

$$\begin{aligned} L_2 &= \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x}))\Big] \cdot \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x}))\Big] \\ &= \Big[(0, \ldots, 1, \ldots, 0, \ldots, 0) - c \cdot (0, \ldots, 0, \ldots, 1, \ldots, 0)\Big] \\ &\quad \cdot \Big[(0, \ldots, 1, \ldots, 0, \ldots, 0) - c \cdot (0, \ldots, 0, \ldots, 1, \ldots, 0)\Big] \\ &= (0, \ldots, 1, \ldots, -c, \ldots, 0) \cdot (0, \ldots, 1, \ldots, -c, \ldots, 0) \\ &= c^2 + 1 \,. \end{aligned}$$

Cas 3 : Les classificateurs h_i et h_j sont en désaccord sur l'exemple $(\mathbf{x}, y)$ et l'un d'entre eux a raison [2]

$$\begin{aligned} L_3 &= \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x}))\Big] \cdot \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x}))\Big] \\ &= \Big[(0, \ldots, 1, \ldots, 0, \ldots, 0) - c \cdot (0, \ldots, 1, \ldots, 0, \ldots, 0)\Big] \\ &\quad \cdot \Big[(0, \ldots, 1, \ldots, 0, \ldots, 0) - c \cdot (0, \ldots, 0, \ldots, 1, \ldots, 0)\Big] \\ &= (0, \ldots, 1-c, \ldots, 0, \ldots, 0) \cdot (0, \ldots, 1, \ldots, -c, \ldots, 0) \\ &= 1 - c\,. \end{aligned}$$

Cas 4 : Les classificateurs h_i et h_j sont en désaccord sur l'exemple $(\mathbf{x}, y)$ et se trompent en même temps

$$\begin{aligned} L_4 &= \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x}))\Big] \cdot \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x}))\Big] \\ &= \Big[(0, \ldots, 1, \ldots, 0, \ldots, 0) - c \cdot (0, \ldots, 0, \ldots, 1, \ldots, 0)\Big] \\ &\quad \cdot \Big[(0, \ldots, 1, \ldots, 0, \ldots, 0) - c \cdot (1, \ldots, 0, \ldots, 0, \ldots, 0)\Big] \\ &= (0, \ldots, 1, \ldots, -c, \ldots, 0) \cdot (-c, \ldots, 1, \ldots, 0, \ldots, 0) \\ &= 1\,. \end{aligned}$$

En résumé :

$$L_{i,j}(\mathbf{x}, y) = \begin{cases} c^2 - 2c + 1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) = y, \\ c^2 + 1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) \neq y, \\ 1 - c & \text{si } h_i(\mathbf{x}) \neq h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) = y \text{ ou } h_j(\mathbf{x}) = y, \\ 1 & \text{si } h_i(\mathbf{x}) \neq h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) \neq y \text{ et } h_j(\mathbf{x}) \neq y. \end{cases} \tag{6.11}$$

De même qu'on l'a fait pour le cas binaire, reprenons le problème d'optimisation original (équation (6.8)) et appliquons une transformation linaire à tous les éléments de la matrice L (équation (6.11)), de tel sorte que le problème de minimisation soit optimal pour la même distribution Q :

$$R_S(G_Q) = \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_i Q_j \cdot L'_{i,j}(\mathbf{x}_k, y_k) \tag{6.12}$$

2. ici on considère que c'est h_i qui a raison, si on considère l'inverse, on arrive au même résultat

où :

$$L'_{i,j}(\mathbf{x}, y) \stackrel{\text{def}}{=} \frac{L_{i,j}(\mathbf{x}, y) - (1-c)}{c} = \begin{cases} c-1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) = y, \\ c+1 & \text{si } h_i(\mathbf{x}) = h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) \neq y, \\ 0 & \text{si } h_i(\mathbf{x}) \neq h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) = y \text{ ou } h_j(\mathbf{x}) = y, \\ 1 & \text{si } h_i(\mathbf{x}) \neq h_j(\mathbf{x}) \text{ et } h_i(\mathbf{x}) \neq y \text{ et } h_j(\mathbf{x}) \neq y. \end{cases}$$

Réécrivons l'équation (6.12) :

$$\begin{aligned} R_S(G_Q) &= \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_i Q_j \cdot \Big[(c-1)\cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) = y_k\big) \\ &\qquad + (c+1)\cdot I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k\big) \\ &\qquad + I\big(h_i(\mathbf{x}_k) \neq h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k \wedge h_j(\mathbf{x}_k) \neq y_k\big)\Big] \\ &= \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_i Q_j \cdot \Big[I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big)\cdot\Big((c-1)\cdot I(h_i(\mathbf{x}_k) = y_k) \\ &\qquad + (c+1)\cdot I(h_i(\mathbf{x}_k) \neq y_k)\Big) \\ &\qquad + I\big(h_i(\mathbf{x}_k) \neq h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k \wedge h_j(\mathbf{x}_k) \neq y_k\big)\Big] \\ &= \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_i Q_j \cdot \Big[I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big)\cdot\Big(c - I(h_i(\mathbf{x}_k) = y_k) + I(h_i(\mathbf{x}_k) \neq y_k)\Big) \\ &\qquad + I\big(h_i(\mathbf{x}_k) \neq h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k \wedge h_j(\mathbf{x}_k) \neq y_k\big)\Big] \\ &= \sum_{k=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} Q_i Q_j \cdot \Big[I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big)\cdot\Big(c - y_k \cdot h_i(\mathbf{x}_k)\Big) \\ &\qquad + I\big(h_i(\mathbf{x}_k) \neq h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k \wedge h_j(\mathbf{x}_k) \neq y_k\big)\Big]. \end{aligned}$$

6.6.2 Ecriture sous la forme d'un problème d'optimisation quadratique

Considérons encore une fois le vecteur de poids $\mathbf{Q} \stackrel{\text{def}}{=} \Big(Q(h_1) \cdots Q(h_n) \Big)^T$ et une matrice $\mathbf{L}'$ de taille $n \times n$ où :.

$$\begin{aligned} \mathbf{L}'_{(i,j)} &\stackrel{\text{def}}{=} \sum_{k=1}^{m} L'_{i,j}(\mathbf{x}_k, y_k) \\ &= \sum_{k=1}^{m} \Big[I\big(h_i(\mathbf{x}_k) = h_j(\mathbf{x}_k)\big) \cdot \Big(c - y_k \cdot h_i(\mathbf{x}_k) \Big) \\ &\qquad + I\big(h_i(\mathbf{x}_k) \neq h_j(\mathbf{x}_k) \wedge h_i(\mathbf{x}_k) \neq y_k \wedge h_j(\mathbf{x}_k) \neq y_k\big) \Big] . \end{aligned}$$

Le problème d'optimisation est donc formulé ainsi :

$$\begin{aligned} \textbf{Minimiser} \quad &: \quad \mathbf{Q}^T \, \mathbf{L}' \, \mathbf{Q} \\ \textbf{sous les contraintes} \quad &: \quad \mathbf{Q}_{(i)} \geq 0 \quad \forall i \in \{1, \ldots, n\} \\ &\qquad \sum_{i=1}^{n} \mathbf{Q}_{(i)} = 1 \, . \end{aligned}$$

6.7 Algorithme SORF

Programme 7 : **SORF** *un programme quadratique pour la classification*

1: **Résoudre** $\min_{\mathbf{Q}} \quad \mathbf{Q}^T \, \mathbf{L}' \, \mathbf{Q}$
2: **sous les contraintes :** $\mathbf{Q}_{(i)} \geq 0$
3: **et :** $\sum_{i=1}^{n} \mathbf{Q}_{(i)} = 1 \quad \forall i \in \{1, \ldots, n\}$

Ce problème quadratique sera convexe seulement si $\mathbf{L}'$ est positive sémi-définie (PSD). La solution sera unique lorsque $\mathbf{L}'$ est définie positive (PD). La matrice $\mathbf{L}'$ est PSD pour $c \geq 1$ puisqu'alors chaque valeur $\mathbf{L}'_{(i,j)}$ peut être exprimée comme un produit scalaire. En effet, si on se réfère à l'équation 6.8 , on a :

$$L_{i,j}(\mathbf{x}, y) \quad = \quad \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x})) \Big] \cdot \Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x})) \Big]$$

Renommons $\Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_i(\mathbf{x})) \Big]$ par $A_i(k)$ et $\Big[\boldsymbol{\phi}(y) - c \cdot \boldsymbol{\phi}(h_j(\mathbf{x})) \Big]$ par $A_j(k)$. On a alors chaque élément $\mathbf{L}_{(i,j)}$ de la matrice $\mathbf{L}$ qui peut être exprimée par $\sum_{k=1}^{m} A_i(k) \cdot A_j(k)$. Cette somme de produits scalaire nous donne un produit scalaire. Chaque élément de la matrice L' est donc un produit scalaire, ce qui implique que L' est PSD.

Chapitre 7

Résultats empiriques

7.1 Introduction

Dans le chapitre précédent, nous avons introduit l'algorithme SORF qui est un algorithme minimisant le risque quadratique empirique de Gibbs. Dans ce chapitre, nous présentons une synthèse des résultats empiriques obtenus par l'exécution de l'algorithme SORF sur plusieurs ensembles de données. Les résultats sont comparés à ceux obtenus par Random forest et MinCq, deux algorithmes d'apprentissage reconnus pour leur excellente performance sur des données réelles. Cette étude nous permettra aussi de cerner les forces et les faiblesses de l'algorithme SORF.

7.2 Méthodologie

Afin d'évaluer la performance de l'algorithme SORF, nous l'avons d'abord exécuté sur plusieurs problèmes de classification binaire. La même méthodologie est appliquée pour chacun des problèmes de classification. Dans un premier temps, on a divisé d'une manière aléatoire les données étiquetées disponibles en un ensemble d'entraînement S et un ensemble test T [1](pour plus de détails sur la subdivision voir la section 7.2.1). Par après, on a entrainé tous les algorithmes d'apprentissage sur le même ensemble S. Pour chaque classificateur obtenu, nous calculons le risque sur l'ensemble test T. En classification multiclasse, nous avons comparé SORF à Random forest seulement, étant donné que pour l'instant, les auteurs de MinCq ne l'ont pas encore adaptés au multiclasse. Pour chacun des trois algorithmes, nous avons utilisé comme classificateurs

1. Précisons que la subdivision des ensembles S et T est effectuée une seule fois. Ainsi, la même subdivision est utilisée pour toutes les expérimentations.

de base, les arbres de décision construits en utilisant la manière suggérée par Breiman pour développer les forêts aléatoires. Pour développer chaque arbre, nous considérons un nombre f d'attributs choisis aléatoirement parmi les a attributs possibles, tel que $f = \lfloor \log_2 a+1 \rfloor$. Etant donné qu'on utilise les mêmes classificateurs de base, la différence entre Random Forests, SORF et MinCq, lors de nos experimentations, s'effectue dans la manière dont le vote de majorité est effectué. En effet, pour Random Forests, cela revient à utiliser juste une distribution uniforme sur les arbres de la forêt, alors que pour SORF et MinCq, la distribution Q sera calculée à postériori, suivant les résultats des programmes quadratiques respectifs.

7.2.1 Ensemble de données

Comme nous en avons parlé dans la section 4.3 du chapitre 4, la plupart des ensembles de données utilisés lors de nos expérimentations proviennent du répertoire «UCI», maintenu par le «Center for Machine Learning and Intelligent Systems» de l'université de Californie à Irvine [2] . Les détails sur ces ensembles de données, sont donnés en annexes (Annexe A). Les colonnes $|S|$ et $|T|$ représentent les cardinalités de l'ensemble des données d'entrainement et de l'ensemble des données test respectivement. Cependant, deux ensembles de données proviennent d'une autre source :

– L'ensemble «MNIST» a d'abord été utilisé par Yann LeCun (25) [3]. Il contient des images de caractères numériques manuscrits.
– L'ensemble «Ringnorm» est un ensemble de données synthétiques qui a d'abord été utilisé par Leo Breiman (10) [4]. Il s'agit d'un ensemble de données pour la classification binaire dont les exemples de chacune des deux classes sont générés par une distribution de données spécifique (deux lois normales multivariés de moyennes et de variances distinctes).

7.3 Resultats empiriques

Pour effectuer nos expérimentations, nous avons utilisé 100 arbres de décision pour chacun des trois algorithmes. La question qui restait donc à régler consistait donc à

2. Le «UCI Machine Learning Repository» est accessible en suivant ce lien `http://archive.ics.uci.edu/ml`.
3. L'ensemble «MNIST» est accessible en suivant ce lien : `http://yann.lecun.com/exdb/mnist/`
4. le code source utilisé pour génerer l'ensemble «Ringnorm» est accessible en suivant ce lien `http://www.cs.toronto.edu/~delve/data/ringnorm/desc.html`

déterminer les hyperparamètres pour les algorithmes SORF et MinCq. Rappelons que Random Forests, dans notre cas n'a pas besoin d'hyperparamètre. Certes, on pourrait déterminer le nombre d'arbres comme un hyperparamètre. Cependant, on a remarqué qu'autour de 100 arbres de décision, le risque empirique calculé sur les échantillons out-of-bag semblait être stable. En effet, en utilisant la méthode des échantillons out-of-bag, on a constaté qu'autour de 100 arbres, le risque empirique sur ces échantillons semblait ne plus varier même si on continuait à augmenter le nombre d'arbres. Pour choisir les hyperparamètres pour SORF (hyperparamètre c) et MinCq (hyperparamètre μ), nous avons également utilisé la méthode des échantillons out-of-bag au lieu de la validation croisée. On a préféré cette méthode car elle donne des résultats comparables à celle de la validation croisée, alors qu'elle est de loin moins coûteuse en temps de calcul. Ainsi donc, les hyperparamètres c et μ ont été choisis, en déterminant, ceux qui minimisent le risque empirique out-of-bag sur l'ensemble d'entrainement S. Pour SORF, l'hyperparamètre c est choisi parmi 20 rééls uniformément espacés dans un intervalle de 1.0 à 2.0 inclusivement. Le choix de cet intervalle est dû au fait que théoriquement, la valeur idéale de l'hyperparamètre c qui minimise le risque de Gibbs est de 1.0. Cependant, on a remarqué lors des expérimentations préliminaires, que pour des valeurs autour de 1.5, SORF donnait parfois de meilleurs résultats. On a donc étendu l'intervalle de 1.0 à 2.0. La même méthode est utilisée pour choisir μ pour l'algorithe MinCq, en considérant un intervalle de 10^{-4} à 0.05. On a choisi cet intervalle en se réferant à l'intervalle utilisé pour le choix de l'hyperparamètre μ dans (20).
Dans les sections suivantes, on présente le sommaire des résultats empiriques qu'on a obtenu après nos expérimentations. En classification binaire, pour chacun des tableaux, on présente la profondeur maximale qu'a pue atteindre les arbres pour chaque ensemble de données. Pour chacun des algorithmes on calcule le risque du classificateur par vote de majorité B_Q sur l'ensemble test désigné par $R_T(B_Q)$. On indique également les valeurs des hyperparamètres c et μ qui ont été utilisés pour SORF et MinCq respectivement. En classification multiclasse, on compare Random Forests et SORF en présentant un tableau qui cumule les résultats sur quelques ensembles de données.

7.3.1 Cas binaire : Premiers résultats empiriques

Dans notre première expérimentation, on a executé nos trois algorithmes, en laissant développer les arbres de décision jusqu'à leur profondeur maximale. Les résultats de cette expérimentation sont présentés dans le tableau 7.1. Sur les 18 ensembles de données utilisés, Random Forests semble avoir un avantage. En effet, on remarque qu'il performe

TABLE 7.1 – Sommaire des risques sur l'ensemble de test T de Random Forests, SORF et MinCq sur des arbres à profondeur maximale

Dataset	Arbre	Random Forest	SORF		MinCQ	
Nom	Profondeur max.	$R_T(B_Q)$	C	$R_T(B_Q)$	μ	$R_T(B_Q)$
Adult	80	**0.151**	1.0	0.155	0.0001	0.152
BreastW	23	**0.035**	1.053	0.041	0.0001	0.038
credit-A	41	0.123	1.0	**0.12**	0.0001	0.13
Glass	22	**0.121**	1.0	0.159	0.0001	0.206
Haberman	40	**0.273**	1.053	0.32	0.0001	0.333
Heart	25	0.184	1.158	0.184	0.0001	0.15
Ionosphere	32	0.063	1.105	0.063	0.0001	0.063
Letter :AB	33	**0.002**	1.0	0.024	0.0001	0.004
Letter :DO	46	**0.026**	1.053	0.027	0.0001	0.026
Letter :OQ	43	0.038	1.053	0.039	0.0001	**0.034**
Mnist :08	32	**0.008**	1.053	0.012	0.0001	0.009
Mnist :18	55	0.014	1.053	**0.013**	0.0001	0.013
Mnist :23	47	**0.039**	1.053	0.041	0.0001	0.039
Mushroom	22	0.0	1.0	0.0	0.0001	0.0
Sonar	23	0.173	1.159	**0.135**	0.0001	0.375
Tic-tac-toe	15	0.146	1.053	0.121	0.0001	**0.117**
USvotes	17	0.045	1.105	0.045	0.0001	0.045
Wdbc	66	**0.035**	1.0	0.042	0.0001	0.035

mieux que SORF et MinCq dans la majorité des cas. Cependant, si on se consacre sur les ensembles de données ou Random Forests performe mieux, on remarque que SORF est presque aussi bon que Random Forests, l'écart entre les risques est toujours très minime. La conclusion en est donc que lorsqu'on a des classificateurs de base forts, Random Forests semble être dans son élément et semble mieux performer que SORF et MinCq.

7.3.2 Cas binaire : Deuxièmes résultats empiriques

Dans notre deuxième expérimentation, on a executé nos trois algorithmes, en limitant nos arbres à avoir une profondeur maximale de 1. Dans ce cas, nos classificateurs de base se réduisent donc à être des decision stumps (voir section 2.5). Cette expérimentation nous a été d'abord influencé par le fait qu'on voulait examiner le comportement de l'algorithme SORF lorsqu'il est utilisé avec des classificateurs simples mais qui sont faibles et donc pouvant avoir un risque élévé. D'autre part, on avait une certaine curiosité scientifique qui nous poussait à le comparer avec MinCq sachant que MinCq est reconnu pour mieux performer avec des votants faibles. Le premier constat est que

Random Forest ne performe pas bien lorsqu'il est utilisé avec des classificateurs faibles. SORF généralement, performe mieux que Random Forest. Il n'y a que sur 5 ensemble de données où il arrive à mieux performer que SORF et MinCq, sur les 13 restants, il est généralement moins performant par rapport à SORF et MinCq. D'autre part, on remarque que SORF semble bien s'adapter à des classificateurs faibles, performant aussi bien que MinCq. L'autre constat est en rapport avec la valeur de l'hyperparamètre c. On remarque que la valeur de c est plus proche de 1.5 que de 1.0 qui est théoriquement la meilleure valeur. Ceci confirme donc notre choix d'étendre l'intervalle de l'hyperparamètre c au delà de 1.0. La conclusion de cette expérimentation est donc que SORF semble bien s'adapter aux classificateurs faibles.

TABLE 7.2 – Sommaire des risques sur l'ensemble de test T de Random Forests, SORF et MinCq sur des decision stumps

Dataset	Arbre	Random Forest	SORF		MinCQ	
Nom	Profondeur max.	$R_T(B_Q)$	C	$R_T(B_Q)$	μ	$R_T(B_Q)$
Adult	1	0.248	1.474	0.21	0.0001	**0.204**
BreastW	1	0.056	1.105	**0.044**	0.0001	0.05
Credit-A	1	0.177	1.369	**0.133**	0.0001	0.14
Glass	1	**0.168**	1.053	0.196	0.0001	0.308
Haberman	1	**0.273**	1.737	0.28	0.0474	0.28
Heart	1	**0.156**	1.105	0.17	0.0001	0.197
Ionosphere	1	0.166	1.053	**0.109**	0.0001	0.16
Letter :AB	1	0.047	1.158	**0.022**	0.0001	0.034
Letter :DO	1	**0.069**	1.158	0.104	0.0001	0.088
Letter :OQ	1	0.182	1.053	0.179	0.0001	**0.146**
Mnist :08	1	0.045	1.105	0.022	0.0001	**0.019**
Mnist :18	1	0.172	1.368	0.077	0.0001	**0.071**
Mnist :23	1	0.079	1.474	**0.067**	0.0001	0.071
Mushroom	1	0.109	1.316	0.081	0.0001	**0.074**
Sonar	1	**0.24**	1.263	0.25	0.0001	0.404
Tic-tac-toe	1	0.365	1.0	**0.338**	0.0342	0.365
USvotes	1	0.14	1.0	**0.055**	0.0001	0.055
Wdbc	1	0.063	1.053	0.06	0.0001	**0.049**

7.3.3 Cas binaire : Troisièmes résultats empiriques

Pour la troisième expérimentation, on a executé nos trois algorithmes, en limitant la moitié des classificateurs de base à avoir une profondeur maximale et l'autre moité à être des decision stumps. Après les deux premières expérimentations, on a remarqué que SORF semblait s'adapter aux classificateurs qu'on lui soummettait et semblait donc

s'adapter mieux que Random Forests. On a donc voulu savoir comment l'algorithme SORF s'adapte à des classificateurs faibles et forts. Le constat dans cette expérimentation est que d'une manière générale SORF et MinCq semblent mieux perfomer que Random Forests. En effet, on remarque que Random Forests semble toujours moins performer que SORF et MinCq dans la majorité des cas, ne pouvant que mieux faire que sur 5 ensembles sur une totalité de 18. D'autre part, SORF et MinCq semblent toujours au coude à coude.

TABLE 7.3 – Sommaire des risques sur l'ensemble de test T de Random Forests, SORF et MinCq sur des arbres à profondeur maximale et des decision stumps pris en 50-50

Dataset	Arbre	Random Forest	SORF		MinCQ	
Nom	Profondeur max.	$R_T(B_Q)$	C	$R_T(B_Q)$	μ	$R_T(B_Q)$
Adult	84	0.208	1.105	**0.154**	0.0001	0.154
BreastW	20	0.05	1.0	0.041	0.0001	**0.038**
Credit-A	37	**0.143**	1.211	0.147	0.0001	0.147
Glass	29	0.168	1.368	**0.15**	0.0001	0.252
Haberman	30	**0.273**	1.0	0.28	0.0001	0.293
Heart	31	**0.197**	1.0	0.231	0.042	0.231
Ionosphere	34	0.109	1.316	**0.074**	0.0001	0.074
Letter :AB	19	0.015	1.0	0.031	0.0001	**0.009**
Letter :DO	40	0.03	1.105	0.024	0.0001	**0.023**
Letter :OQ	49	**0.037**	1.211	0.041	0.0001	0.044
Mnist :08	36	0.013	1.053	**0.008**	0.0001	0.009
Mnist :18	47	0.033	1.053	0.017	0.0001	**0.011**
Mnist :23	66	0.043	1.263	**0.041**	0.0001	0.045
Mushroom	24	0.002	1.0	**0.0**	0.0001	0.0
Sonar	19	0.24	1.105	**0.212**	0.0001	0.346
Tic-tac-toe	16	0.355	1.105	0.136	0.0001	**0.125**
USvotes	13	0.075	1.053	0.055	0.0001	**0.05**
Wdbc	54	**0.032**	1.211	0.039	0.0001	0.039

7.3.4 Cas multiclasse

Pour les problèmes multiclasses, on a comparé les forêts aléatoires à l'algorithme SORF sur 5 ensemble de données en laissant pousser les arbres jusqu'à leur profondeur maximale. Les résultats obtenus pour SORF et Random Forests sont très similaires. On remarque que les forêts aléatoires ont quand même un très léger avantage sur l'algorithme SORF lorsqu'on laisse pousser les arbres jusqu'à leur profondeur maximale. Dans le tableau 7.4, on remarque que sur les 5 ensembles, les forêts aléatoires ont un

TABLE 7.4 – Sommaire des risques sur l'ensemble de test T de Random Forests, SORF sur quelques ensembles de données multi-classe

Dataset	Arbre	Random Forest	SORF	
Nom	Profondeur max.	$R_T(B_Q)$	C	$R_T(B_Q)$
Image	57	**0.023**	1.053	0.025
Letters	65	0.051	1.0	0.051
Sat-image	100	**0.218**	1.0	0.219
Vehicle	50	0.279	1.0	0.279
Vowel	34	**0.382**	1.105	0.384

très petit avantage sur trois de ces ensembles, alors que sur les 2 ensembles restants, SORF et les forêts aléatoires obtiennent la même performance.

Chapitre 8

Conclusion et Travaux futurs

Dans ce mémoire, nous avons présenté les travaux résultant d'une première tentative d'élaborer un algorithme basé sur les forêts aléatoires minimisant le risque quadratique de Gibbs comme le suggère l'approche PAC-Bayes. Les forêts aléatoires constituent un algorithme de l'état de l'art avec d'excellentes performances. Les arbres de décision utilisés dans les forêts aléatoires sont construits en utilisant l'algorithme CART sans effectuer d'élagage. Pour générer les arbres de décision, les forêts aléatoires utilisent une version du bagging modifiée substantiellement. Comparativement au baggging, les forêts aléatoires sélectionnent un nombre d'attributs aléatoires dans la construction des noeuds de chaque arbre au lieu d'utiliser la totalité des attributs comme le fait le bagging. Par après, un vote de majorité est effectué pour déterminer la classe majoritaire en pondérant uniformément la classification de chaque arbre de la forêt. La question à laquelle on a essayé donc de répondre était de savoir s'il existe une distribution optimale qui pourrait permettre d'avoir une performance encore meilleure que celle des forêts aléatoires. De cette question, est née l'algorithme SORF (Structured Output Random Forests). Cet algorithme s'inspire de l'approche PAC-Bayes qui pour minimiser le risque du classificateur de Bayes, minimise le risque du classificateur de Gibbs, qui constitue une fonction convexe bornant supérieurement le risque du classificateur de Bayes. Pour chercher la distribution qui pourrait être optimale, l'algorithme SORF se réduit à être un simple programme quadratique minimisant le risque quadratique de Gibbs pour chercher une distribution Q sur les classificateurs de base qui sont des arbres de la forêt. Les résultas de l'algorithme SORF montrent que lorsqu'il est utilisé avec des classificateurs forts, il est presque aussi bien performant que les forêts aléatoires, malgré un léger avantage des forêts aléatoires. Cependant, lorsque les classificateurs de base utilisés sont faibles avec un risque pouvant être élévé, l'algorithme SORF performe

mieux généralement que les forêts aléatoires. Malgré que notre objectif principal ne se focalisait pas à comparer SORF à un autre algorithme autre que les forêts aléatoires, on l'a comparé avec un autre algorithme de l'état de l'art nommé MinCQ, car les deux algorithmes semblent bien performer lorsque des classificateurs faibles sont utilisés.

A la lumière des résultats de ces travaux, nous pouvons suggérer des voies possibles à explorer :

- Pour développer l'algorithme SORF, nous avons eu recours au Structured Output Prediction. Pour notre cas, les sorties $\boldsymbol{\phi}(y)$ sont des vecteurs de type (0, 0,.., 1, 0). Il serait aussi pertinent d'explorer le cas où les sorties $\boldsymbol{\phi}(y)$ ne sont pas des vecteurs (0, 0,.., 1, 0). Les classificateurs de base seraient des arbres à sorties vectorielles qui ne sont pas contraints à être nécessairement des vecteurs unitaires 0 ou 1. Cela pourrait permettre de mieux aborder des fonctions de perte différentes (une perte hiérarchique par exemple lorsque les classes forment une taxonomie) de la perte 0-1 que nous avons utilisé.
- Nous n'avons pas effectué une analyse PAC-Bayes profonde pour l'algorithme SORF. Les arbres de décision étant dévéloppés en utilisant l'ensemble d'entrainement, la théorie PAC-Bayes autour des algorithmes par compression d'échantillons pourrait peut être apporter plus d'éclaircissements qui permettraient d'aboutir sur un nouvel algorithme plus performant. Une théorie sample compress sur l'algorithme SORF nous semble donc être un autre chemin qui vaudrait la peine d'être explorée.

Bibliographie

[1] Audibert, J., *Théorie statistique de l'apprentissage : Une approche pac-bayésienne*, 2004.

[2] Bertsimas, D. et I. Popescu, « Optimal inequalities in probability theory : A convex optimization approach », *SIAM Journal on Optimization*, 15, p. 780–804, 2000.

[3] Biau, G., F. Cérou et A. Guyader, « On the rate of convergence of the bagged nearest neighbor estimate », *J. Mach. Learn. Res.*, 11, p. 687–712, March 2010a.

[4] Biau, G., F. Cérou et A. Guyader, « Rates of convergence of the functional k-nearest neighbor estimate », *IEEE Trans. Inf. Theor.*, 56, p. 2034–2040, April 2010b.

[5] Biau, G. et L. Devroye, « On the layered nearest neighbour estimate, the bagged nearest neighbour estimate and the random forest method in regression and classification », *J. Multivar. Anal.*, 101, p. 2499–2518, November 2010.

[6] Blumer, A., A. Ehrenfeucht, D. Haussler et M. K. Warmuth, « Learnability and the vapnik-chervonenkis dimension », *J. ACM*, 36(4), p. 929–965, octobre 1989.

[7] Breiman, L. et al., *Classification and regression trees*, Chapman & Hall, New York, 1984.

[8] Breiman, L., « Heuristics of instability and stabilization in model selection », *Ann. Stat.*, 24(6), p. 2350–2383, 1996a.

[9] Breiman, L., « Bagging predictors », *Machine Learning*, 24, p. 123–140, 1996b.

[10] Breiman, L., *Bias, variance, and arcing classifiers*, rapport technique, 1996c.

[11] Breiman, L., « Random forests », *Machine Learning*, 45, p. 5–32, October 2001.

[12] Buntine, W. et T. Niblett, « A further comparison of splitting rules for decision-tree induction », *Mach. Learn.*, 8, p. 75–85, January 1992.

[13] de recherches en informatique (Toulouse), I., *L'intelligence artificielle, mais enfin de quoi s'agit il ? : des chercheurs de l'irit répondent*, Les Livrets du Service culture UPS, Université Paul Sabatier, 2001.

[14] Devroye, L., L. Györfi et G. Lugosi, *A probabilistic theory of pattern recognition*, Applications of Mathematics, 1996.

[15] Dietterich, T. G., « Ensemble methods in machine learning », *International workshop on multiple classifier systems*, p. 1–15, 2000a.

[16] Dietterich, T. G., « An experimental comparison of three methods for constructing ensembles of decision trees : Bagging, boosting, and randomization », *Machine Learning*, 40, p. 139–157, 2000b.

[17] Germain, P., A. Lacasse, F. Laviolette et M. Marchand, « PAC-Bayesian Learning of Linear Classifiers », 2009a.

[18] Germain, P., A. Lacasse, F. Laviolette, M. Marchand et S. Shanian, *From PAC-Bayes Bounds to KL Regularization*, *Advances in neural information processing systems 22*, Bengio, Y., D. Schuurmans, J. Lafferty, C. K. I. Williams et A. Culotta, éditeurs, p. 603–610, 2009b.

[19] Giguère, S., F. Laviolette et M. Marchand, *From structured output prediction to multiple-output regression : Pac-bayes risk bounds and learning algorithms*, 2013.

[20] Jean-Françis Roy, F. L. et M. Marchand, « From PAC-Bayes bounds to quadratic programs for majority votes », *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, 2011.

[21] Kuncheva, L. I., « That elusive diversity in classifier ensembles », *Lecture notes in computer science*, p. 1126–1138, 2003.

[22] Kuncheva, L. I., *Combining pattern classifiers : Methods and algorithms*, Wiley-Interscience, 2004.

[23] Lacasse, A., F. Laviolette, M. Marchand, P. Germain et N. Usunier, « PAC-Bayes bounds for the risk of the majority vote and the variance of the Gibbs classifier », *Proceedings of the 2006 conference on neural information processing systems (nips-06)*, 2007.

[24] Lacasse, A., *Bornes PAC-Bayes et algorithmes d'apprentissage*, thèse de doctorat, 2010.

[25] Lecun, Y., Y. Bengio et P. Haffner, « Gradient-based learning applied to document recognition », *Proceedings of the ieee*, p. 2278–2324, 1998.

[26] Marchand, M., *Notes de cours : Apprentissage automatique (ift-65764), chapitre 1*, Université Laval, 2007a.

[27] Marchand, M., *Notes de cours : Apprentissage automatique (ift-65764), chapitre 6*, Université Laval, 2007b.

[28] McAllester, D., « Some PAC-Bayesian theorems », *Machine Learning*, 37, p. 355–363, 1999.

[29] Platt, J. C., N. Cristianini et J. Shawe-taylor, « Large margin dags for multiclass classification », *Advances in neural information processing systems 12*, p. 547–553, 2000.

[30] Quinlan, J. R., « Induction of decision trees », *Mach. Learn.*, 1(1), p. 81–106, mars 1986.

[31] Quinlan, J. R., *C4.5 : programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[32] Schapire, R. E., Y. Freund, P. Bartlett et W. S. Lee, « Boosting the margin : A new explanation for the effectiveness of voting methods », *The Annals of Statistics*, 26, p. 1651–1686, 1998.

[33] Wooldridge, M., N. R. Jennings et D. Kinny, *A methodology for agent-oriented analysis and design*, 1999.

Annexe A

Détails sur les ensembles de données utilisées

TABLE A.1 – Description des ensembles de données pour la classification binaire

Nom	$\|S\|$	$\|T\|$	#attributs
Adult	1809	10000	14
BreastW	343	340	9
Credit-A	353	300	15
Glass	107	107	9
Haberman	144	150	3
Heart	150	147	13
Ionosphere	176	175	34
Letter :AB	500	1055	16
Letter :DO	500	1058	16
Letter :OQ	500	1036	16
Liver	170	175	6
MNIST :08	500	1916	784
MNIST :17	500	1922	784
MNIST :18	500	1936	784
MNIST :23	500	1905	784
Mushroom	4062	4062	22
Sonar	104	104	60
Tic-tac-toe	479	479	9
Usvotes	235	200	16
Wdbc	285	284	30

TABLE A.2 – Description des ensembles de données pour la classification multiclasse

Nom	$\|S\|$	$\|T\|$	#attributs	#classes
Image	1155	1155	19	7
Letters	10000	10000	16	26
Sat-image	3217	3218	36	6
Soybean	341	342	35	19
Vehicle	423	423	18	4
Vowel	495	495	10	11