1 Abréviations et conventions

1.1 Liste des abréviations

AUC : *Area Under Curve* (aire sous la courbe)
CART : *Classification And Regression Tree* (arbre de classification et de régression)
CSV : *Comma Separated Values* ([fichier de] valeurs séparées par des virgules)
DOE : *Design of Experiments* (plan d'expériences)
EDA : *Exploratory Data Analysis* (analyse exploratoire des données)
ESE : *Enhanced Stochastic Evolutionnary* ([algorithme] évolutionnaire stochastique amélioré)
LDA : *Linear Discriminant Analysis* (analyse linéaire discriminante)
LHS : *Latin Hypercube Sample* (échantillonnage par hypercube latin)
ML : *Machine Learning* (apprentissage machine)
NOLH : *Nearly Orthogonal Latin Hypercube* (hypercube latin quasi orthogonal)
PDA : *Penalized Discriminant Analysis* (analyse discriminante pénalisée)
RF : *Random Forest* (forêt aléatoire)
RF-RI : *Random Forest Random Inputs* (forêt aléatoire à [variables d']entrées aléatoires)
ROC : *Receiver Operating Characteristic* (fonction d'efficacité du récepteur)
VCS : *Version Control System* (logiciel de contrôle des versions)

1.2 Conventions utilisées

- Les références bibliographiques seront indiquées par des exposants numériques : [1], [2], [3]...
- Les références de pied de page seront indiquées par des exposants alphabétiques : [a], [b], [c]...
- Les notations scientifiques utiliseront la *notation E* : 3e –5 $= 3 \times 10^{-5}$.

2 Introduction

2.1 Propos liminaire

L'identification des macromycètes est un sujet difficile, ne devant évidemment pas être pris à la légère. Les espèces rencontrées varient considérablement d'un écosystème à un autre, d'un continent à un autre, et aucun lot de données ni ouvrage sur les champignons ne saurait couvrir toute la diversité du monde fongique.

Le lot de données mycologiques constitué dans cette étude, bien que constituant l'un des lots en libre accès les plus complets du domaine de la *data science*, n'est bien entendu pas exhaustif.

Ce lot se concentre exclusivement sur les champignons habituellement rencontrés au Nord de la France. Nombre de variétés, parfois très connues, ne sont donc pas présentes, parmi lesquelles nous pouvons par exemple citer les représentants du genre *psilocybe*, connus pour leurs propriétés psychédéliques. Certains critères pourront également varier de manière considérable selon le stade de maturité du champignon : alors que les chapeaux vert-olive de l'*Amanita phalloides* mature sont faciles à reconnaître, les spécimens jeunes sont blancs et pourraient facilement être confondus avec des espèces comestibles (par exemple du genre *Agaricus*).[1]

L'ingestion de certains de ces champignons est *mortelle*, même en faible quantité. Le diagnostic de l'intoxication fongique peut être difficile, et parfois trop tardif pour un traitement efficace. Des composés toxiques tels que les amanitines ne sont pas altérés ou détruits par cuisson ou congélation, et seront absorbés par l'intestin, avant de passer dans la circulation sanguine afin d'être filtrés par le foie, détruisant les cellules hépatiques, puis excrétées dans l'intestin, réabsorbées, refiltrées... chaque passe détruisant les cellules hépatiques ayant survécu à la précédente, dans un cycle connu sous le nom de réabsorption hépato-entérique.[2]

Il ne faut jamais, *sous aucune circonstance*, utiliser les lots de données générés par des méthodes similaires à celles de notre études dans le but de déterminer si un champignon est comestible ou non.

2.2 But de l'étude

L'identification des plantes et champignons est un problème de classification classique, qui est usuellement effectué de façon manuelle, à l'aide de clés d'identification. La plupart de ces clés sont basées sur un processus utilisant des arbres décisionnels, ce qui pourrait sembler logique car rappelant la logique en arbre de l'évolution. Quoique séduisant, cet argument rencontre certaines limites :

La première limite est le nombre de chaînons manquants. Certaines espèces sont évidemment éteintes, ce qui signifie que certaines branches et nœuds de l'arbre phylogénétique seront manquants,

ce qui peut compliquer l'analyse quand deux espèces apparentées ont un nombre élevé de chaînons et nœuds communs manquants. Certaines similarités entre espèces peuvent également ne pas être identifiables de façon macroscopique.

La seconde limite, plus profonde, est la logique inhérente au processus évolutionnaire. Deux phénomènes antagonistes sont en jeu : convergence et divergence évolutives. Ces deux phénomènes sont liés à la nécessaire adaptation des espèces à leurs environnements. La divergence évolutive explique par exemple la diversité des mammifères : les chauves-souris, baleines et chevaux sont apparentés, mais ont des aspects très dissemblables en raison de leur adaptation à des environnements très différents. D'un autre côté, la convergence évolutive explique la similarité entre l'aile de la chauve-souris et celle de l'abeille. Toutefois, malgré leur apparente dissimilarité, l'aile de la chauve-souris est plus proche de la main humaine ou de la nageoire de la baleine que de l'aile de l'abeille. La façon la plus fiable pour évaluer le processus évolutionnaire et trouver les liens phylogénétiques de la manière la plus précise possible est l'analyse des génomes : les caractéristiques visibles peuvent être trompeuses. Malheureusement, ces caractéristiques sont souvent les seules aisément identifiables.

Le troisième problème est le critère principal de la classification. Ce critère peut être lié ou non au processus évolutionnaire ou aux critères visibles, surtout si ce critère principal est vague. Le critère de comestibilité ou de non-comestibilité retenu pour les lots de données mycologiques usuellement utilisés en *data science* souffre de ce problème : il est essentiellement centré sur la toxicité contre les humains, or de nombreux mécanismes de toxicité peuvent exister, et une toxicité ou non-toxicité d'un métabolite fongique ou végétal peut être liée à des variations métaboliques très ténues entre une espèce et une autre.

Pour ces raisons parmi d'autres, la logique arborescente, bien qu'utilisée habituellement dans l'identification des champignons et des plantes, et souvent justifiée par la nature arborescente du processus évolutionnaire, pourrait ne pas nécessairement être l'approche optimale à la classification des espèces basée sur des critères macroscopiques. Le but de cette étude est notamment de déployer des algorithmes d'apprentissage machine afin d'effectuer cette tâche de classification basée sur des indices visuels limités, et d'évaluer les performances relatives de différentes stratégies de classification.

2.3 État de l'art des lots de données mycologiques

Le tout premier lot de données mycologiques en libre accès mentionné en data science est probablement le *Mushroom Dataset* créé par Jeff Schlimmer en 1987.[3]

Un lot de données plus conséquent a été publié par Dennis Wagner en 2021[4] et mis en libre accès sous le nom de *Secondary Mushroom Dataset*.

Des bases de données dédiées à la mycologie sont également disponibles,[5,6] mais leur caractère généraliste empêche leur utilisation en data science sans un processus complexe de moissonnage (*data scraping*) et de nettoyage des données (*data cleaning*).

3 Création du lot de données

3.1 Configuration matérielle et logicielle

Le code d'apprentissage machine, les méthodes d'évaluation, ainsi que cette thèse ont été rédigés sur l'équipement suivant :

- CPU : AMD Ryzen 5 5600G
- RAM : 2x16 Go DDR4-3200
- SSD : Crucial P5 M2 NVMe
- OS : Xubuntu Linux 22.04.2 LTS
- R : version 4.2.2 (2022)
- IDE : RStudio version 2022.7.2.576, "Spotted Wakerobin"
- VCS : git version 2.34.1
- Librairies : tidyverse[7] (v1.3.2), microbenchmark[8] (v1.4.9), MASS[9] (v7.3.58.2), caret[10] (v6.0.93), ?GGally?[11] (v2.1.2), twinning[12] (v1.0), rpart[13] (v4.1.19), rpart.plot[14] (v3.1.1), party[15] (v1.3.11), ranger[16] (v0.14.1), rFerns[17] (v5.0.0), Rborist[18] (v0.3.2), rmarkdown[19] (v2.20), knitr[20] (v1.41), ggpubr[21] (v0.6.0), DiceDesign[22] (v1.9), DiceEval[23] (v1.5.1), bookdown[24] (v0.32).

3.2 Principes de conception d'un lot de données synthétiques

3.2.1 Principes généraux

Un lot de données synthétiques est un lot de données généré par un algorithme, par opposition aux lots de données issus d'une collecte effectuée en "vie réelle".

Trois stratégies sont usuellement utilisées :

- Données factices (*dummy data*) : l'ensemble des données est généré aléatoirement.
- Données générées à partir de règles (*rule-based data*) : l'ensemble des données est généré suivant des lois définies au préalable (distribution, valeurs moyennes, minimales, maximales...)
- Données générées par intelligence artificielle (*AI generated*) : l'ensemble des données est généré suivant des lois extraites par l'IA suite à l'analyse d'un échantillon de données obtenues en "vie réelle".

Les données générées par ces stratégies peuvent être de types variés, que nous pouvons grossièrement regrouper en données alphanumériques (quantitatives et qualitatives), en séries temporelles, et en données d'imagerie.

Pour des raisons pratiques et de maturité des technologies disponibles à l'heure actuelle, la méthode retenue pour créer le lot de données exploité dans notre étude sera la génération de données alphanumériques à partir de règles, extraites d'ouvrages mycologiques de référence.[1,2,25]

3.2.2 Principes de génération des paramètres quantitatifs

Dans le cadre de cette étude, les variables quantitatives générées aléatoirement sont :

- La longueur du stipe L_S,
- Le diamètre du stipe D_S,
- Le diamètre du chapeau D_C.

En première approximation, nous pouvons considérer que toutes ces valeurs sont intrinsèquement liées à la croissance du champignon. Ces trois variables peuvent, dans l'absolu, être susceptibles de varier indépendamment des autres au cours de la croissance du champignon, les variables L_S, D_S et D_C obéissant alors aux lois suivantes :

$$\begin{cases} L_S = L_{S_{max}}.F_{Ls} \\ D_S = D_{S_{max}}.F_{Ds} \\ D_C = D_{C_{max}}.F_{Dc} \end{cases}$$

Avec :

- $L_{S_{max}}$, $D_{S_{max}}$ et $D_{C_{max}}$ les valeurs maximales de longueur de stipe, diamètre du stipe et diamètre de chapeau de chaque variété fongique, extraites de la littérature,
- F_{Ls}, F_{Ds}, F_{Dc} des variables générées aléatoirement dans l'intervalle $]0;1]$, et représentatives de la croissance du spécimen.

Toutefois, nos recherches bibliographiques n'ont pas permis de distinguer de différences de la cinétique de croissance de chacune de ces trois caractéristiques dimensionnells du sporophore. Nous supposerons donc, en première approximation, que la croissance du stipe en longueur et en largeur, ainsi que la croissance du chapeau s'effectuent à des vitesses identiques. Nous obtenons par conséquent :

$$F_{Ls} = F_{Ds} = F_{Dc} = F_T$$

Avec F_T un facteur représentatif de la taille globale de chaque spécimen, généré aléatoirement.

Ainsi, le problème de génération de nos trois variables aléatoires se simplifie en un problème de génération d'une seule variable aléatoire : le facteur de taille de chaque spécimen. Un certain nombre de distributions d'intérêt sont susceptibles d'être utilisées afin de générer des facteurs de taille F_T aléatoires, il convient donc de définir le cahier des charges de la distribution la plus adaptée au sujet de cette étude.

Les critères de sélection retenus afin de choisir la loi la plus appropriée sont :

- Efficience calculatoire,
- Distribution continue,
- Distribution bornée, ou aisément normalisable sur un intervalle $[0;1]$,
- Distribution asymétrique.

Le premier critère n'est, en pratique, pas un facteur limitant, les temps de calcul pour la génération d'un nombre de facteurs de taille F_T suffisant étant typiquement inférieurs à 200 ms (pour 10^6 facteurs générés) avec la plupart des distributions d'intérêt (voir figure 1).

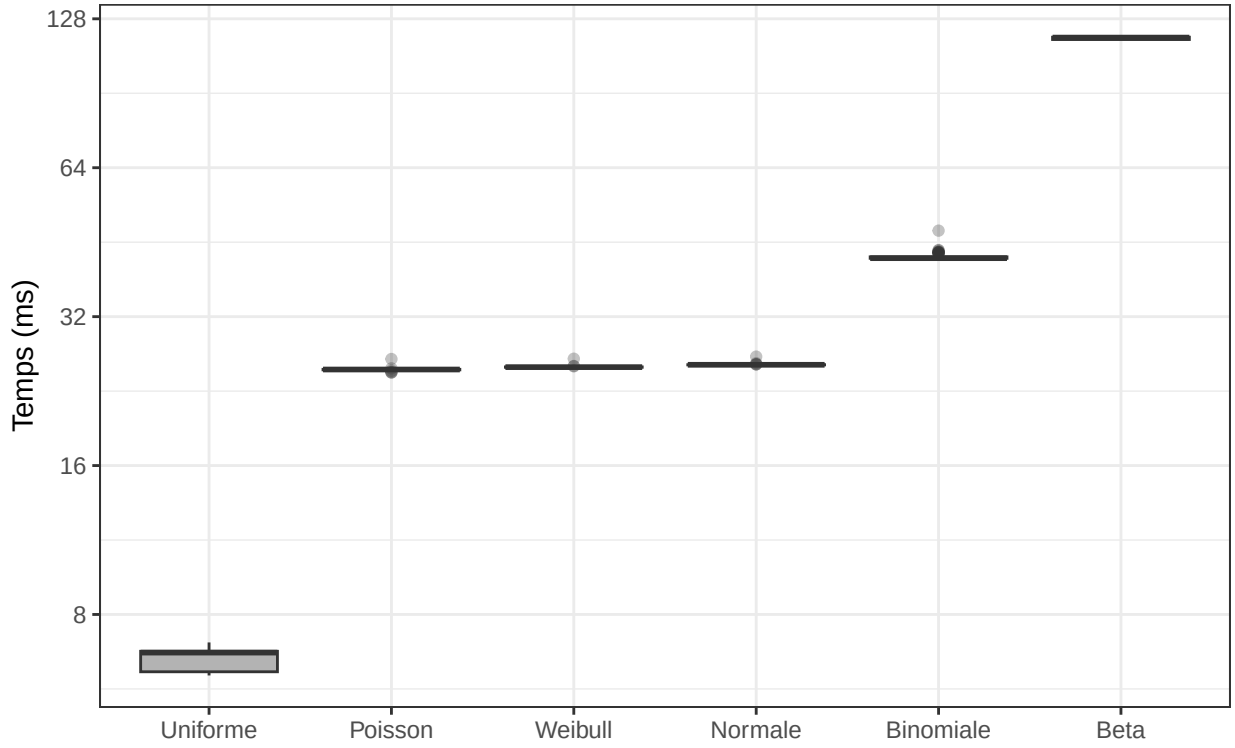


Figure 1: Temps de calcul des principales distributions d'intérêt pour 1e+06 facteurs, (100 iter.)

Les critères de continuité et de normalité n'appellent que peu de commentaires. Ces critères permettent simplement de garantir la possibilité d'une infinité de valeurs dimensionnelles, dans l'intervalle considéré. Le critère de continuité proscrit toutefois l'utilisation de lois de distributions discrètes telles que la loi binomiale ou la loi de Poisson, et celui de normalité écarte des distributions telles que la loi de Weibull, dont la normalisation est parfois délicate.

Le critère d'asymétrie est un critère permettant de tenir compte des différents paramètres pouvant impacter la distribution de taille des spécimens prélevés, parmi lesquels :

- Différences de cinétique de croissance d'une famille à une autre,
- Particularités de la croissance fongique, notamment par la croissance hyphale,[26,27]
- Probabilité de prélèvement variable selon la taille du spécimen (par difficulté de détection, considérations éthiques, intérêt mycologique ou gastronomique. . .).

Le premier paramètre évoqué précédemment n'a pu être exploité dans le cadre de cette étude en raison du manque de données concernant les cinétiques relatives de croissance des sporophores des différentes familles de macromycètes. Le modèle que nous proposons permet toutefois des développements ultérieurs dans ce domaine.

Les deux derniers paramètres permettent de supposer que la distribution de taille des spécimens d'une même espèce issus d'une récolte en vie réelle ne sera pas symétrique, d'une part en raison de la rapidité de la croissance fongique, et d'autre part parce que le prélèvement se fera préférentiellement en épargnant les spécimens de petite taille.

Ainsi, la génération de la variable aléatoire F_T obéira idéalement à une loi de distribution asymétrique vers la droite ($G_1 < 0$). Ce critère d'asymétrie écarte par conséquent les lois de distribution symétriques telles que la loi normale ou la loi uniforme.

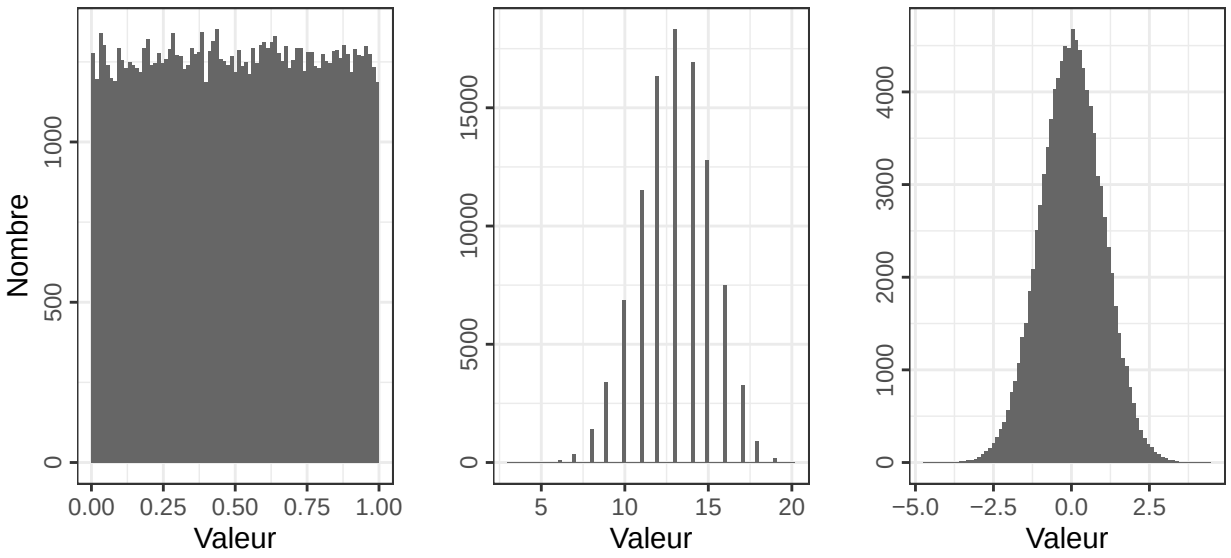


Figure 2: Exemples de distributions de la loi uniforme (à gauche), binomiale (au centre) et normale (à droite)

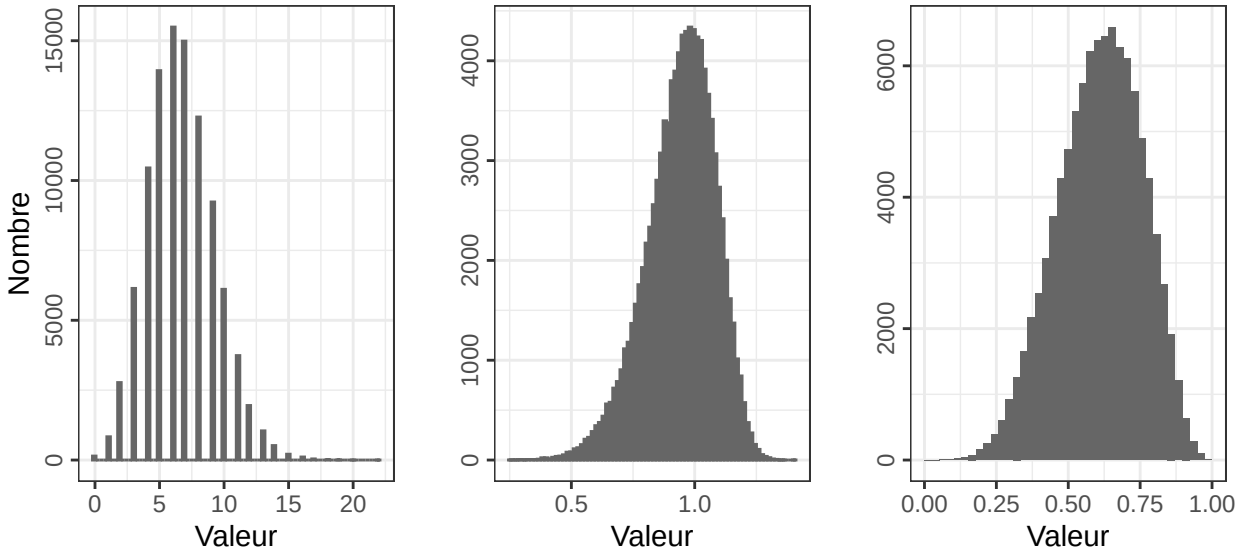


Figure 3: Exemples de distributions de la loi de Poisson (à gauche), de Weibull (au centre) et bêta (à droite)

En raison des contraintes imposées précédemment ainsi que de par sa grande polyvalence,[28] la loi retenue dans le cadre de cette étude pour la génération des facteurs de taille aléatoires (F_T) est une loi bêta non-centrale, définie comme la fonction de distribution de :[28,29]

$$X = \frac{\chi^2_{2\alpha}(\lambda)}{\chi^2_{2\alpha}(\lambda) + \chi^2_{2\beta}}$$

Avec, comme paramètres définis empiriquement pour cette étude :

$$\begin{cases} \alpha = 6\,F_c & (shape1) \\ \beta = 4 & (shape2) \\ \lambda = F_c/2 & (ncp) \end{cases}$$

F_c est ici défini comme un facteur de croissance permettant de rendre compte de la cinétique de croissance de chaque variété d'une part, et du prélèvement préférentiel des spécimens de plus grande taille d'autre part, comme l'illustre la figure 4.

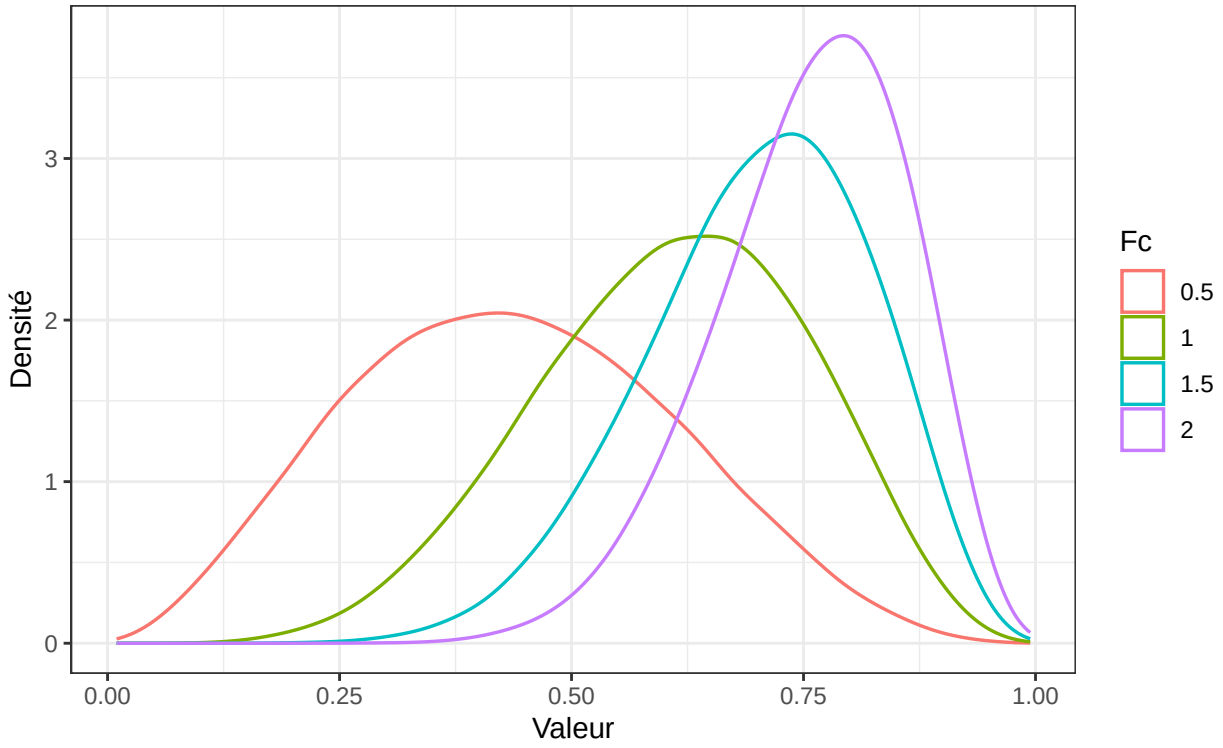


Figure 4: Distribution de différentes lois bêta, en fonction du facteur de croissance Fc

Le modèle défini à ce stade impose une stricte proportionnalité entre diamètre du chapeau D_c, diamètre du stipe D_s et longueur du stipe L_s. Dans un souci de réalisme, il apparaît souhaitable d'améliorer ce modèle mathématique en y ajoutant un facteur de dispersion, afin de proposer le modèle suivant :

$$\begin{cases} L_S = L_{Smax}.F_T.\delta_{Ls} & avec \quad \delta_{Ls} \sim \mathcal{N}(\mu = 1; \sigma = 0.05) \\ D_S = D_{Smax}.F_T.\delta_{Ds} & avec \quad \delta_{Ds} \sim \mathcal{N}(\mu = 1; \sigma = 0.05) \\ D_C = D_{Cmax}.F_T.\delta_{Dc} & avec \quad \delta_{Dc} \sim \mathcal{N}(\mu = 1; \sigma = 0.05) \end{cases}$$

L'impact de cette dispersion sur la distribution des paramètres de taille L_S, D_S et D_C est illustré par les figures 5 et 6.

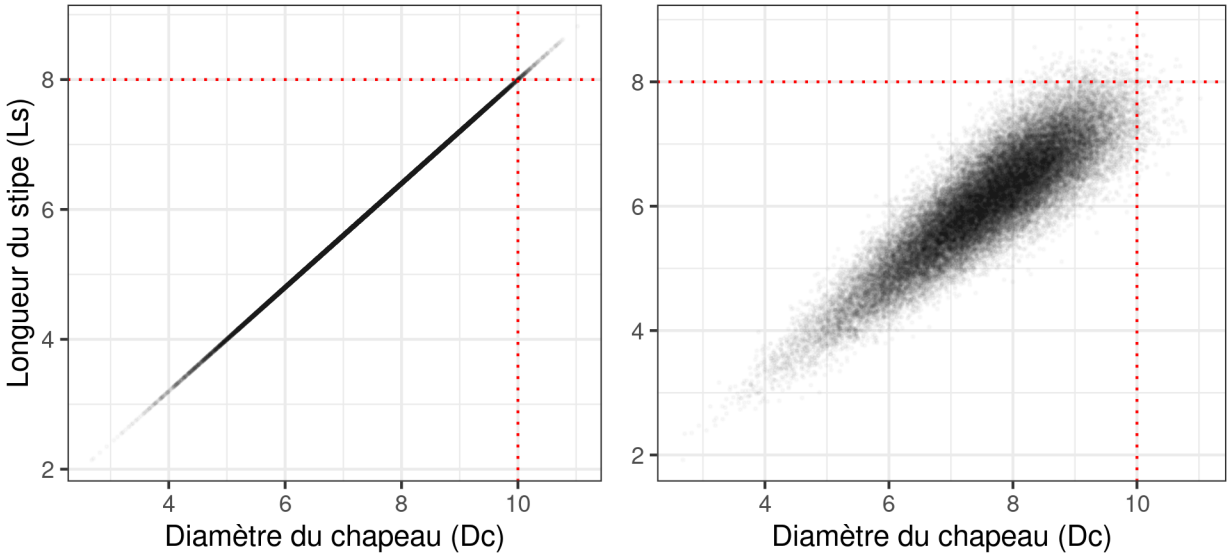


Figure 5: Nuages de points de 2 paramètres de taille (Ls et Dc), sans dispersion (à gauche) et avec dispersion (à droite), pour 50000 champignons

La dispersion ainsi créée permet de générer de légères variations des rapports entre les différents paramètres de taille, tout en se situant à proximité de la première bissectrice et majoritairement dans la zone 50-90% de la taille maximale (voir figure 6). Cette dispersion autorise par ailleurs l'existence d'une faible proportion de spécimens dépassant les valeurs dimensionnelles maximales généralement admises par la littérature.

Une simulation de Monte Carlo unidimensionnelle effectuée sur 10^5 spécimens nous permet ainsi d'évaluer la proportion de spécimens "hors normes" dépassant la valeur dimensionnelle maximale à environ 0.4 % (voir figure 7). La même simulation nous permet d'évaluer que la proportion de spécimens "exceptionnels", dépassant de plus de 10% cette valeur maximale, sera quant à elle inférieure à 0.01 %.

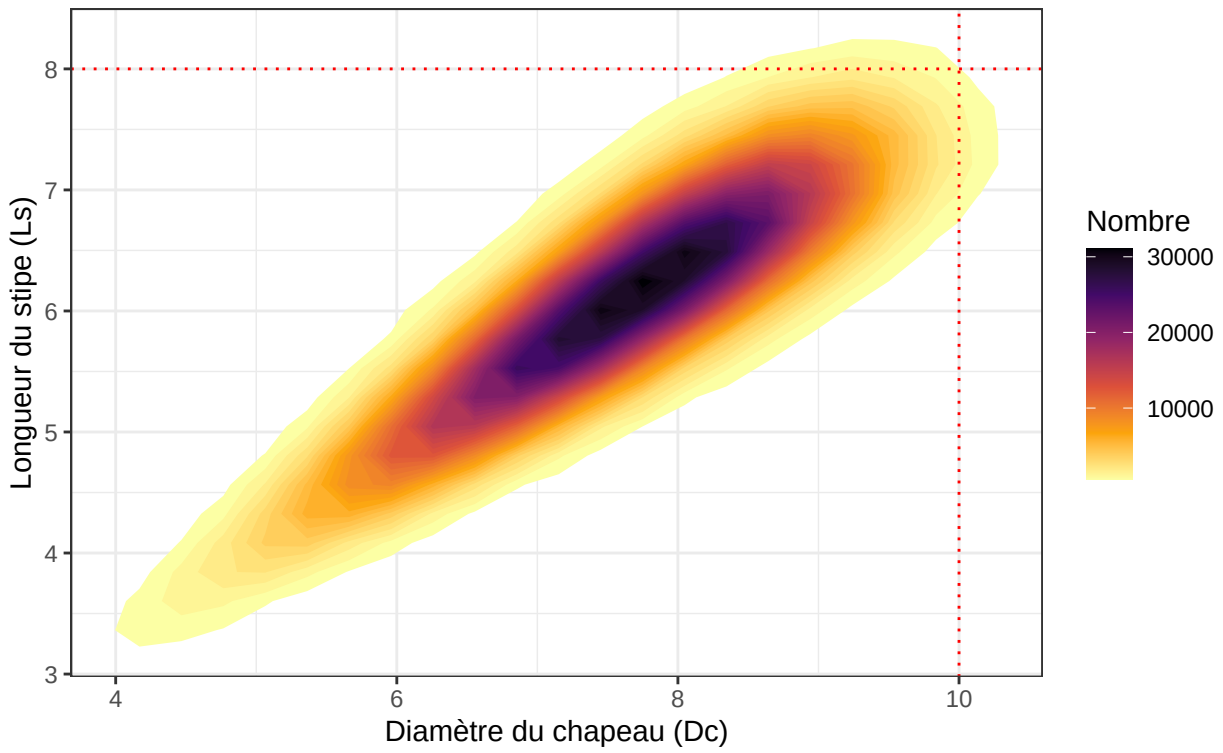


Figure 6: Diagramme de densité de 2 paramètres de taille, avec dispersion

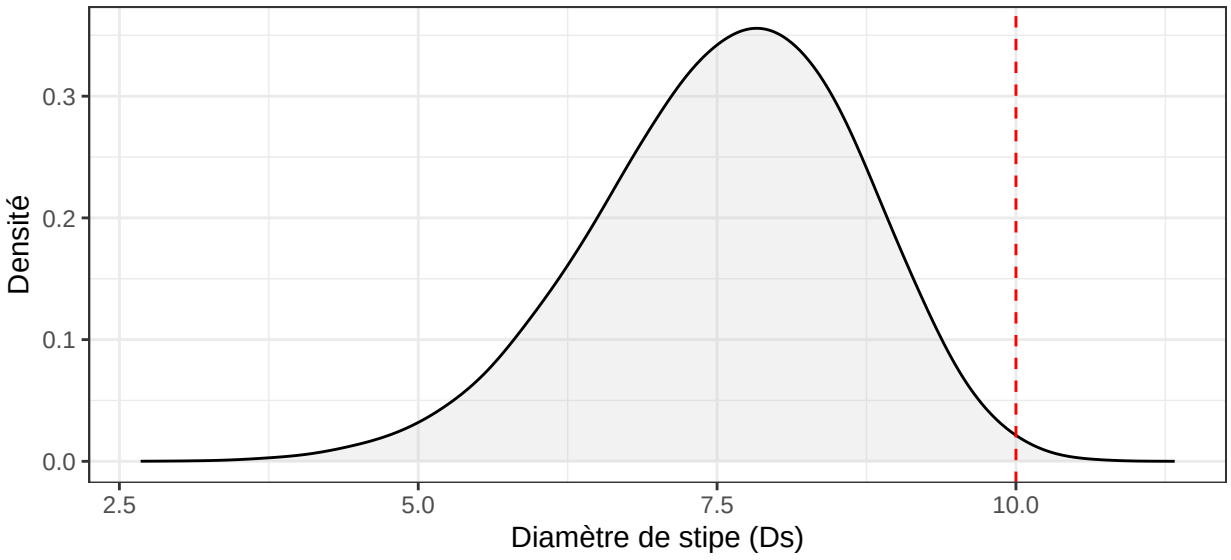


Figure 7: Distribution du diamètre de stipe Ds, pour Dsmax = 10

3.2.3 Principes de génération des paramètres qualitatifs

La génération des paramètres qualitatifs, tels que la couleur des spores ou le type d'hyménophore, est nettement moins complexe que celle des paramètres quantitatifs.

L'ensemble des valeurs quantitatives possibles pour un critère et pour une variété donnée est insérée dans un vecteur de valeurs, et une valeur sera tirée aléatoirement parmi celles contenues dans ce vecteur afin de caractériser le paramètre en question pour chaque spécimen.

4 Principes de l'apprentissage machine

L'apprentissage machine est un domaine scientifique se situant à l'interface entre les statistiques et l'informatique, et constitue un sous-domaine de l'intelligence artificielle. L'objet de l'apprentissage machine est conceptuellement de permettre aux machines d' « apprendre », c'est-à-dire d'améliorer leurs performances par l'entraînement.

En pratique, ce domaine se consacre à la génération d'algorithmes autonomes capables d'améliorer leurs performances prédictives par exposition à un lot de données et de prédire par inférence la meilleure décision ou réponse à une question.[30]

4.1 Types d'apprentissage machine

L'apprentissage machine peut se dérouler suivant un certain nombre de paradigmes, concernant en particulier la stratégie d'apprentissage – définie à partir du type de données disponibles – parmi lesquels nous pouvons notamment citer :[30,31]

- Apprentissage machine supervisé : les données du problème sont disponibles sous forme de couples exemple-annotation. Chaque point de données possède des caractéristiques (covariables) et une annotation. Le principe de cet apprentissage est donc de générer une application liant les vecteurs des entrées (covariables) et une variable de sortie (annotation). L'apprentissage machine supervisé permet également de mesurer les performances du modèle obtenu.
- Apprentissage machine semi-supervisé : les données du problème contiennent une petite quantité de données annotées, associée avec une grande quantité de données non annotées. La présence de cette grande quantité de données non-annotées peut apporter une amélioration considérable de la précision des prédictions par rapport à l'utilisation d'un seul lot de données annotées de plus petite taille.
- Apprentissage machine non supervisé : les données ne sont pas annotées. Ce paradigme permet d'acquérir une connaissance concise de la structure du lot de données, notamment par des méthodes de partitionnement des données (*clustering*), de réduction de dimensionalité ou de réseaux neuronaux. Il peut notamment être utilisé pour l'analyse exploratoire de données ou pour diminuer la quantité de données fournie à des algorithmes supervisés afin de d'augmenter l'efficience calculatoire du système.

4.2 Jeux de données

Les jeux de données dédiés à l'apprentissage machine supervisé ou semi-supervisé sont tous construits sur la base de couples données-résultats. Selon l'étape de ces types d'apprentissage machine, le résultat peut être fourni à la machine ou lui être caché, le but étant dans le premier cas de permettre à la machine d'effectuer son apprentissage, et dans le second cas d'évaluer les performances de la prédiction par rapport au résultat réel.

Le déroulement de l'apprentissage machine supervisé se décompose conceptuellement en trois étapes principales, mettant en jeu trois lots de données distincts :

1. Entraînement : le modèle d'apprentissage machine est exposé à un *jeu de données d'entraînement* (*training data set*), censé être représentatif[a] des données auquel le modèle sera exposé en utilisation réelle. Cette phase est la phase d'apprentissage du modèle.
2. Validation : le modèle d'apprentissage machine élaboré à l'étape précédente, est ici soumis à un *jeu de données de validation* (*validation data set*) et tentera d'apporter des prédictions quant à une variable d'intérêt considérée comme le résultat (ex: comestibilité, espèce...), sur la base des informations contenues dans le lot de données de validation (ex : dimensions, couleurs, morphologie du champignon...). Ces prédictions sont comparées avec les valeurs réelles (ex : comestibilité, espèce...), ce qui permet d'évaluer les performances du modèle proposé en fonction des indicateurs retenus (spécificité, sensibilité, indice de Rand, temps de calcul...). Les étapes d'apprentissage et de validation sont répétées de manière itérative en explorant l'ensemble des paramètres de configuration du modèle (hyperparamètres) à fins d'optimisation.
3. Test : les performances du meilleur modèle (avec hyperparamètres optimaux), sélectionné à l'issue de l'étape de validation, sont évaluées vis-à-vis d'un *jeu de données test* (*test* ou *holdout data set*).

La séparation entre étapes d'optimisation et de test peut sembler artificielle. Le problème est en partie lié à un flou sémantique : si l'étape initiale d'entraînement ou d'apprentissage ne pose que peu de problèmes conceptuels, l'étape intermédiaire, dite de *validation* correspond en réalité à une étape d'*optimisation* du modèle et de ses hyperparamètres. Par ailleurs, l'étape finale de *test* sera parfois qualifiée d'étape de *validation* dans la littérature, ce qui peut entretenir la confusion entre ces étapes.[32]

Une distinction sémantique plus nette entre phases d'*apprentissage*, d'*optimisation* et de *test* permet de comprendre plus aisément le fondement épistémologique de cette dernière phase pouvant parfois sembler superflue : l'optimisation effectuée lors de l'étape de validation aboutit à un modèle potentiellement biaisé par surajustement (problème dit d'*overfitting*) vis-à-vis du jeu de données utilisé comme référence lors de cette étape. Seule une exposition du modèle à des données n'ayant jamais servi à son entraînement ou son optimisation permettra réellement d'évaluer avec précision son caractère prédictif, donc sa validité.[b]

Les phases d'entraînement, d'optimisation et d'évaluation utilisent chacune un lot de données spécifique. Chacun de ces lots de données est habituellement obtenu suite à dichotomies successives (voir figure 8) du lot de données initial, avec des proportions pouvant être variables d'une scission à l'autre :

[a] Voir section 3.2.1

[b] Dans un souci de clarifier le propos, nous utiliserons les termes de lots et de phases d'entraînement, d'optimisation et d'évaluation dans la suite de cette étude.

1. Découpage du jeu de données initial, en un jeu d'évaluation d'une part, et un jeu d'entraînement et optimisation d'autre part,
2. Découpage du jeu de données entraînement et optimisation, en un jeu d'entraînement et un jeu d'optimisation.

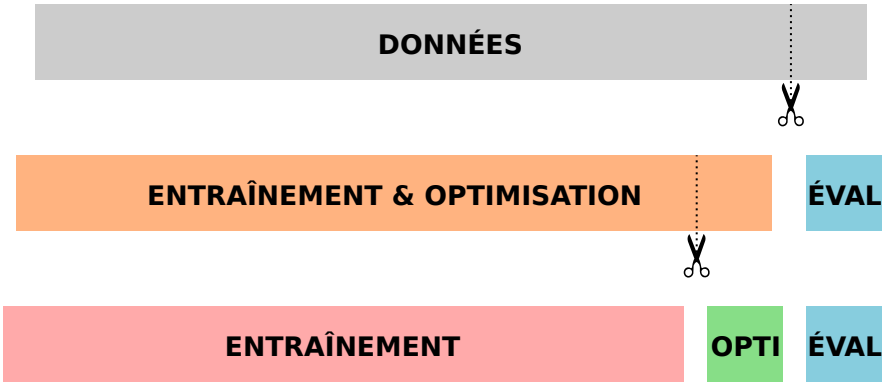


Figure 8: Principe de scissions successives d'un jeu de données initial en jeux d'apprentissage, d'optimisation et d'évaluation.

Les rapports de taille entre jeux de données d'entraînement, d'optimisation et d'évaluation de cette étude suivront la loi $p : \sqrt{p} : \sqrt{p} + 1$, avec p le nombre de coefficients du modèle.[33]

Ce nombre p de coefficients peut être approché par l'expression $p \approx \sqrt{N_u}$, avec N_u le nombre de lignes uniques de notre jeu de données, c'est-à-dire, dans notre cas, le nombre de champignons contenus dans le lot de données.[33]

Il est possible d'obtenir ce rapport $p : \sqrt{p} : \sqrt{p} + 1$ par une première scission entre jeu d'entraînement et optimisation d'une part (de taille relative $p + \sqrt{p}$) et jeu d'évaluation d'autre part (de taille relative $\sqrt{p} + 1$), avec, pour ce dernier, une taille représentant la fraction du lot total :

$$f_{test} = \frac{\sqrt{p} + 1}{p + 2\sqrt{p} + 1} = \frac{1}{\sqrt{p} + 1}$$

Cette première dichotomie peut être suivie par une seconde dichotomie entre jeu d'entraînement (de taille relative p) et jeu d'optimisation (de taille relative $\sqrt{p}$), de fraction :

$$f_{opti} = \frac{\sqrt{p}}{p + \sqrt{p}} = \frac{1}{\sqrt{p} + 1} \Rightarrow f_{test} = f_{opti} = \frac{1}{\sqrt{p} + 1}$$

En pratique, pour notre lot de données contenant $N_u = 61069$ spécimens, nous pouvons calculer $p \approx 247.1$, soit deux dichotomies successives de ratio 16:1.

4.3 Méthodes de construction des jeux de données

Les méthodes de division mises en œuvre dans cette étude appellent quelques précisions, car elles apportent certaines améliorations par rapport à l'utilisation de deux scissions successives effectuées de manière purement aléatoire.

La première division, entre jeu d'entraînement/optimisation et jeu d'évaluation, mettra en œuvre une méthode de découpage basée sur les points-supports[34] (*support-points based splitting*) exploitant un algorithme du plus proche voisin (*nearest neighbour*), afin d'optimiser la représentativité des jeux de données par rapport à ceux pouvant être obtenus par un découpage aléatoire.[35,36]

Notre seconde division, entre jeu d'entraînement et d'optimisation, exploitera quant à elle la méthode de validation croisée à k blocs (*k-folds cross-validation*). Le principe de la validation croisée repose sur une rotation de la séparation créée entre jeux d'entraînement et d'optimisation (voir figure 9).

Le jeu d'entraînement/optimisation est découpé, de façon aléatoire, en k blocs de données de taille égale, dont k-1 sont utilisés pour l'entraînement du modèle prédictif et 1 pour son optimisation. Cette opération est répétée k fois, en utilisant un jeu d'optimisation différent à chaque itération. L'évaluation de la performance globale s'effectue en évaluant la performance moyenne des k itérations. Cette méthode permet de limiter les biais potentiels générés par une simple dichotomie des données d'entraînement et d'optimisation en exploitant la totalité des données du lot afin d'effectuer ces deux tâches.

Comme démontré précédemment, une validation croisée *k-folds* avec $k = 17$ permettrait d'optimiser l'apprentissage et l'optimisation des modèles de cette étude.[33]

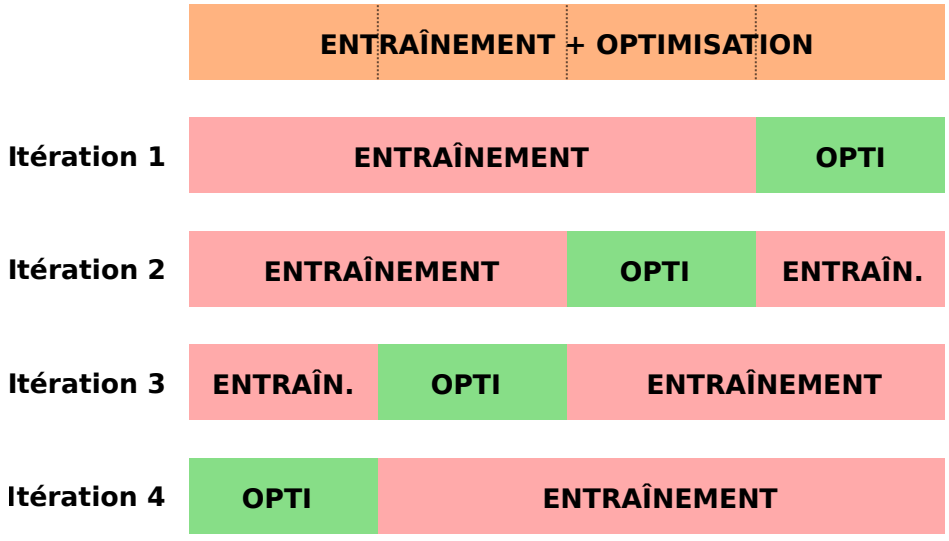


Figure 9: Principe de la validation croisée à k blocs (*k-fold cross-validation*), pour $k = 4$.

4.4 Modèles utilisés

4.4.1 Analyses discriminantes

Cette étude proposera plusieurs classifieurs s'appuyant sur des méthodes d'analyse discriminante, en particulier l'analyse discriminante linéaire (LDA : *Linear Discriminant Analysis*).

L'analyse discriminante linéaire est une méthode ayant été proposée par Ronald Fisher en 1936[37,38] pour résoudre des problèmes de classification taxonomique dans le domaine de la botanique.[c] La LDA est basée sur la construction de l'hyperplan de projection permettant de maximiser la distance entre les moyennes projetées des différentes classes et de minimiser la variance intraclasse (voir figure 10).[39] La LDA peut être utilisée à fins de classification, mais aussi pour effectuer des réductions de dimensionnalité ou encore afin de faciliter l'interprétation de l'importance de certaines caractéristiques.

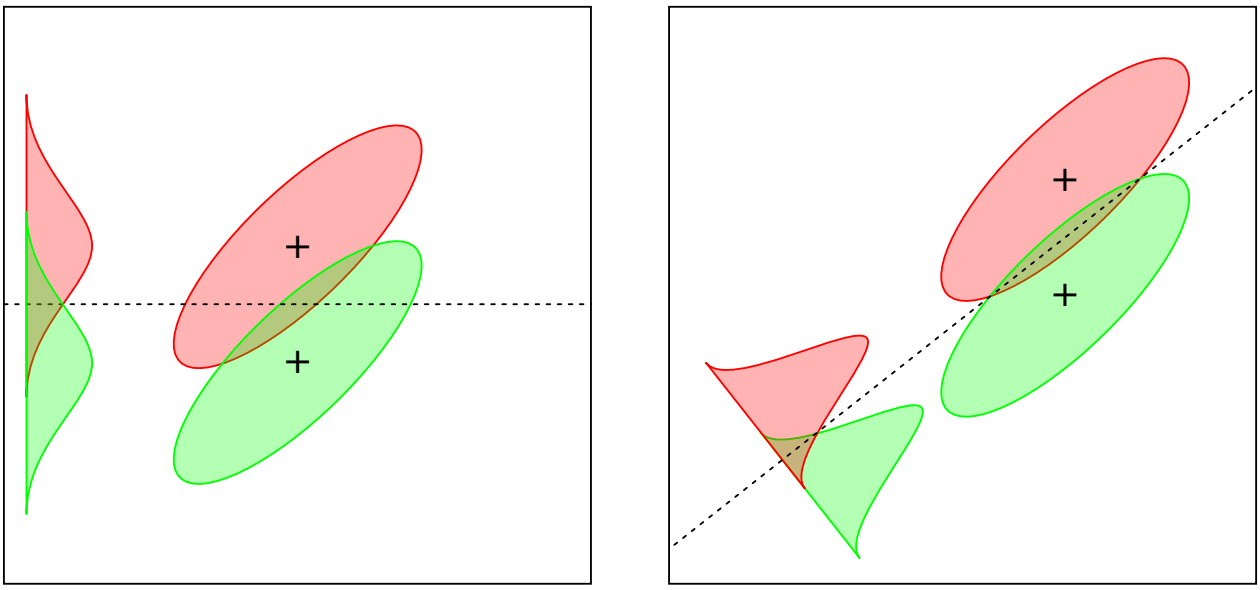


Figure 10: Séparation par distance maximale des moyennes interclasses (à gauche), et par projection sur l'hyperplan optimal tenant compte des variances intraclasses (LDA, à droite)

En pratique, la LDA consiste à construire un indice synthétique, combinaison linéaire des caractéristiques des classes, dont les coefficients permettent de rendre les points du problème originel le plus aisément "séparables". La LDA étant utilisée dans cette étude pour construire un classifieur binaire, c'est ce type de classifieur qui sera présenté dans cette section, et illustré avec

[c] Cette étude, proposant une méthode de classification des variétés *Iris setosa*, *Iris virginica* et *Iris versicolor* est par ailleurs à l'origine du célèbre jeu de données *Iris*.

un exemple extrait du jeu de données *Iris*, avec séparation entre l'espèce *Iris versicolor* et *Iris setosa*.

Dans ce cadre, la LDA vise ainsi à définir la fonction linéaire :

$$X = \sum_{i=1}^{n} \lambda_i . x_i$$

avec n le nombre de paramètres caractérisant les individus, x_i les caractéristiques mesurées pour chaque individu et chaque paramètre i, et λ_i des coefficients à optimiser, de sorte que la fonction X maximise le rapport entre les différences des moyennes de chaque classe D et la somme des produits des caractéristiques intraclasses S (proportionnelle à la variance intraclasse), définis par :

$$D = \sum_{i=1}^{n} \lambda_i . d_i$$

avec d_i la différence des caractéristiques moyennes pour chaque paramètre i, et

$$S = \sum_{p=1}^{n} \sum_{q=1}^{n} \lambda_p . \lambda_q . S_{pq}$$

avec S_{pq} la somme des produits des caractéristiques intraclasses pour chaque combinaison de paramètres p et q.

L'application sur les espèces *iris versicolor* et *iris setosa* nous donne les résultats présentés dans les tables 1 et 2 :

Table 1: Moyennes et différences de moyennes des 4 paramètres d'Iris setosa et versicolor

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
diff.	-0.930	0.658	-2.798	-1.080

Table 2: Produits des différences à la moyenne des 4 paramètres d'Iris setosa et versicolor (S_{pq})

	S.l	S.w	P.l	P.w
S.l	19.1434	9.0356	9.7634	3.2394
S.w	9.0356	11.8658	4.6232	2.4746
P.l	9.7634	4.6232	12.2978	3.8794
P.w	3.2394	2.4746	3.8794	2.4604

La maximisation du rapport entre les carrés des distances des moyennes interclasses et les variances intraclasses revient à maximiser D^2/S pour chaque coefficient λ_i soit, par dérivation pour chacun des λ_i :

$$\frac{\partial}{\partial\lambda_i}\frac{D^2}{S} = 0 \Leftrightarrow \frac{1}{S}\frac{\partial}{\partial\lambda_i}D^2 + D^2\frac{\partial}{\partial\lambda_i}\frac{1}{S} = 0 \Leftrightarrow \frac{D}{S^2}\left(2S\frac{\partial D}{\partial\lambda_i} - D\frac{\partial S}{\partial\lambda_i}\right) = 0 \Leftrightarrow \frac{1}{2}\frac{\partial S}{\partial\lambda_i} = \frac{S}{D}\frac{\partial D}{\partial\lambda_i}$$

En supposant que les distributions des classes soient unimodales, cette équation admet une solution unique. Le rapport S/D étant un facteur supposé constant pour tous les coefficients λ_i inconnus, ces coefficients sont donc les solutions du système :

$$\begin{cases} d_1 = S_{11}\lambda_1 + S_{12}\lambda_2 + S_{13}\lambda_3 + S_{14}\lambda_4 \\ d_2 = S_{21}\lambda_1 + S_{22}\lambda_2 + S_{23}\lambda_3 + S_{24}\lambda_4 \\ d_3 = S_{31}\lambda_1 + S_{32}\lambda_2 + S_{33}\lambda_3 + S_{34}\lambda_4 \\ d_4 = S_{41}\lambda_1 + S_{42}\lambda_2 + S_{43}\lambda_3 + S_{44}\lambda_4 \end{cases} \Rightarrow \mathbf{S}.\lambda = \mathbf{D} \Leftrightarrow \lambda = \mathbf{S^{-1}}.\mathbf{D}$$

avec $\mathbf{S}$ la matrice des produits S_{pq}, $\mathbf{D}$ le vecteur des différences des moyennes d_i et $\boldsymbol{\lambda}$ celui des coefficients λ_i.

En indiçant les facteurs :

- $i = 1$ pour la longueur de sépale L_s,
- $i = 2$ pour la largeur de sépale ℓ_s,
- $i = 3$ pour la longueur de pétale L_p,
- $i = 4$ pour la largeur de pétale ℓ_p.

Nous pouvons calculer les coefficients :

$$\begin{cases} \lambda_1 = 0.0311507 \\ \lambda_2 = 0.1839077 \\ \lambda_3 = -0.222104 \\ \lambda_4 = -0.3147364 \end{cases}$$

Soit, après normalisation sur le facteur λ_1 :

$$\begin{cases} \lambda_1 = 1 \\ \lambda_2 = 5.904 \\ \lambda_3 = -7.13 \\ \lambda_4 = -10.104 \end{cases}$$

$$X = L_s + 5.904.\ell_s - 7.13.L_p - 10.104.\ell_p$$

Le seuil de séparation est alors défini par :

$$X_{sep.} = \frac{\overline{X_{ver.}} + \overline{X_{set.}}}{2}$$

La valeur absolue des coefficients λ_i calculés précédemment nous indique la pondération de chaque caractère dimensionnel dans l'indice synthétique X permettant d'obtenir une séparation optimale, ainsi que l'illustrent les figures 11 et 12.

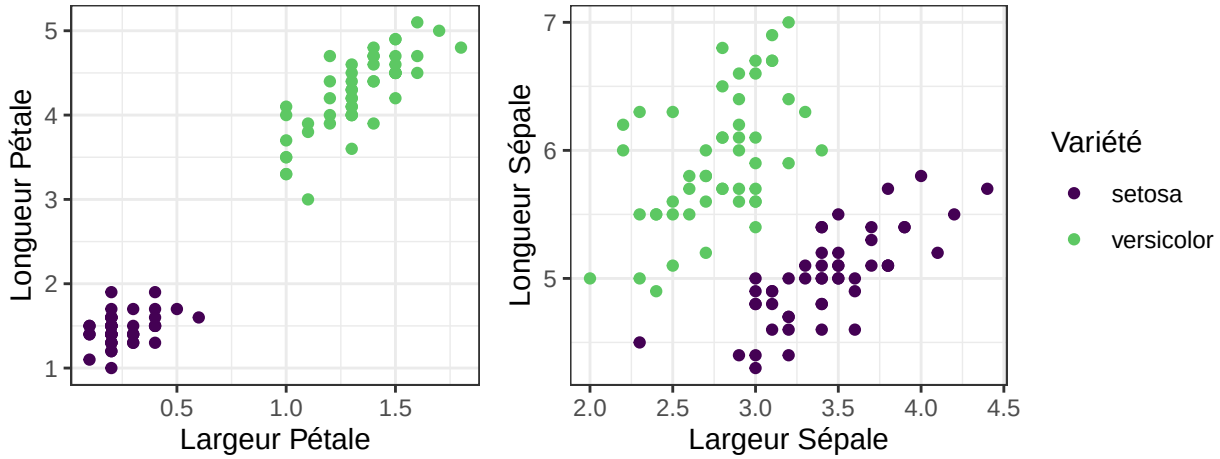


Figure 11: Distribution des variétés setosa et versicolor en fonction de leurs caractéristiques (paramètres fortement pondérés à gauche, faiblement pondérés à droite)

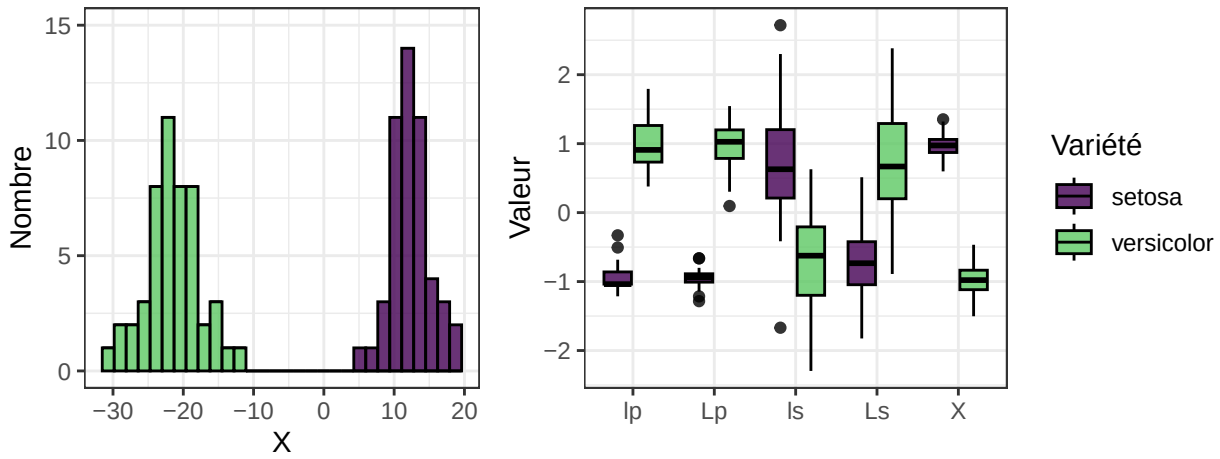


Figure 12: Distribution de X (à gauche) et des paramètres dimensionnels normalisés (à droite) en fonction des espèces

Nous pouvons conclure cette présentation de la LDA en évoquant ses limites, soulignées par certains développements de cette section :

- Les modèles d'analyse discriminante ne sont adaptés qu'à des données quantitatives ou qualitatives ordinales,
- Dans la LDA, la séparation entre les différentes classes est basée sur la construction d'un indice synthétique purement linéaire : sur l'hyperplan de projection, la séparation entre "territoires" des différentes classes sera toujours une droite,
- La plupart des modèles d'analyse discriminante se basent sur des hypothèses tacites concernant les distributions des spécimens, notamment normalité et homoscédasticité.

4.4.2 Arbres de décision

Les arbres de décision sont l'une des formes les plus populaires de représentation des connaissances utilisées dans le *data mining*.[40] Une des raisons de cette popularité est que les modèles obtenus sont généralement considérés comme performants, et que l'apprentissage comme les prédictions sont aisément compréhensibles, contrairement à certaines approches dont la complexité peut les rapprocher de modèles assimilables à des boîtes noires.[41]

Les arbres de décision peuvent aussi bien être utilisés pour des problèmes de classification (prédictions de classes) que de régression (prédiction de valeurs numériques). Les arbres de décision sont des structures hiérarchiques, séquentielles, composées de nœuds reliés par des branches. Ces nœuds sont conceptuellement de deux types :

- Les nœuds internes, qui possèdent des descendants ; le premier d'entre eux, dépourvu de prédécesseur, est qualifié de racine. A chacun de ces nœuds est associé un test, à deux ou plusieurs issues, dont chacune constitue une branche qui aboutira au nœud suivant.
- Les nœuds terminaux, dépourvus de descendants, qualifiés de feuilles, responsables de la prédiction : classe (arbre de classification) ou valeur, voire modèle numérique (arbre de régression).

Le fonctionnement de l'arbre de décision s'articule donc sur une succession de tests – les plus importants se situant près de la racine – dont les résultats permettent d'avancer dans la hiérarchie de l'arbre, étape après étape, afin d'aboutir à une décision finale, dans une démarche pouvant mimer certains aspects du raisonnement humain.[41]

Les tests des arbres de décision peuvent être :

- Univariés, usuellement sous forme de test d'inégalité de type $x_i < S$ avec x_i la caractéristique à mesurée et S un seuil permettant la décision. Ce test d'inégalité aboutit typiquement à une bifurcation, mais certains modèles peuvent regrouper plusieurs tests sur une caractéristique unique pour aboutir à une multifurcation.
- Multivariés, qui exploitent plus d'une caractéristique. Le cas le plus habituel est la scission oblique (*oblique split*), basée sur un hyperplan représentant une combinaison linéaire de caractéristiques. Ce type de test multivarié se rapproche ainsi de la LDA.[d]

Les différents modèles d'arbres de classification pourront mettre en œuvre des tests purement univariés (ID3, C4.5, CHAID...), purement obliques (arbres dits *perceptron*) ou autoriser les deux types de tests (arbres CART : *Classification And Regression Trees*).[42]

L'algorithme le plus utilisé pour la construction de la structure des arbres de classification est l'induction du haut vers le bas, basée sur l'algorithme classique dit "diviser pour régner" (*divide and conquer*), qui consiste à découper le problème de base en sous-problèmes, puis à résoudre ces sous-problèmes avant de les combiner en une solution globale.

[d] cf. section 4.4.1, page 33

En pratique, un nœud racine est tout d'abord créé, auquel sera associé la totalité du jeu d'apprentissage. En partant de ce nœud initial sera appliqué un algorithme récursif qui essaie de diviser ces données à chaque nœud. A chaque étape de cet algorithme, l'opportunité d'une scission est évaluée et chaque nœud créé pourra soit être marqué en tant que feuille (auquel cas l'induction est terminée pour cette branche), soit en tant que nœud interne (auquel cas l'algorithme se poursuit et l'arbre continuera à s'étoffer). Notons qu'à chaque étape de cet algorithme, la taille du jeu de données disponible pour les nœud descendants se réduira en raison de la scission, et par conséquent, la probabilité que ces descendants deviennent des feuilles augmentera.

L'optimalité du critère de scission appliqué par chaque nœud est usuellement obtenue en minimisant un critère dit d'*impureté*, représentatif :

- De l'inhomogénéité des classes peuplant chaque nœud postérieur à la scission pour les arbres de classification,
- De la somme de la valeur absolue des résidus ou de leurs carrés pour les arbres de régression.

Pour l'arbre CART qui sera utilisé dans la classification binaire cette étude, le critère d'impureté mise en œuvre au sein du modèle est le coefficient de Gini :[40]

$$G = 1 - p^2 - (1 - p)^2$$

avec p la fréquence relative de l'une des deux classes, au sein du nœud considéré.

Les arbres obtenus peuvent enfin être rationalisés à l'aide d'un algorithme d'élagage (*pruning*) visant à réduire la complexité de l'arbre. Cet algorithme effectue l'analyse de l'arbre des feuilles vers la racine – donc dans le sens inverse de l'induction qui l'a construit – et en évaluant si chaque nœud intermédiaire pourrait être avantageusement remplacé par une feuille ou par une branche. Différents critères d'optimisation existent, parmi lesquels nous pouvons citer l'élagage coût-complexité,[43] exploité par le modèle CART, et qui vise à minimiser le facteur coût-complexité $CC(T)$ de notre arbre T :

$$CC(T) = Erreur(T) + \alpha \times Taille(T)$$

Avec α un hyperparamètre de complexité[e] strictement positif évalué expérimentalement.[f]

Les résultats de la mise en œuvre d'une classification par arbre de décision sur le jeu de données Iris sont représentés par la figure 13.

Des exemples d'application sur les macromycètes sont également illustrés par les figures 14 et 15, qui permettent d'appréhender la potentielle complexité de la classification des champignons, même en limitant la prédiction à un critère binaire (comestible ou toxique).

[e] cf. section C.3.1, p. 83

[f] cf. section 4.5, p. 43

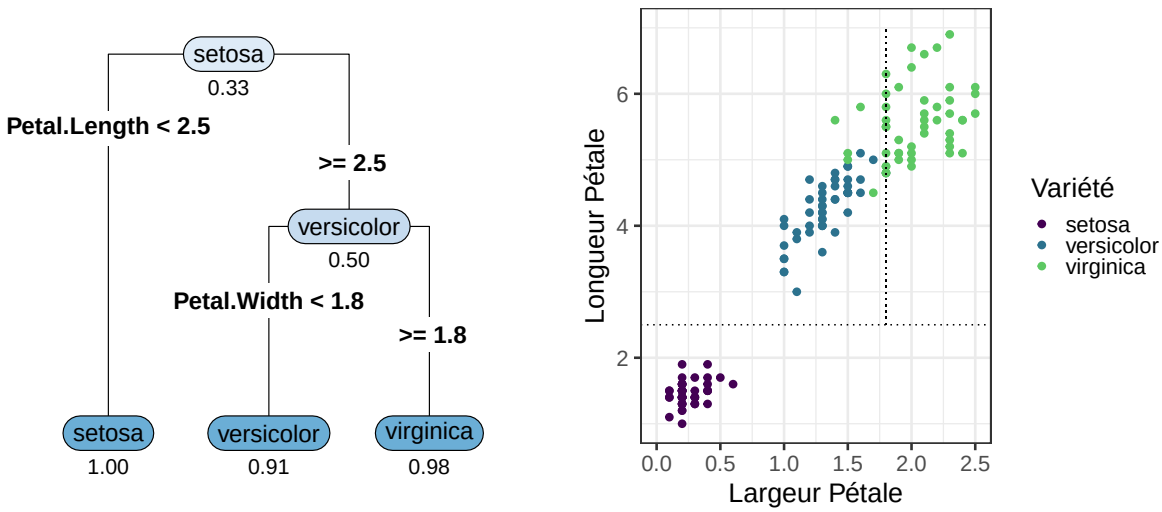


Figure 13: Arbre et critères de classification des trois espèces du lot de données Iris

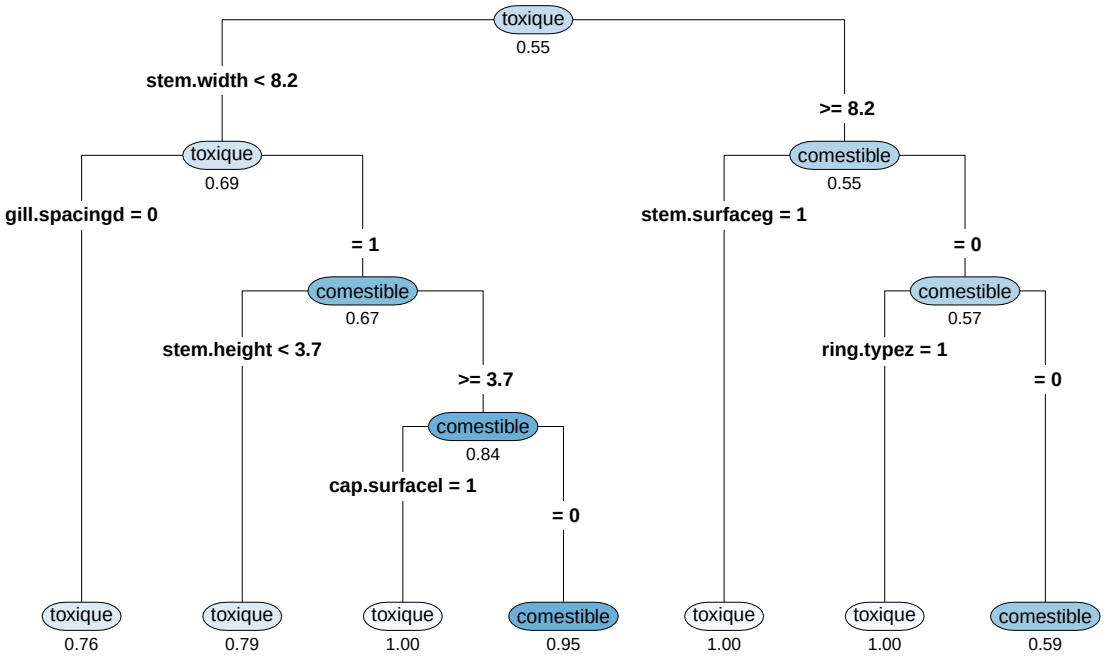


Figure 14: Arbre simplifié proposé par CART pour la classification des champignons

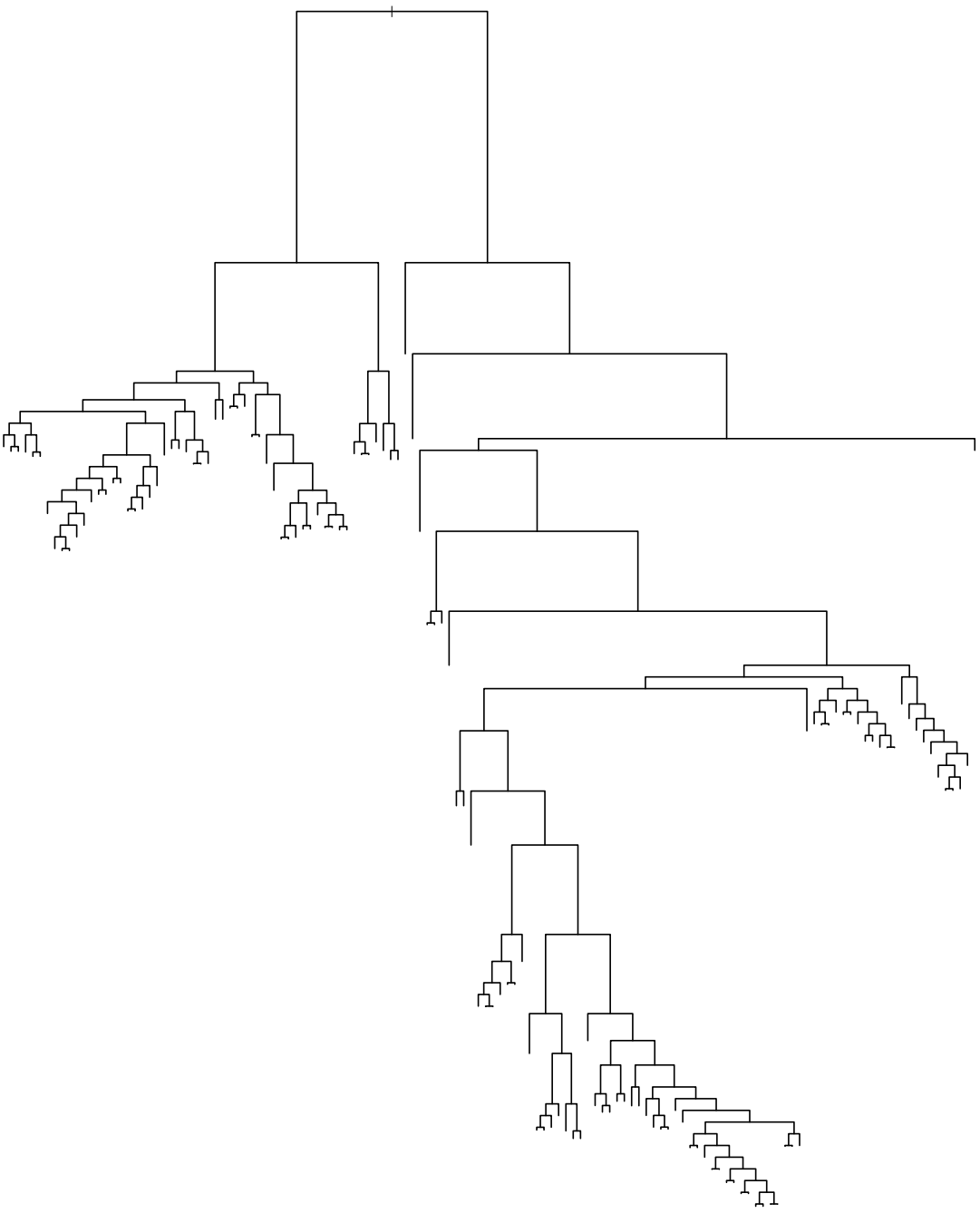


Figure 15: Structure arborescente complète proposée par CART pour la classification des champignons

4.4.3 Forêts aléatoires

Les arbres décisionnels conventionnels ont montré, au cours de leur utilisation dans le traitement des données, quelques limites récurrentes :[45]

- Tendance à l'instabilité des structures arborescentes, même avec des perturbations mineures : une petite modification du lot d'entraînement peut entraîner des modifications majeures de l'arborescence d'un arbre décisionnel,
- Apparition, grâce notamment aux avancées de la génomique, de problèmes dits de type *grand p, petit n*, avec un grand nombre de prédicteurs p pour un petit nombre d'observations n,[g] qui mettent en échec la vision classique de parcimonie d'un modèle unique pour un problème unique,
- Performances limitées face à des combinaisons linéaires de facteurs,
- Impossibilité usuelle d'inférence théorique, liée à la nature adaptative de la construction des arbres décisionnels.

La génération de forêts s'est imposée comme une solution élégante à ces limitations. Une forêt est un ensemble d'arbres qui, individuellement, sont suboptimaux, mais dont la combinaison en un comité améliore considérablement les performances.[47] L'exploitation de nombreux arbres offre la possibilité d'utiliser plus d'informations, et d'avoir ainsi une meilleure compréhension des données : des arbres différents peuvent proposer des cheminements alternatifs vers une solution.

Une forêt aléatoire est un type particulier de forêt, dont la définition canonique correspond, d'après Breiman,[48] à une collection $\left(\widehat{h}(.,\Theta_1),\dots,\widehat{h}(.,\Theta_q)\right)$ d'arbres de prédiction avec $\Theta_1,\dots,\Theta_q$ des variables indépendantes, identiquement distribuées, aléatoirement sélectionnées dans le lot de données $\mathcal{L}_n$ comptant n observations et q prédicteurs.

L'agrégation des données s'obtient par un vote majoritaire :

$$\widehat{h}_{RF}(x) = \underset{1\leq c\leq C}{\operatorname{argmax}} \sum_{\ell=1}^{q} \mathbf{1}_{\widehat{h}(x,\Theta_\ell)=c}$$

Le type le plus commun de forêt aléatoire est la forêt aléatoire à entrées aléatoires (RF-RI : *Random Forests Random Inputs*).\footnote{Les forêts RF-RI sont d'ailleurs le type de forêt aléatoire auquel il est souvent tacitement référence lorsque le terme *forêt aléatoire* est utilisé dans la littérature.[48] Le terme *entrées aléatoires* est ici à interpréter comme *variables d'entrées aléatoires* : le principe de la forêt RF-RI est de construire une forêt aléatoire avec des prédicteurs arborescents dont les variables d'entrées sont aléatoires, chacun construit à partir d'un échantillon bootstrapé.[h]

[g] En l'espèce, les problèmes posés par la génomique imposent souvent d'analyser des dizaines de milliers de gènes (p), sur quelques dizaines ou centaines de patients (n).

[h] L'échantillonnage bootstrap correspond à la construction d'un échantillon par prélèvement d'individus *avec* remplacement.

En pratique, l'algorithme de génération d'une forêt aléatoire RF-RI, pour un lot de données structuré autour de n observations et p prédicteurs, fonctionne de la façon suivante, également résumée par la figure 16 :[48]

1. Prélever un échantillon bootstrap de taille n,
2. Appliquer un partitionnement récursif sur cet échantillon bootstrap. A chaque nœud, sélectionner aléatoirement q prédicteurs parmi les p (avec $q \ll p$),
3. Poursuivre le partitionnement récursif jusqu'à la création d'un arbre[i] : $\widehat{h}(., \Theta_\ell), 1 \leq \ell \leq q$,
4. Répéter les étapes précédentes jusqu'à l'obtention d'une forêt, puis agréger les résultats, étape souvent présentée en classification comme le fruit du *vote des arbres*.

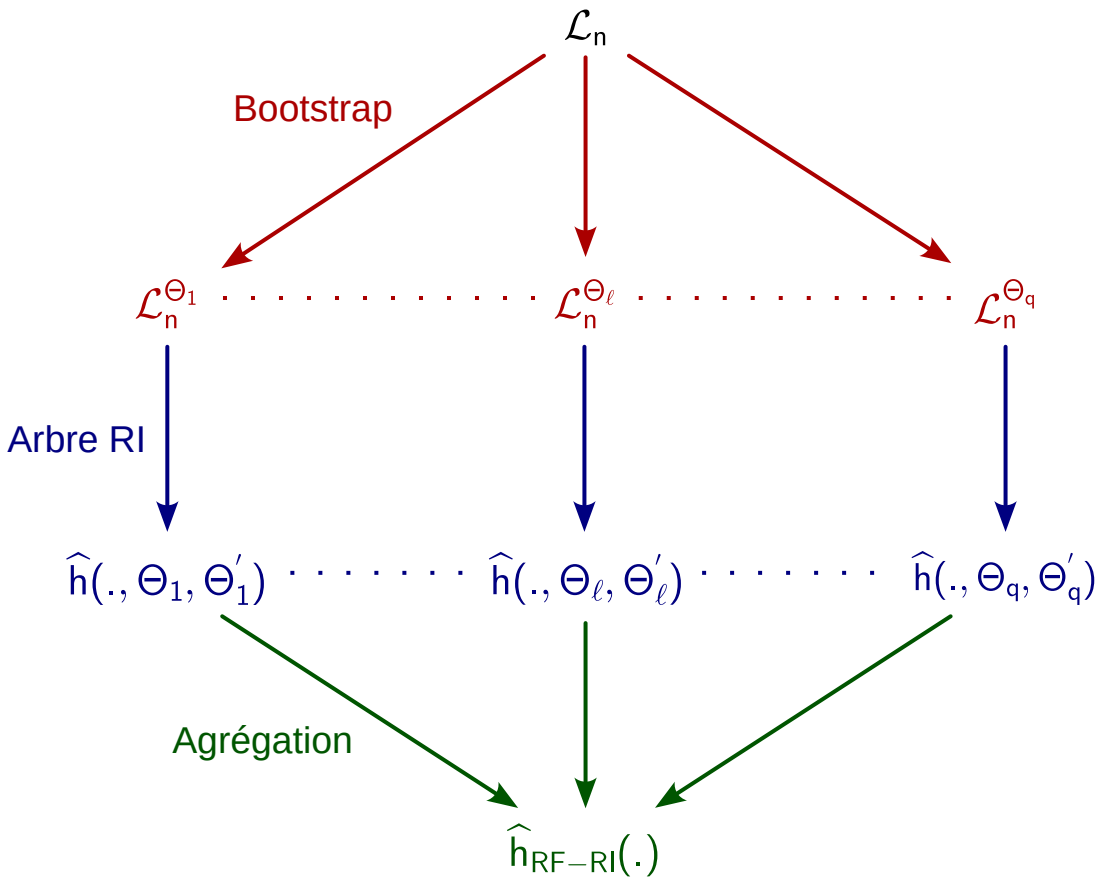


Figure 16: Algorithme de génération des forêts aléatoires de type RF-RI

Bien que les algorithmes de type forêts aléatoires soient des outils particulièrement performants, ils ont toutefois quelques limites, la principale étant la relative opacité du modèle obtenu, assimilable à une véritable "boîte noire", et donc inadapté en tant qu'outil descriptif permettant d'acquérir une meilleure compréhension des données.

[i] cf. section 4.4.2, page 37

4.5 Optimisation par plans d'expérience

Certains modèles nécessiteront une optimisation de leurs hyperparamètres, afin d'obtenir des performances maximales. Cette optimisation relève du domaine des plans d'expérience (*DOE : Design Of Experiments*). De nombreux plans et stratégies sont envisageables, le choix dépendant en partie des caractéristiques du processus à optimiser.

En effet, l'optimisation des paramètres d'un modèle informatique présente quelques particularités notables ayant un impact sur l'utilisation des plans d'expérience :

- La réalisation d'une expérience supplémentaire a un coût faible,
- Plusieurs métriques relatives aux performances peuvent coexister,[j]
- La fonction de réponse peut s'avérer relativement complexe.

Ces particularités imposent d'explorer de manière méthodique la totalité de l'espace expérimental. Il existe une multitude de méthodes permettant de générer des plans expérimentaux dits SFD (*Space Filling Design*), afin d'optimiser l'occupation de l'espace expérimental. La méthode retenue pour cette étude sera celle des hypercubes latins, en raison de son utilisation répandue[49] et de sa simplicité conceptuelle.

La méthode des hypercubes latins est une extension du principe des carrés latins. Un carré latin est une grille $n \times n$, remplie de n éléments distincts arrangés de sorte que chaque ligne et chaque colonne ne contienne qu'un seul exemplaire de chacun des n éléments. Dans le domaine des plans d'expériences, l'application des carrés latins revient à diviser un domaine expérimental bidimensionnel en une grille $n \times n$, et à placer une expérience et une seule sur chaque ligne et chaque colonne.

L'application du concept de carré latin dans un domaine expérimental à trois dimensions aboutit au cube latin. La généralisation dans un espace n-dimensionnel mène au concept d'hypercube latin.

De nombreux plans expérimentaux basés sur les hypercubes latins peuvent être générés. Nous pouvons citer principalement trois types d'hypercubes latins :

- Aléatoires,
- Optimisés, afin d'améliorer l'occupation spatiale,
- Orthogonaux, visant à minimiser la corrélation entre estimateurs des effets principaux.

Dans le cadre de cette étude, nous utiliserons des hypercubes latins quasi-orthogonaux, dont les propriétés nous permettront de modéliser de façon plus précise les performances de nos modèles en fonction de leurs paramètres de configuration (*hyperparamètres*).

[j] cf. section 4.6, page 44

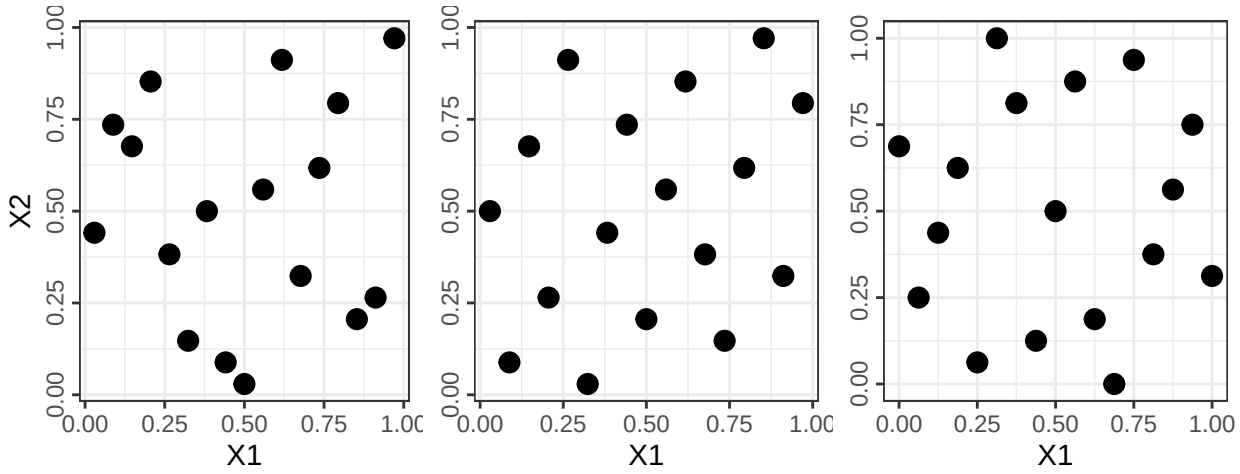


Figure 17: Carré latin aléatoire (à gauche), carré latin avec optimisation évolutive ESE maximin (au milieu), carré latin quasi-orthogonal (à droite)

Le but des plans expérimentaux de cette étude ne sera pas l'obtention d'une prédiction exacte de la valeur de la réponse, mais plus modestement la recherche des facteurs permettant d'obtenir cette réponse optimale. A cet effet, la modélisation de la performance pourra s'effectuer à l'aide d'un modèle quadratique avec interactions, de formule générale :

$$Y = \beta_0 + \sum_{i=1}^{k} \beta_i . X_i + \sum_{i<j}^{k} \sum_{j>1}^{k} \beta_{ij} . X_i . X_j + \sum_{i=1}^{k} + \beta_{ii} . X_i^2 + \varepsilon$$

Avec β_n les coefficients des effets principaux et X_n leurs facteurs réduits.

4.6 Évaluation des performances des modèles

L'optimisation des modèles ainsi que la comparaison de leurs performances relatives implique nécessairement de définir quel sera le critère vis-à-vis duquel cette performance sera évaluée.

De nombreux critères sont utilisables, en fonction du cahier des charges défini pour la résolution du problème, mais également du type de tâche effectuée : régression, classification binaire, classification multiclasse.

Dans une tâche de classification binaire, les critères usuels sont la spécificité, la sensibilité, et l'aire sous la courbe de fonction d'efficacité du récepteur (*AUC ROC*, parfois abrégé en *ROC*). Il conviendra bien évidemment, avant d'utiliser des indicateurs tels que la spécificité et la sensibilité, de définir la notion de test positif et test négatif.

D'autres indicateurs d'intérêt existent, nous retiendrons ici l'index J de Youden[50] pondéré, qui permet de d'ajuster l'importance relative accordée à la spécificité et à la sensibilité, au sein d'un index synthétique.[51] Cet indicateur présente un intérêt particulier lorsqu'il apparaît souhaitable de tenir compte de la différence d'impact entre un faux positif et un faux négatif, sans pour autant autoriser des sensibilités ou spécificités trop faibles.

En l'espèce, l'index J de Youden pondéré nous permet d'élaborer un indice synthétique tenant compte du fait qu'il est plus grave de classer à tort comme comestible un champignon toxique que d'écarter à tort un champignon parfaitement comestible – sans pour autant autoriser le modèle à écarter un nombre inconsidéré de champignons comestibles.

L'index J de Youden pondéré est donné par :[51]

$$J_w = 2.\left(w.Sen + (1-w).Spe\right) - 1 \qquad avec \quad w \in [0;1]$$

Dans la classification binaire de cette étude, problème qui revient classiquement en mycologie à classer les espèces en fonction de leur comestibilité, la valeur positive sera ici arbitrairement attribuée à la valeur "champignon toxique". Nous cherchons donc à maximiser la sensibilité de la détection, afin d'écarter les espèces toxiques, la spécificité – c'est-à-dire la capacité à ne pas écarter trop d'espèces comestibles – apparaissant alors comme un critère relativement secondaire. En établissant arbitrairement un index J de Youden pondéré accordant 10 fois plus d'importance à la sensibilité qu'à la spécificité, nous pouvons établir $w = 10/11$.

Le problème de classification binaire étant relativement simple (*comestible* ou *non-comestible*), nous fixerons arbitrairement le critère de performance minimum à atteindre à $J_w \geq 0.999$, soit :

$$\begin{cases} Sen_{max} = 1 \Leftrightarrow Spe_{min} = 0.9945 \\ Spe_{max} = 1 \Leftrightarrow Sen_{min} = 0.99945 \end{cases}$$

Dans les tâches de classification multiclasse, d'autres indicateurs d'intérêt pourront être utilisés, tels que le kappa de Cohen, l'indice de Rand (*accuracy*, ou précision), mais aussi la sensibilité et la spécificité moyennes.

Nous retiendrons dans les classifications multiclasses effectuées au cours de notre étude le kappa de Cohen,[52] calculé à partir de la matrice de confusion (illustrée en figure 18), et donné par :

$$\kappa = \frac{\pi_0 - \pi_e}{1 - \pi_e}$$

Avec π_0 la probabilité d'accord entre notre modèle et la classe réelle du champignon, et π_e la probabilité d'un même accord résultant du pur hasard.

Landis et Koch ont élaboré une échelle de validité du kappa de Cohen, avec un accord qualifié de *quasi-parfait* pour $\kappa > 0.80$.[53] Nous considérerons donc que ce critère sera le minimum requis pour qu'un modèle de classifieur multiclasse élaboré au cours de cette étude puisse être considéré comme ayant des performances acceptables.

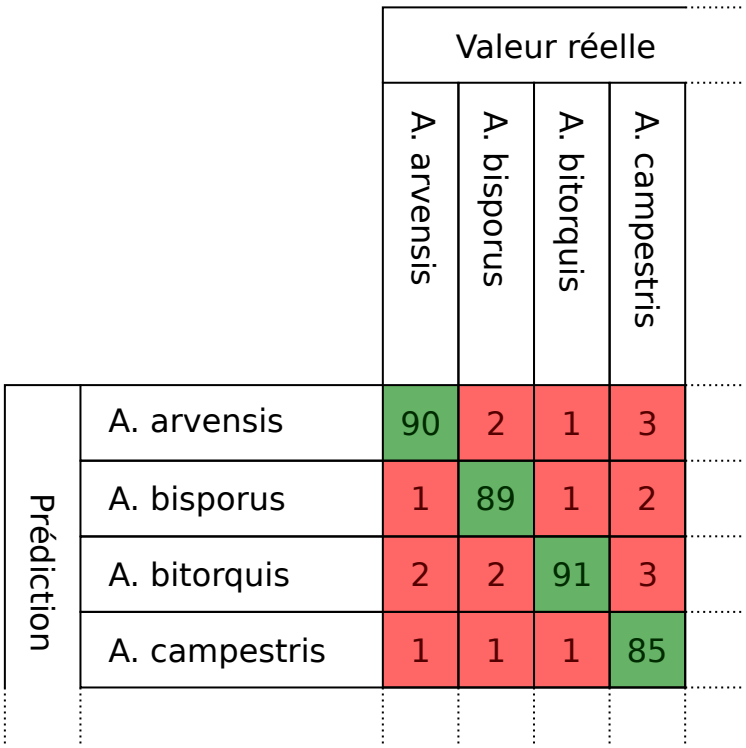


Figure 18: Extrait d'une matrice de confusion, pour une classification multiclasse

L'interprétation du kappa pouvant parfois être assez contre-intuitive, cette étude la complétera parfois par l'indice de Rand R, métrique moins robuste en présence de données non-équilibrées[k], mais présentant l'avantage d'être de compréhension plus aisée, car représentant le pourcentage de prédictions exactes.

[k] Ce cas se présente habituellement lors d'une surreprésentation de certaines classes dans les jeux de données.

5 Apprentissage machine et classification binaire

BROUILLON, le lot de données utilisé ici est le Secondary Mushroom Dataset de D.Wagner.

5.1 Analyse exploratoire des données (EDA)

Partie permettant de présenter globalement le lot de données synthétiques créé dans cette étude (cf. section 3.2.1) avant utilisation des IA.

La génération du lot de données fonctionne, la génération des graphiques aussi, mais le texte sera à retravailler en fonction du lot de données synthétiques obtenu. . .

Le lot de données d'origine contient 61069 spécimens de champignons, caractérisés par 21 propriétés morphologiques ou environnementales. La structure de ce lot de données est résumée dans le tableau 3.

Ce lot de données original a été découpé en un jeu d'apprentissage/optimisation et un jeu de données d'évaluation, avec un rapport 16:1, conformément aux principes mentionnés dans nos développements précédents.[l]

Toutes les distributions des variables du lot d'entraînement ont ensuite été tracées par histogrammes pour les variables numériques, et diagrammes en barres pour les variables alphabétiques et catégorielles.

Les diagrammes en barres n'ont rien illustré de particulièrement remarquable et n'ont pas été inclus dans le rapport. Toutefois, les distributions dimensionnelles sont plus intéressantes : à première vue, elles semblent suivre une courbe en cloche (figure 19), avec une longue queue à droite. Une transformation logarithmique (figure 20) montre plus nettement la forme de cette queue.

La distribution du diamètre du chapeau D_C a l'apparence d'une courbe en cloche, avec une longue queue à droite, mais est en réalité bimodale, avec un mode principal à 5 cm, et un mode secondaire beaucoup plus petit pour $D_C \approx 50$ cm. Cette taille exceptionnelle est attribuable à des variétés telles que *Polyporus squamosus*.[2]

La distribution de la longueur de stipe L_S a également une forme de courbe en cloche avec une longue queue à droite, un mode principal à 5 cm et un mode secondaire à 0 cm. Cette valeur peut également sembler surprenante, mais certains champignons du lot n'ont pas de pied, ce qui explique cette valeur.

[l] cf. section 4.2, page 31.

Table 3: Structure du lot de données initial

	Type	Niveaux
class	factor	2
cap.diameter	numeric	2571
cap.shape	factor	7
cap.surface	factor	12
cap.color	factor	12
does.bruise.or.bleed	factor	2
gill.attachment	factor	8
gill.spacing	factor	4
gill.color	factor	12
stem.height	numeric	2226
stem.width	numeric	4630
stem.root	factor	6
stem.surface	factor	9
stem.color	factor	13
veil.type	factor	2
veil.color	factor	7
has.ring	factor	2
ring.type	factor	9
spore.print.color	factor	8
habitat	factor	8
season	factor	4

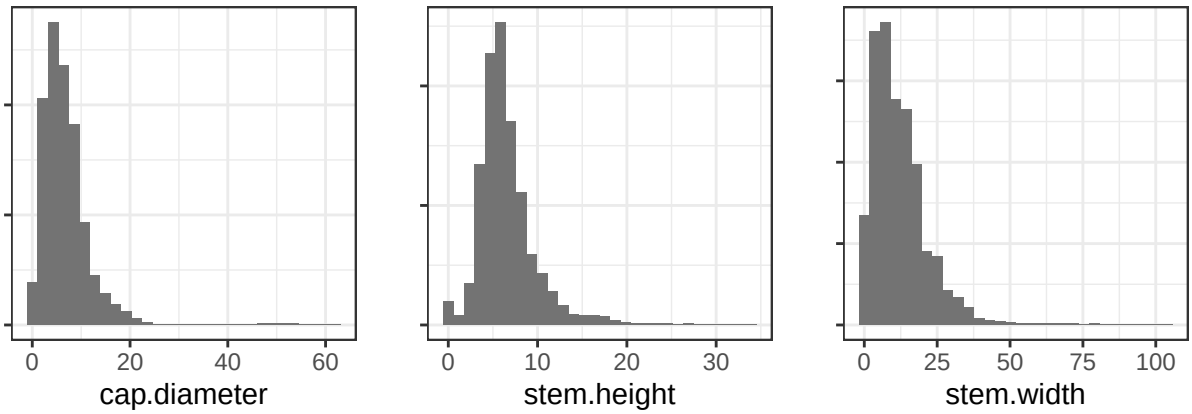


Figure 19: Distribution des diamètres de chapeau, longueur de stipe, diamètre de stipe

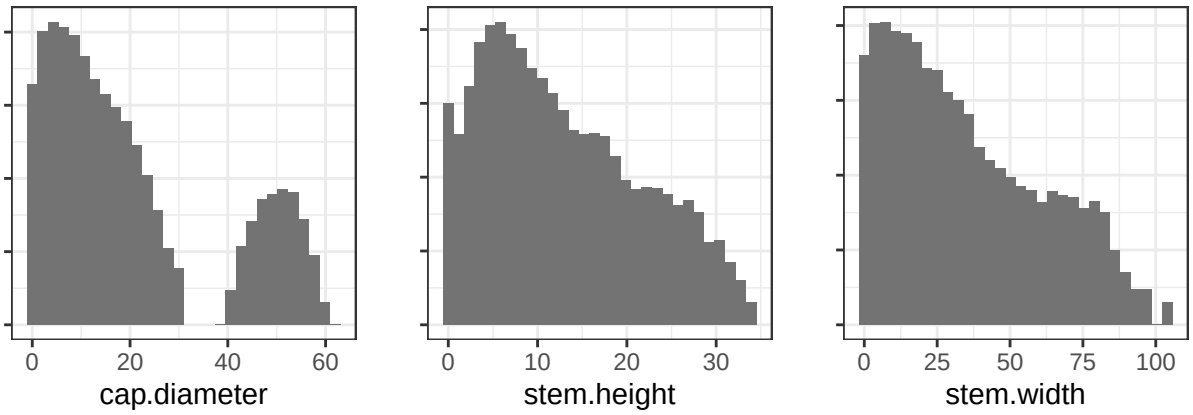


Figure 20: Distribution des diamètres de chapeau, longueur de stipe, diamètre de stipe (échelle logarithmique en ordonnée)

La distribution du diamètre de stipe D_S a aussi l'apparence d'une courbe en cloche avec une longueur queue à droite, et un pic à $D_S \approx$ 10-15 mm. Dans toutes ces distributions, la longue queue à droite peut probablement s'expliquer par l'utilisation, dans le *Secondary Mushroom Dataset*, d'une distribution normale pour chaque variété, associée à l'impossibilité d'avoir des valeur dimensionnelles négatives.

L'étude des distributions dans un espace bidimensionnel (figure 21, p. 50) permet déjà de dégager des associations de caractéristiques d'intérêt, ici représentées sous formes de zones unicolores où un type de champignons (comestible ou non) dominera, ce qui permet déjà d'élaborer des stratégies de classification basiques.[m]

Ainsi, il est déjà possible de constater, d'après le nuage de points en haut à droite de la figure 21 que dans le lot de données, *tous* les champignons pour lesquels *cap.diameter* ≥ 40 seront comestibles, et de même pour *stem.height* ≥ 20. Toutefois, ainsi que l'illustre le graphique de densité situé dans le coin opposé, cette possible séparation ne concernera qu'une minorité de champignons, l'immense majorité des spécimens ayant des caractéristiques *stem.heigh* ≤ 10 et *cap.diameter* ≤ 15, où les courbes de densité se superposent, laissant entendre que cette association de critères ne sera que peu d'intérêt pour classer un grand nombre de champignons.

La distribution du diamètre de stipe D_S a aussi l'apparence d'une courbe en cloche avec une longueur queue à droite, et un pic à $D_S \approx$ 10-15 mm. Dans toutes ces distributions, la longue queue à droite peut probablement s'expliquer par l'utilisation, dans le *Secondary Mushroom Dataset*, d'une distribution normale pour chaque variété, associée à l'impossibilité d'avoir des valeur dimensionnelles négatives.

De même, les graphiques de distribution des variables *cap.diameter* et *stem.height* en fonction des variables *veil.type* et *has.ring* montrent des distributions multimodales, qui mettront probablement

[m] Ce type de classifieur sera développé ultérieurement, en section 5.2.2, page 52

en difficulté des modèles d'analyse discriminante linéaire, basées sur l'hypothèse de variables distribuées normalement – ou a minima dotées de distributions monomodales.[n]

Cette brève analyse exploratoire des données illustre donc l'intérêt de l'EDA, qui constitue un outil puissant permettant de mieux appréhender un lot de données, voire d'orienter le choix des stratégies d'apprentissage machine qui pourraient être exploitées à fins de classification.

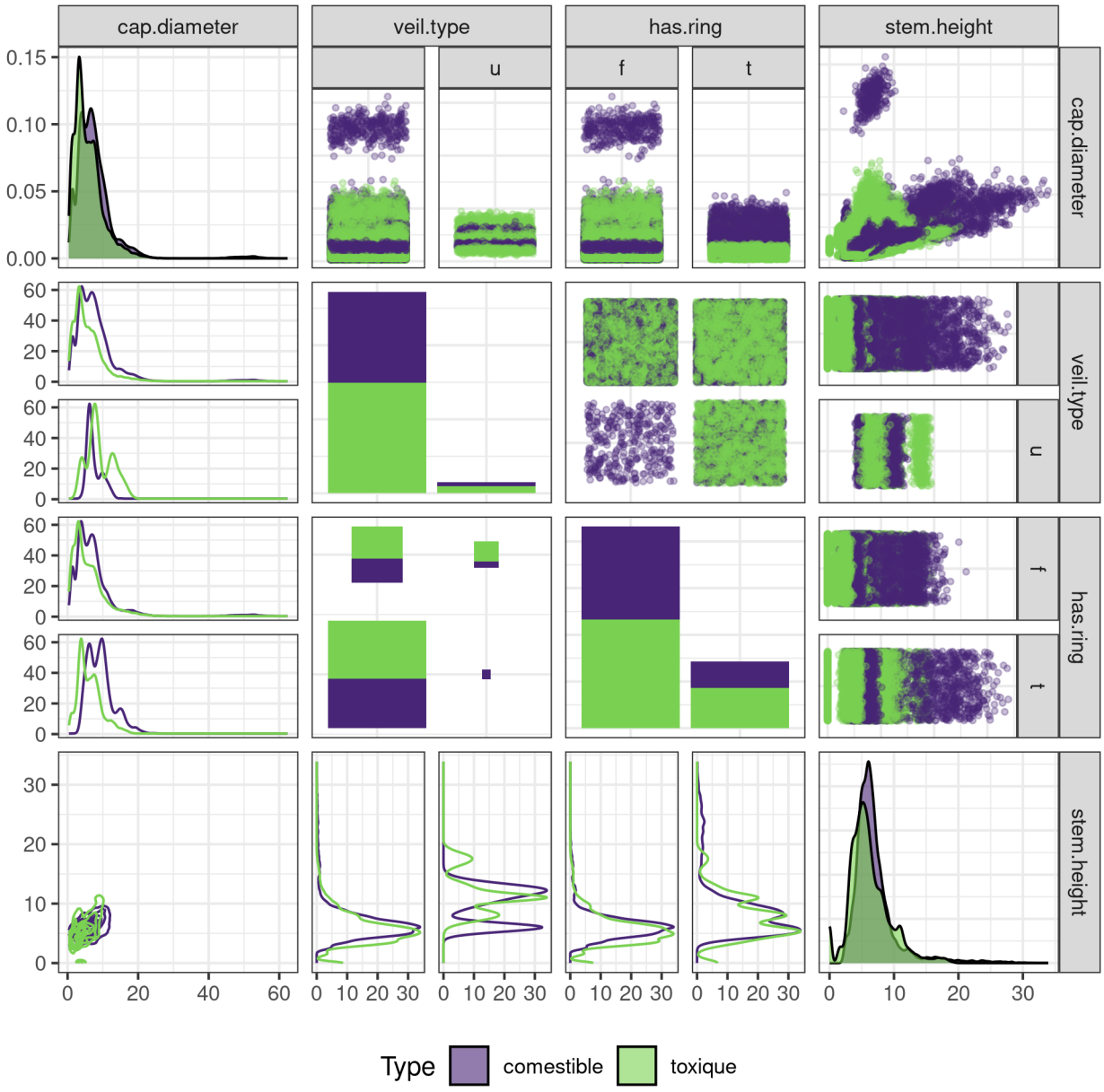


Figure 21: Distributions bi-critères pour 4 critères d'intérêt

[n]cf. section 4.4.1, page 36

5.2 Optimisation et sélection des modèles

Il existe une grande variété de modèles exploitables pour bâtir un système d'apprentissage machine. Cette section expliquera la stratégie utilisée pour l'évaluation de certains de ces modèles, ainsi que pour l'exploration de l'espace de leurs hyperparamètres à fins d'optimisation et la mesure de leurs performances.

Les modèles sélectionnés pour cette étude sont de types variés :

- Analyse naïve
- Analyse discriminante linéaire (LDA) : *Linear Discriminant Analysis* (lda2), *Penalized Discriminant Analysis* (pda)
- Modèle arborescent : *Classification And Regression Tree* (CART) (rpart, rpartCost), C5.0 tree
- Forêt aléatoire : *Random Ferns* (rferns), *Random Forest* (ranger, Rborist)

5.2.1 Stratégie d'optimisation

Les algorithmes d'apprentissage machine développés au cours de cette étude mettent en œuvre les méthodes présentées dans les sections précédentes afin d'effectuer automatiquement les tâches suivantes :

1. Découpage du lot de données en un jeu d'entraînement/optimisation et en un jeu de validation, avec adaptation des rapports de taille en fonction du volume de données du lot initial,[o]
2. Apprentissage sur le jeu d'entraînement, exploitant une validation croisée à k blocs, avec adaptation du nombre de blocs à la taille du lot de données,[p]
3. Exploration de l'ensemble de l'espace expérimental des hyperparamètres du modèle, via la méthode des hypercubes latins quasi-orthogonaux,\footnote{cf. section 4.5, page 43
4. Mesure des performances en exploitant une métrique adaptée,[q]
5. Modélisation des performances en fonction des hyperparamètres, via un modèle quadratique avec interactions,[q]
6. Sélection des hyperparamètres permettant d'optimiser les performances du modèle,
7. Mesure des performances de chaque modèle avec les hyperparamètres optimaux,
8. Sélection des modèles les plus performants pour prédiction et mesure finale des performances contre le lot d'évaluation,
9. Génération, sauvegarde et insertion automatique de tous les graphiques et données numériques dans le texte de la présente étude.

[o] cf. section 4.2, page 31
[p] cf. section 4.3, page 32
[q] cf. section 4.6, page 44

5.2.2 Modèle naïf

Le premier classifieur présenté dans cette étude est un classifieur naïf, dont l'algorithme est extrêmement simple :

1. Considérer par défaut tous les champignons comme toxiques,
2. Tester chaque combinaison de n variables, à la recherche des valeurs qualitatives et quantitatives pour lesquelles tous les champignons sont comestibles.

Trois classifieurs sont ainsi proposés : un classifieur "stupide", ne tenant compte d'aucun critère ($n = 0$) et considérant tous les champignons comme toxiques, un classifieur monovariable ($n = 1$), et un classifieur exploitant des combinaisons de 2 variables ($n = 2$).

Les performances respectives de ces différents modèles sont synthétisées dans le tableau 4.

Table 4: Performances de différents classifieurs naïfs

	n	Sens	Spec	Jw	Kappa	Temps (s)
Stupide	0	1.000	0.000	0.818	0.000	0.0
MonoCritere	1	1.000	0.099	0.836	0.108	4.3
BiCritere	2	0.996	0.834	0.963	0.844	759.5

Les performances de ces modèles paraissent relativement bonnes, mais sont à relativiser en raison de la définition de notre indice de Youden qui accorde une très grande importance à la sensibilité. Ainsi, notre classifieur dit "stupide" – excessivement pusillanime et d'un intérêt pratiquement nul car rejetant tous les champignons – semble montrer une performance honorable d'après ce critère, avec $J_w = 0.818$, alors que, par construction, $\kappa = 0$.

Les performances du modèle monovariable ne sont guère plus élevées, ce qui démontre l'inefficacité des adages populaires prétendant garantir la comestibilité de tous les champignons d'une couleur donnée.[r]

En revanche, les performances du modèle bivariable sont honorables, avec une amélioration sensible de la spécificité ($Spec \approx 0.834$), donc de la capacité à réellement distinguer des champignons comestibles. Les performances de ce modèle ($J_w = 0.963$) ne répondent toutefois pas au critère de performance défini précédemment ($J_w \geq 0.999$)[s]. Les performances calculatoires sont médiocres ($t = 759.5$ s), et probablement à mettre en relation avec un code peu optimisé.

[r] Même si le modèle venait à montrer le contraire, il convient de rappeler que ce lot de données se limite aux espèces les plus courantes, d'une fonge limitée dans l'espace et dans le temps.

[s] cf. section 4.6, page 44

5.2.3 Modèles d'analyse discriminante

Les modèles d'analyse discriminante choisis pour cette étude sont lda2 (LDA : *Linear Discriminant Analysis*) et pda (*Penalized Discriminant Analysis*). Le modèle lda2 dispose d'un seul hyperparamètre (*dimen*, nombre de fonctions discriminantes). Le modèle pda a également un unique hyperparamètre (*lambda*, pénalité de réduction des coefficients).

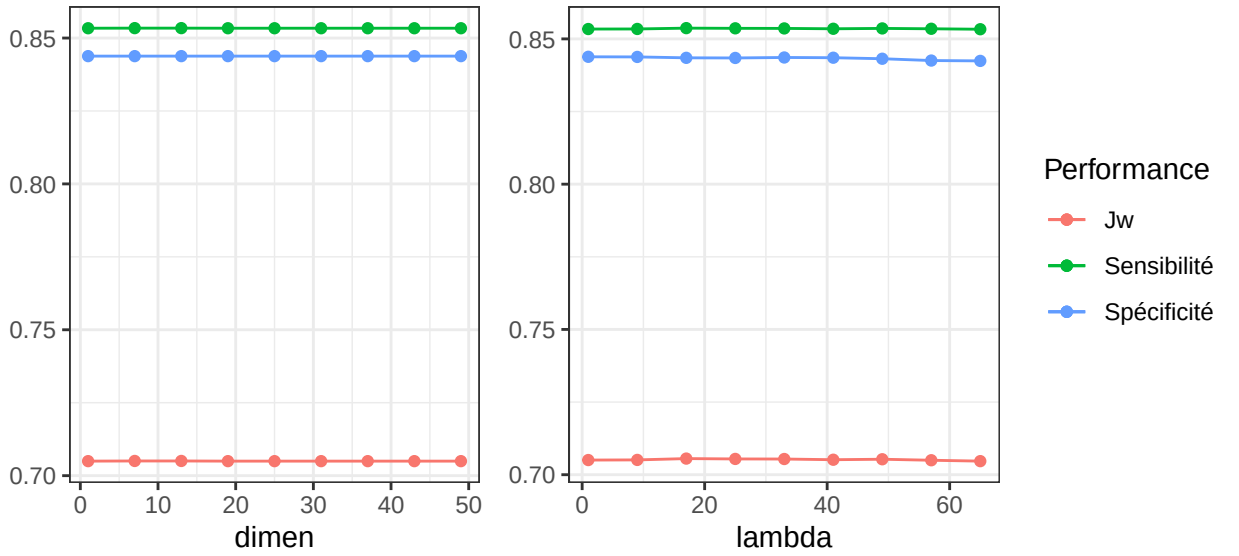


Figure 22: Performances des modèles lda2 (à gauche) et pda (à droite), dans une tâche de classification binaire

Comme l'illustre la figure 22, les performances des modèles PDA et LDA sont très proches, et relativement constantes sur la totalité de l'espace expérimental de leurs hyperparamètres.

Le paramètre *dimen* du modèle lda2 ne semble en effet pas avoir d'effet significatif sur ses performances, avec un index J de Youden pondéré relativement constant ($J_{w_{moy}} = 0.705$).

De même, le paramètre *lambda* du modèle pda n'impacte ses performances que de matière très marginale, avec des lambdas faibles donnant une légère amélioration des résultats ($J_{w_{max}} = 0.706$).

Toutefois, les performances de ces deux modèles restent malheureusement insuffisantes pour notre étude, aussi bien en sensibilité qu'en spécificité :

$$\begin{cases} Sen \approx 0.853 \\ Spe \approx 0.848 \end{cases}$$

Ces performances médiocres s'expliquent par le fonctionnement même des modèles d'analyse discriminante qui, s'ils peuvent analyser des données qualitatives à fins de classifications, ne peuvent le faire que si une quantification sous-jacente est possible, par exemple :

- Données binaires ou booléennes,
- Données qualitatives ordinales.

L'inclusion de ces modèles, présentant ici des performances très modestes, a un intérêt essentiellement didactique, permettant de souligner l'intérêt d'une connaissance élémentaire des fondamentaux mathématiques et algorithmiques des modèles d'apprentissage machine mis en œuvre, pour en connaître les limites ou évaluer les besoins de nettoyage préalable des données avant déploiement de l'apprentissage machine, afin d'éviter de confronter certains modèles face à des problèmes de classification pour lesquels ils n'ont pas été conçus.

5.2.4 Modèles d'arbres de décision

Les modèles basés sur des arbres de décision ont un intérêt tout particulier pour cette étude, pour deux raisons majeures :

- La logique en arbre de décision est habituellement usitée pour la classification manuelle des champignons,
- Les arbres de décision obtenus peuvent être tracés, et facilement interprétés par l'humain.

Les premiers modèles présentés dans le cadre de notre étude sont deux modèles CART (*Classification And Regression Tree*). Le modèle CART le plus simple proposé dans notre étude (rpart) ne dispose que d'un seul hyperparamètre : *cp* (complexité).

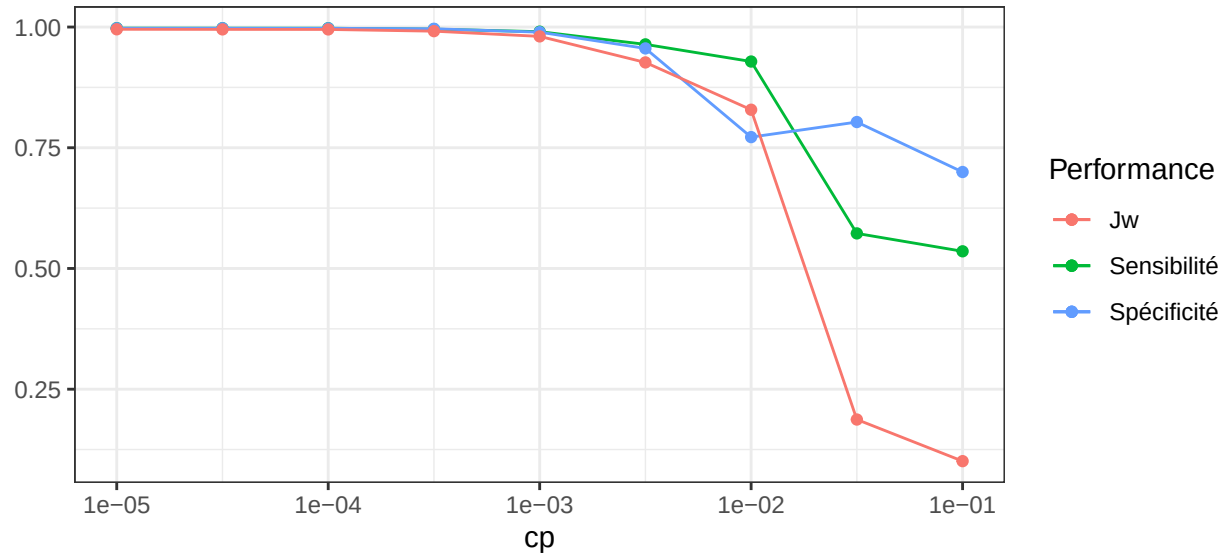


Figure 23: Performances du modèle rpart en classification binaire, en fonction du paramètre de complexité (cp)

Le modèle CART le plus simple n'atteint jamais les performances requises, le critère étant $Jw \geq 0.999$. Toutefois, ce modèle s'en approche, et donne de bons résultats globaux, avec :

$$\begin{cases} J_{w_{max}} = 0.9954 \\ Spe_{Jw_{max}} = 0.9974 \\ Sen_{Jw_{max}} = 0.9977 \end{cases}$$

Le second modèle CART utilisé dans cette étude (rpartCost) associe des hyperparamètres de complexité (cp) et de coût ($Cost$). Les graphiques de sensibilité et de spécificité en fonction des hyperparamètres (figure 24) illustrent bien, dans leur partie supérieure ($cp \geq 0.05$) la notion classique de compromis entre sensibilité et spécificité : dans cette zone, toute amélioration de la sensibilité se fera inévitablement au détriment de la spécificité, et réciproquement.

En pratique, pour $cp \geq 0.05$, notre modèle d'IA basé sur ce type d'arbre de décision se montrera soit excessivement sévère, rejetant un nombre considérable de champignons comestibles (quadrant supérieur gauche, $cost \leq 1.5$), soit au contraire excessivement laxiste, admettant un nombre important de champignons non-comestibles (quadrant supérieur droit, $cost \geq 1.5$).

C'est dans la section inférieure de ces graphiques ($cp \leq 0.025$) que le modèle montre une performance acceptable tant en sensibilité qu'en spécificité.

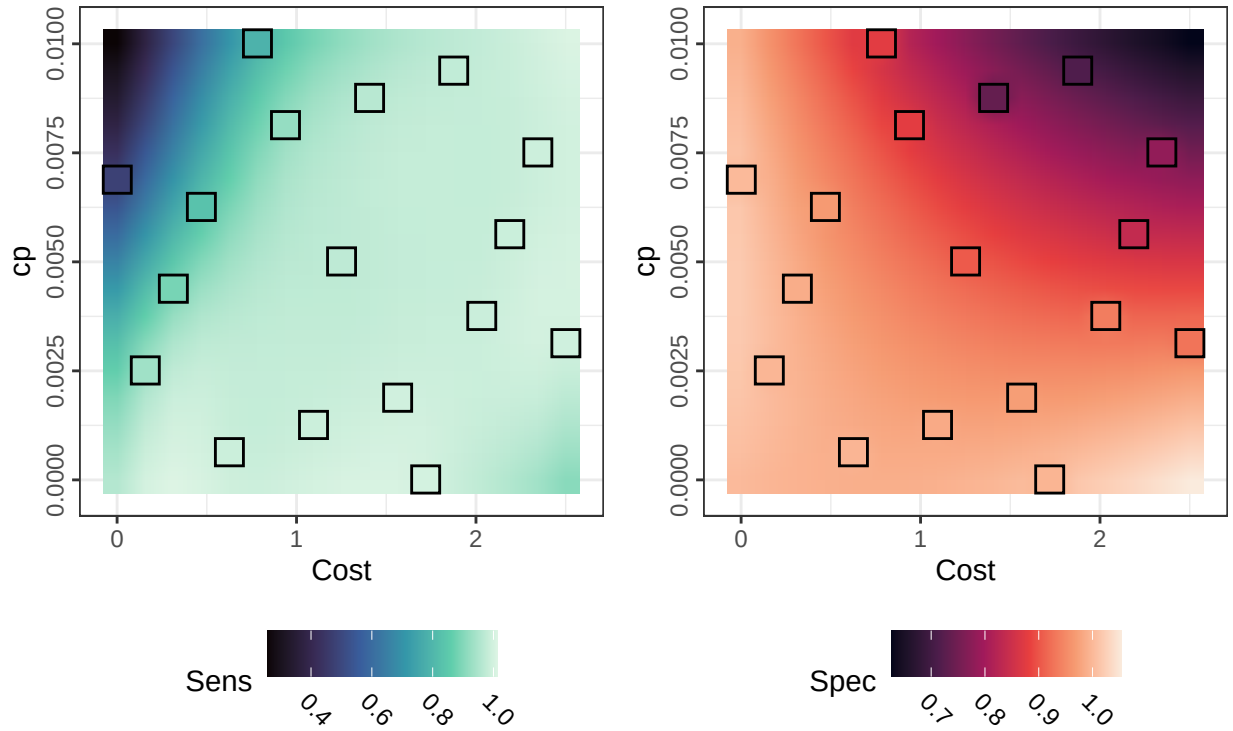


Figure 24: Sensibilité (à gauche) et spécificité (à droite) de rpartCost en classification binaire, en fonction de la complexité et du coût (interpolation quadratique, points expérimentaux encadrés en noir)

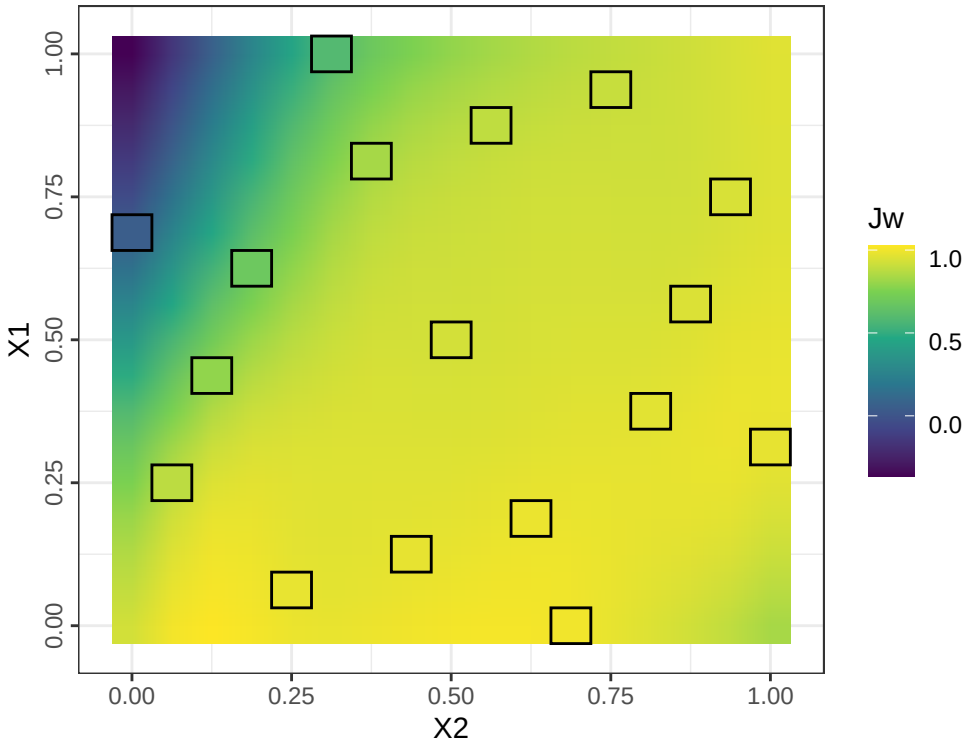


Figure 25: Performances (index J de Youden pondéré 10:1) de rpartCost en classification binaire, en fonction des hyperparamètres réduits de complexité X1 et de coût X2 (interpolation quadratique, points expérimentaux encadrés en noir)

En posant comme facteurs réduits :

- $X_1 \in [0;1]$ pour le paramètre *minNode*,
- $X_2 \in [0;1]$ pour le paramètre *predFixed*,

Nous pouvons modéliser la réponse Y (J_w) par un modèle quadratique avec interaction[t] :

$$Y = b_0 + b_1.X_1 + b_2.X_2 + b_{12}.X_1.X_2 + b_{11}.X_1^2 + b_{22}.X_2^2$$

Avec Y l'indice J de Youden pondéré, X_1 le facteur réduit dans la plage [0;1] associé à l'hyperparamètre de complexité (*cp*), X_2 le facteur réduit associé à l'hyperparamètre de coût (*Cost*) et b_n les coefficients des effets. La modélisation permet de calculer les effets suivants :

$$\begin{cases} b_0 = 0.813 \\ b_1 = -0.7723 \\ b_2 = 1.2453 \end{cases} \qquad \begin{cases} b_{12} = 1.3354 \\ b_{11} = -0.3065 \\ b_{22} = -1.2955 \end{cases}$$

Les performances maximales seront ici atteintes pour :

$$\begin{cases} X_1 = 0 & soit \quad cp = 1e-05 \\ X_2 = 0.5 & soit \quad Cost = 1.25 \end{cases}$$

[t] cf. section 4.5, page 43

Ces hyperparamètres optimaux permettent au modèle d'atteindre :

$$\begin{cases} J_{w_{max}} = 0.995 \\ Spe_{Jw_{max}} = 0.9966 \\ Sen_{Jw_{max}} = 0.9976 \end{cases}$$

Les performances du modèle rpartCost, bien qu'excellentes, ne permettent pas d'atteindre l'index J de Youden requis.

Le dernier modèle d'arbre décisionnel proposé dans notre étude est C5.0 tree (c50tree). Ce modèle ne dispose d'aucun hyperparamètre.

Les performances obtenues sont :

$$\begin{cases} J_w = 0.9978 \\ Spe = 0.9988 \\ Sen = 0.9989 \end{cases}$$

De manière assez surprenante, bien que ne disposant d'aucun hyperparamètre, ce modèle a donné d'excellents résultats sans aucune optimisation nécessaire. Toutefois, le modèle C5.0 tree n'a pas rempli l'objectif posé par le critère $J_w \geq 0.999$.

Les modèles d'arbres de classification sont particulièrement adaptés aux problèmes de classification avec variables quantitatives et surtout qualitatives, et ont pu s'illustrer dans cette étude en fournissant des résultats très acceptables ($Spe \geq 0.994$, $Sen \geq 0.998$), mais n'atteignant pas pour autant les exigences imposées par le critère de performance que nous avons défini pour les classifieurs binaires de notre étude.

5.2.5 Forêts aléatoires

Le premier modèle de forêt aléatoire évalué dans notre étude est le modèle de fougères aléatoires rFerns (*Random Ferns*). Ce modèle ne possède qu'un seul hyperparamètre, la profondeur (*depth*).

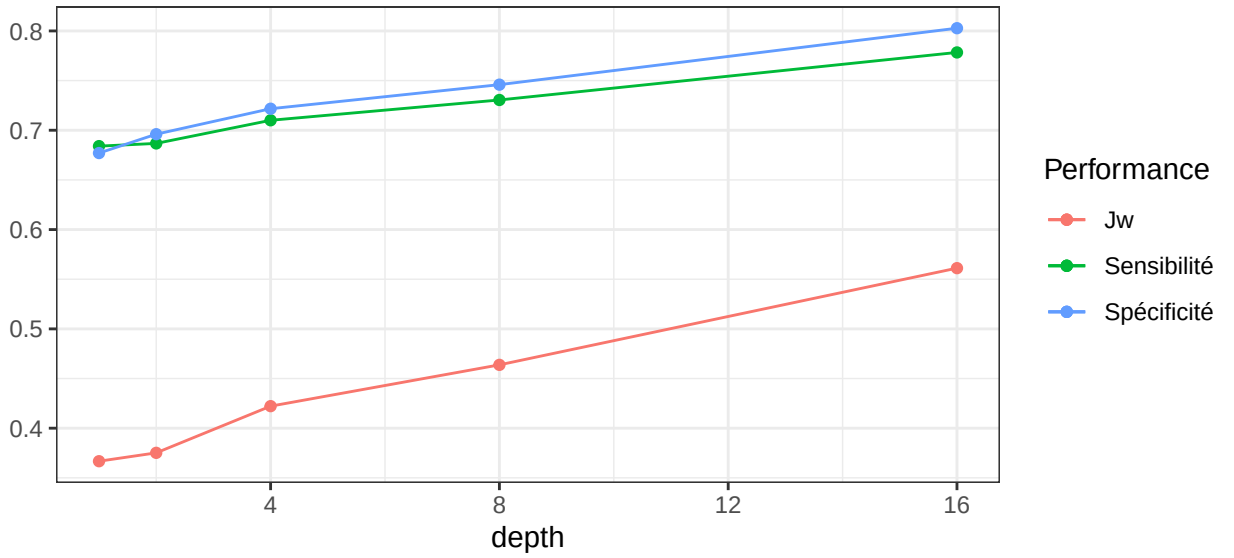


Figure 26: Performances du modèle de fougères aléatoires, en classification binaire

Quoique très efficient sur le plan calculatoire, le modèle de fougères aléatoires a fourni des résultats assez peu satisfaisants, avec :

$$\begin{cases} J_{w_{max}} = 0.561 \\ Spe_{Jw_{max}} = 0.803 \\ Sen_{Jw_{max}} = 0.778 \end{cases}$$

Le second modèle de forêt aléatoire évalué dans cette étude est Rborist. Deux hyperparamètres régissent ce modèle : le nombre de prédicteurs testés pour une scission (*predFixed*) et le nombre minimal de lignes-références distinctes avant de scinder un nœud (*minNode*).

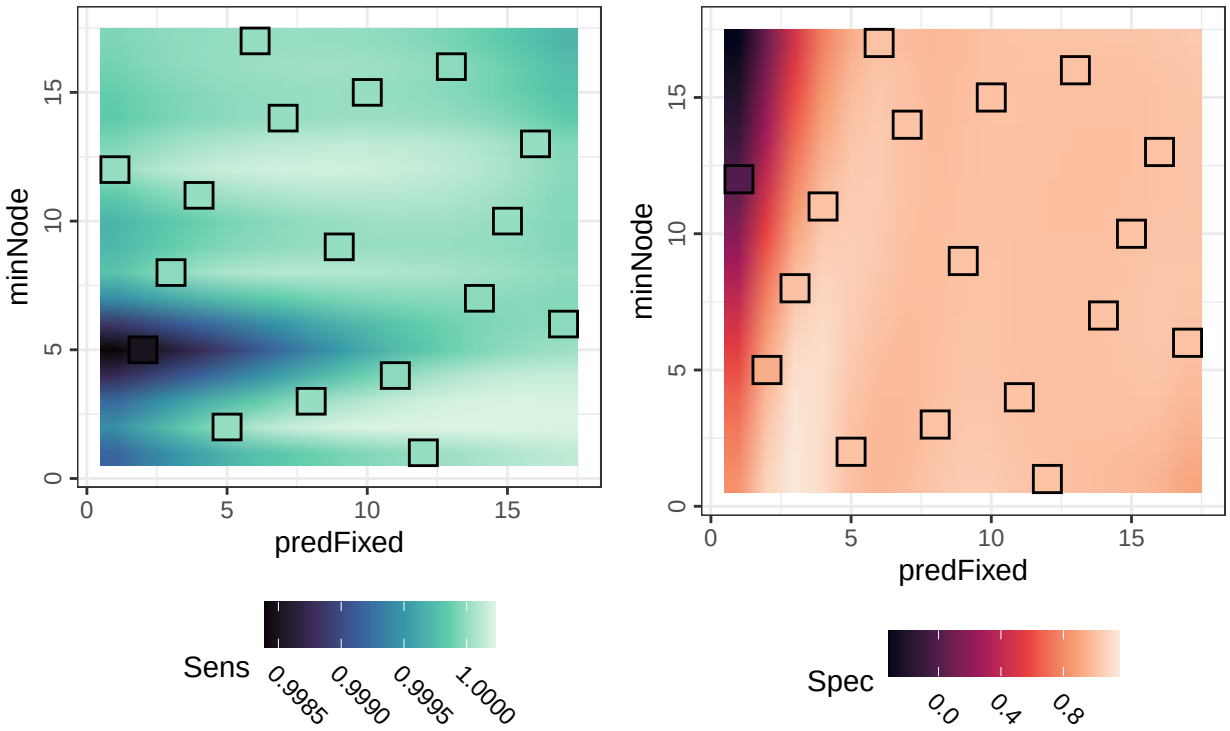


Figure 27: Sensibilité (à gauche) et spécificité (à droite) du modèle Rborist en classification binaire, en fonction de ses deux hyperparamètres (interpolation quadratique, points expérimentaux encadrés en noir)

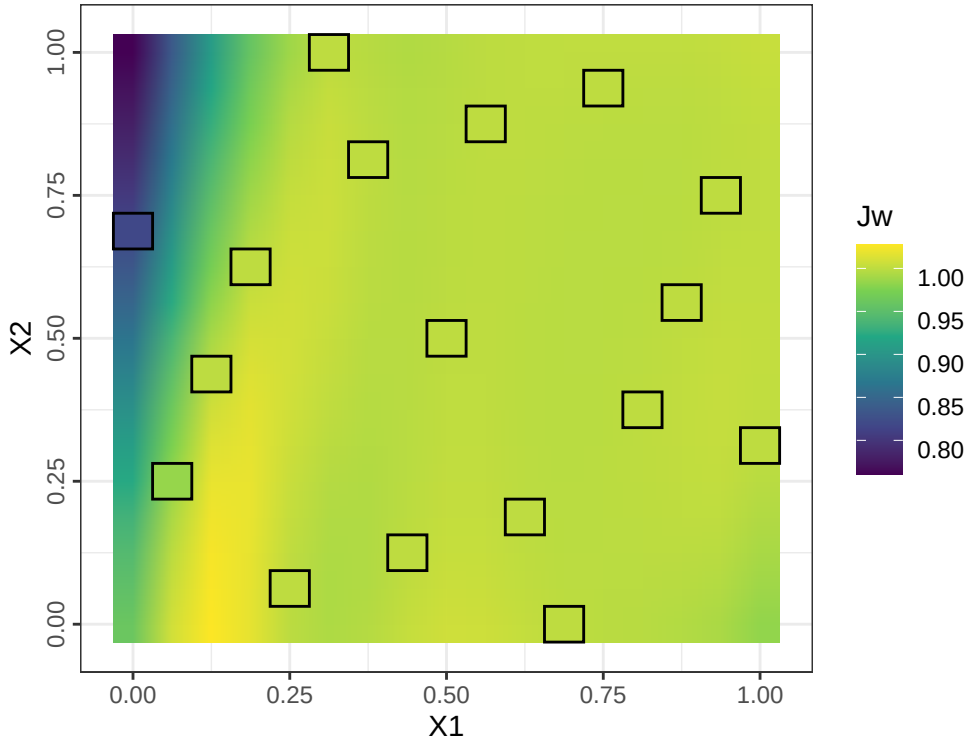


Figure 28: Performance (indice J de Youden pondéré) du modèle Rborist en classification binaire, en fonction de ses deux hyperparamètres réduits de nombre de prédicteurs X1 et de nombre de références avant scission X2 (interpolation quadratique, points expérimentaux encadrés en noir)

Nous pouvons modéliser la réponse par un modèle quadratique avec interaction[u] :

$$Y = b_0 + b_1.X_1 + b_2.X_2 + b_{12}.X_1.X_2 + b_{11}.X_1^2 + b_{22}.X_2^2$$

Avec Y l'indice J de Youden pondéré J_w, X_1 le facteur réduit dans la plage [0;1] associé à l'hyperparamètre de nombre de prédicteurs testés par scission (*predFixed*), X_2 le facteur réduit associé à l'hyperparamètre de nombre minimal de références distinctes avant scission (*minNode*) et b_n les coefficients des effets. La modélisation permet de calculer les effets suivants :

Le calcul numérique nous permet d'obtenir les estimation des effets :

$$\begin{cases} b_0 = 0.9238 \\ b_1 = 0.421 \\ b_2 = -0.1161 \end{cases} \qquad \begin{cases} b_{12} = 0.1851 \\ b_{11} = -0.3877 \\ b_{22} = -0.0143 \end{cases}$$

Ce modèle quadratique avec interactions permet d'évaluer les hyperparamètres optimaux permettant de maximiser l'indice J de Youden pondéré ($minNode = 1$ et $predFixed = 10$) afin de lancer la prédiction sur un modèle optimisé. Les performances obtenues sont excellentes :

$$\begin{cases} J_{w_{max}} = 1 \\ Spe_{Jw_{max}} = 1 \\ Sen_{Jw_{max}} = 1 \end{cases}$$

Le dernier modèle de forêt aléatoire que nous évaluons dans cette étude est le modèle ranger. Celui-ci dispose de trois hyperparamètres : la taille minimale de nœud (*min.node.size*), le nombre de caractéristiques à séparer à chaque nœud (*mtry*) et la règle contrôlant cette séparation (*splitrule*).

[u] cf. section 4.5, page 43

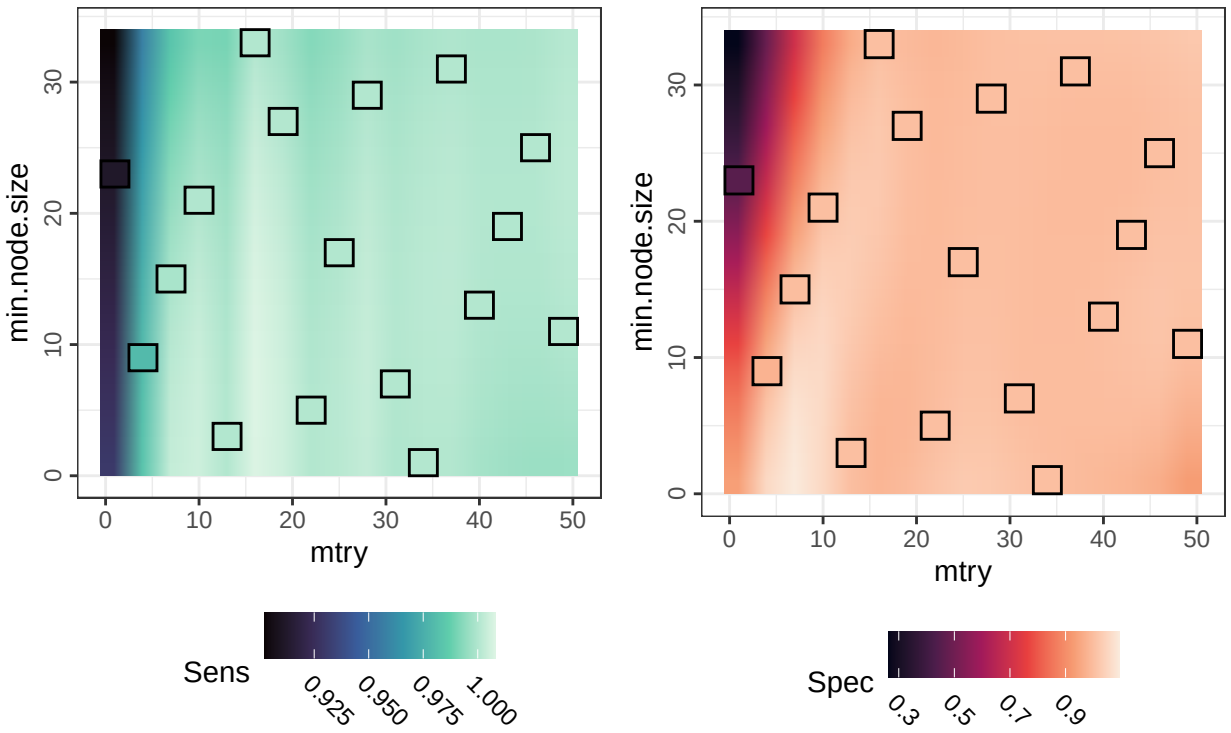


Figure 29: Sensibilité (à gauche) et spécificité (à droite) du modèle Ranger en classification binaire, en fonction des 2 hyperparamètres (algorithme de scission : Gini)

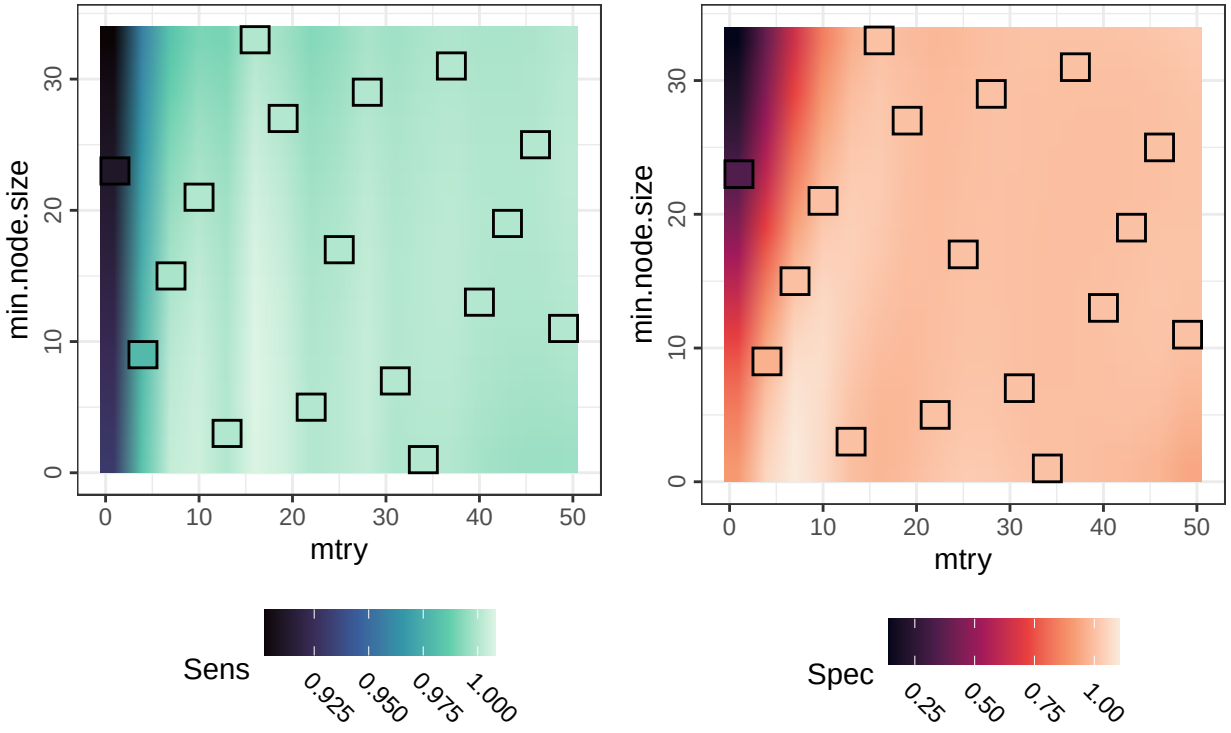


Figure 30: Sensibilité (à gauche) et spécificité (à droite) du modèle Ranger en classification binaire, en fonction des 2 hyperparamètres (algorithme de scission : extratrees)

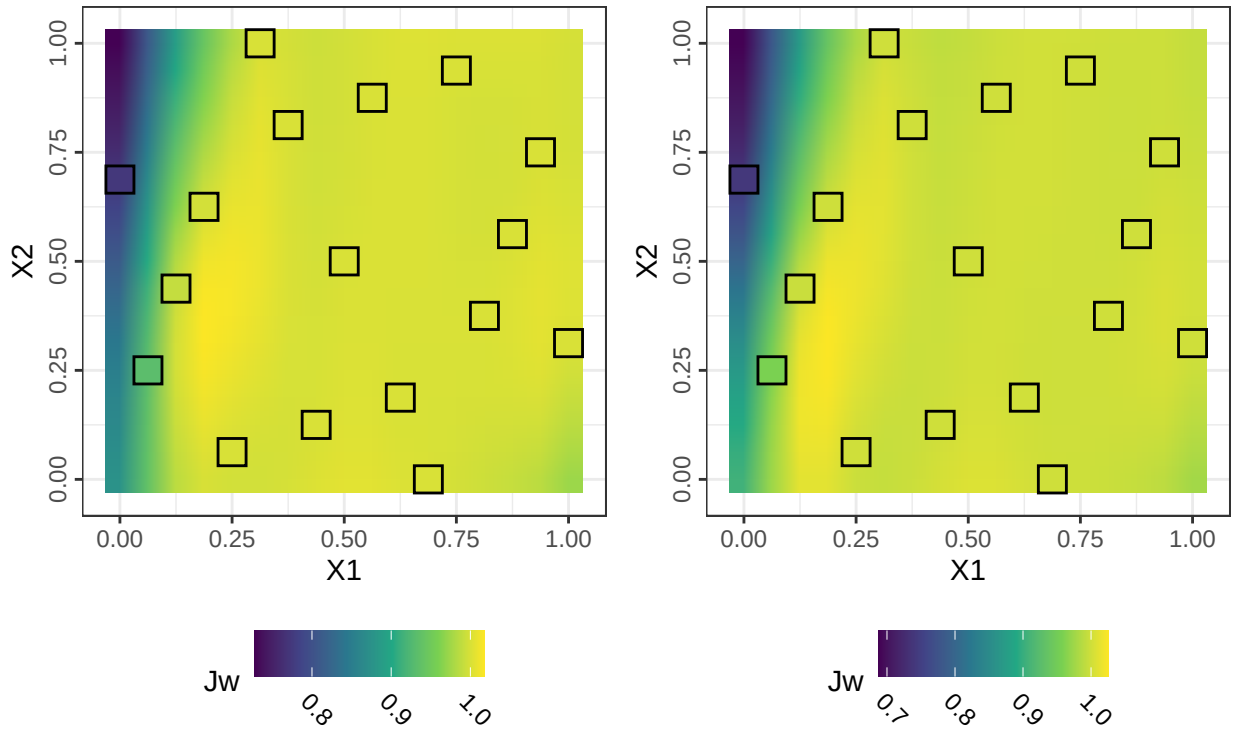


Figure 31: Performances du modèle Ranger en classification binaire, en fonction de l'algorithme de scission (extratrees à gauche, gini à droite) et des hyperparamètres réduits : caractéristiques à séparer X1 et taille minimale de noeud X2 (interpolation quadratique, points expérimentaux encadrés en noir)

Nous pouvons proposer pour ce modèle la modélisation quadratique suivante :

$$Y = b_0 + b_1.X_1 + b_2.X_2 + b_3.X_3 + b_{12}.X_1.X_2 + b_{23}.X_2.X_3 + b_{13}.X_1.X_3 + b_{11}.X_1^2 + b_{22}.X_2^2$$

Avec Y l'indice J de Youden pondéré, X_1 le facteur réduit associé au paramètre de taille minimale de nœud (*min.node.size*), X_2 le facteur réduit associé au paramètre de nombre de caractéristiques à séparer à chaque nœud (*mtry*), X_3 le facteur régissant la règle de séparation (*splitrule*, la valeur 0 étant attribuée à *gini*, 1 à *extratrees*) et b_n les estimations des coefficients des effets. Le facteur X_3 n'ayant que deux niveaux, il est évidemment impossible de lui attribuer une composante quadratique.

La modélisation permet de calculer les effets suivants :

$$\begin{cases} b_0 = 0.9224 \\ b_1 = 0.3981 \\ b_2 = -0.0618 \\ b_3 = -0.0036 \end{cases} \qquad \begin{cases} b_{12} = 0.1829 \\ b_{23} = 0.0081 \\ b_{13} = -8 \times 10^{-4} \\ b_{11} = -0.3957 \\ b_{22} = -0.0536 \end{cases}$$

Le modèle ranger semble déjà, par simple interprétation graphique (voir figures 29, 30 et 31), donner de bons résultats sur une très large plage de l'espace expérimental de ses hyperparamètres, même avec un nombre réduit d'arbres ($n = 6$). Une optimisation des hyperparamètres grâce à la modélisation quadratique (*min.node.size* $= 20$, *mtry* $= 32$ et *splitrule* $=$ *extratrees*) a donné d'excellents résultats :

$$\begin{cases} J_{w_{max}} = 1 \\ Spe_{Jw_{max}} = 1 \\ Sen_{Jw_{max}} = 1 \end{cases}$$

Le modèle Ranger a donné des résultats similaires à ceux du modèle Rborist, avec une sensibilité, une spécificité et un indice de Youden excellents.

Ces résultats soulignent un fait intéressant : tous les modèles de forêts aléatoires ne sont pas égaux. Notre étude montre une différence considérable en sensibilité et spécificité entre les forêts aléatoires de type rFerns, Ranger et Rborist. Lors des étapes préliminaires de cette étude, d'autres algorithmes de forêts aléatoires ont montré de grandes disparités d'efficience sur le plan calculatoire, ce qui nous a conduit à écarter certains modèles pour des raisons pratiques, alors que d'autres se sont avérés sensiblement plus rapides et ont donc pu être retenus pour notre étude.

5.3 Résultats

5.3.1 Protocole d'évaluation

Les modèles ayant atteint les performances requises ($J_w \geq 0.999$) lors de l'étape d'optimisation ont été choisis pour l'évaluation. Les deux modèles retenus sont deux modèles de type forêt aléatoire :

- Forêt aléatoire avec algorithme de type Ranger,
- Forêt aléatoire avec algorithme de type Rborist.

Tous les modèles ont été entraînés sur le jeu de données d'apprentissage, après application des hyperparamètres optimaux obtenus précédemment[v] par modélisation des performances via un modèle quadratique avec interactions. Les performances de nos modèles face au jeu de données d'évaluation, auquel ils n'ont encore jamais été exposés[w], seront évaluées avec le même critère que précédemment : $J_w \geq 0.999$

5.3.2 Performances des modèles de forêts aléatoires

La matrice de confusion du modèle ranger (table 5) donne les résultats détaillés de ses prédictions.

Table 5: Matrice de confusion du modèle Ranger (prédictions à gauche, référence en haut)

	toxique	comestible
toxique	1994	0
comestible	0	1599

La forêt aléatoire de type Ranger a donné d'excellents résultats, sa précision finale étant égale à 1, avec un intervalle de confiance à 95% de [0.999 ; 1], le tout en un temps raisonnable ($24.02 min$), preuve de son efficience calculatoire.

La forêt aléatoire de type Rborist a donné des résultats similaires, atteignant une précision finale égale à 1, avec un intervalle de confiance à 95% de [0.999 ; 1]. Le modèle Rborist, donnant des résultats sensiblement identiques à Ranger, s'est de plus avéré extrêmement efficient sur le plan calculatoire ($3.36 min$).

[v] cf. section 5.2.5

[w] cf. section 4.3, p. 32

Table 6: Performances des modèles Ranger et Rborist (jeu d'évaluation)

	Sensibilité	Spécificité	J de Youden	Durée (min)
Ranger	1	1	1	24.02
Rborist	1	1	1	3.36

6 Apprentissage machine et classification multiclasse

Brouillon, le lot de données utilisé ici est un lot synthétique créé par moi-même (avec algorithme de création fonctionnel), mais à partir des données primaires du Secondary Mushroom Dataset de D.Wagner.

Étant données les performances qu'ont montré les différents modèles lors de la classification binaire, seuls les modèles basés sur les arbres décisionnels et les forêts aléatoires seront évalués dans cette section.

6.1 Classification par familles

6.1.1 Modèles d'arbres de décision

Brouillon, à étoffer et finir, peut-être avec plus de modèles.

Le modèle d'arbre de décision présenté dans cette partie est rpart, le plus simple des modèles CART.

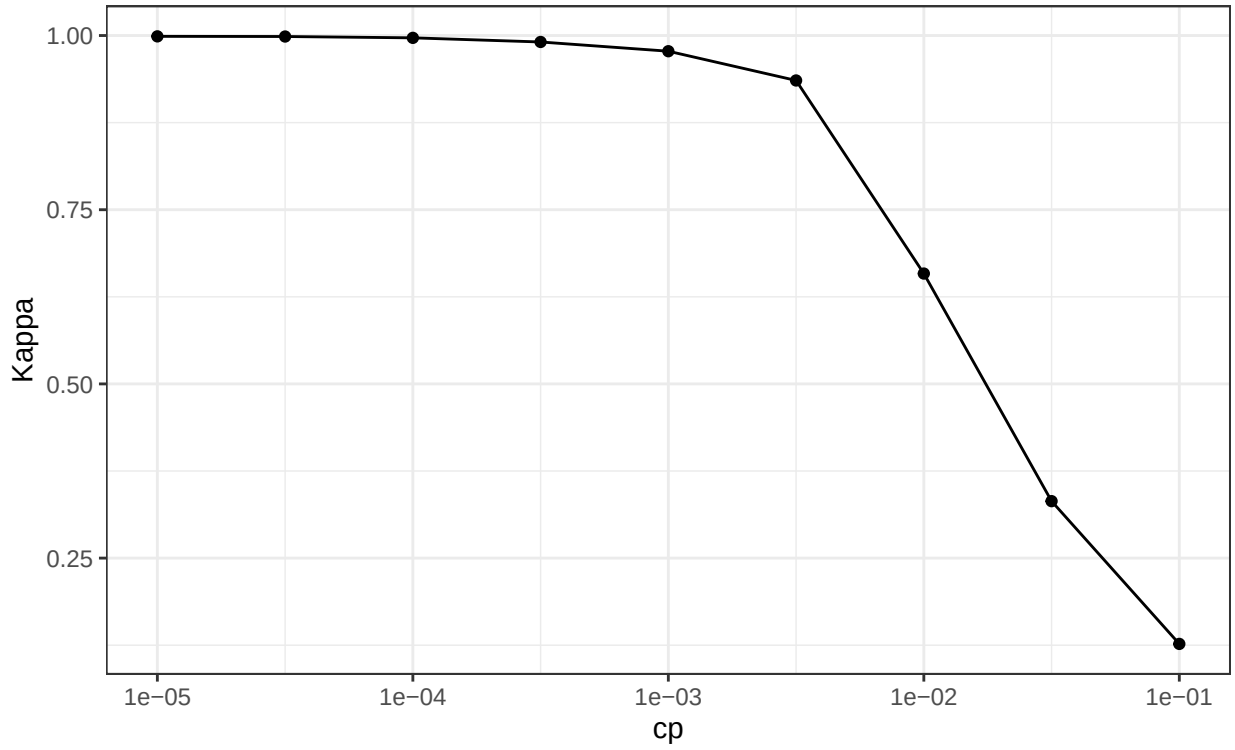


Figure 32: Performances du modèle CART (rpart) dans une classification par familles, en fonction du paramètre de complexité

Ce modèle, pourtant très simple, donne déjà de très bons résultats globaux, avec ($\kappa_{max} = 0.999$ et une précision $R_{max} = 0.999$).

6.1.2 Forêts aléatoires

Le premier modèle de forêt aléatoire évalué dans cette partie est ranger, que nous avons déjà présenté précédemment. Les graphiques des performances en fonction des hyperparamètres laissent entrevoir d'excellentes caractéristiques sur une large plage d'hyperparamètres.

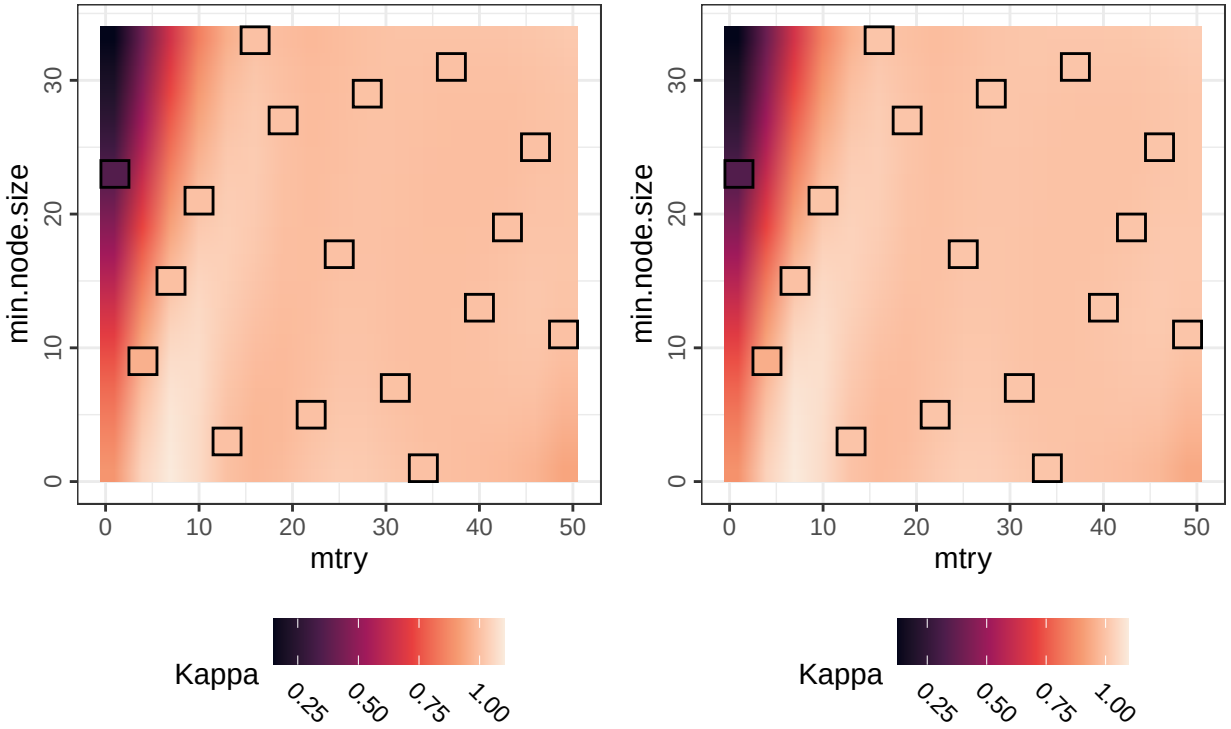


Figure 33: Performances du modèle Ranger dans une classification par familles, en fonction de ses 2 hyperparamètres (algorithme de scission : Gini à gauche, extratrees à droite)

Après optimisation des hyperparamètres (*min.node.size* = 11, *mtry* = 49 et *splitrule* = extratrees), ce modèle a donné d'excellents résultats.

Table 7: Performances du modèle Ranger (hyperparamètres optimaux)

mtry	min.node.size	splitrule	Accuracy	Kappa
49	11	extratrees	0.99997	0.99997

Le dernier modèle de forêt aléatoire est Rborist.

Avec des paramètres optimaux (*predFixed* = 8 et *minNode* = 3), la performance est estimée à :

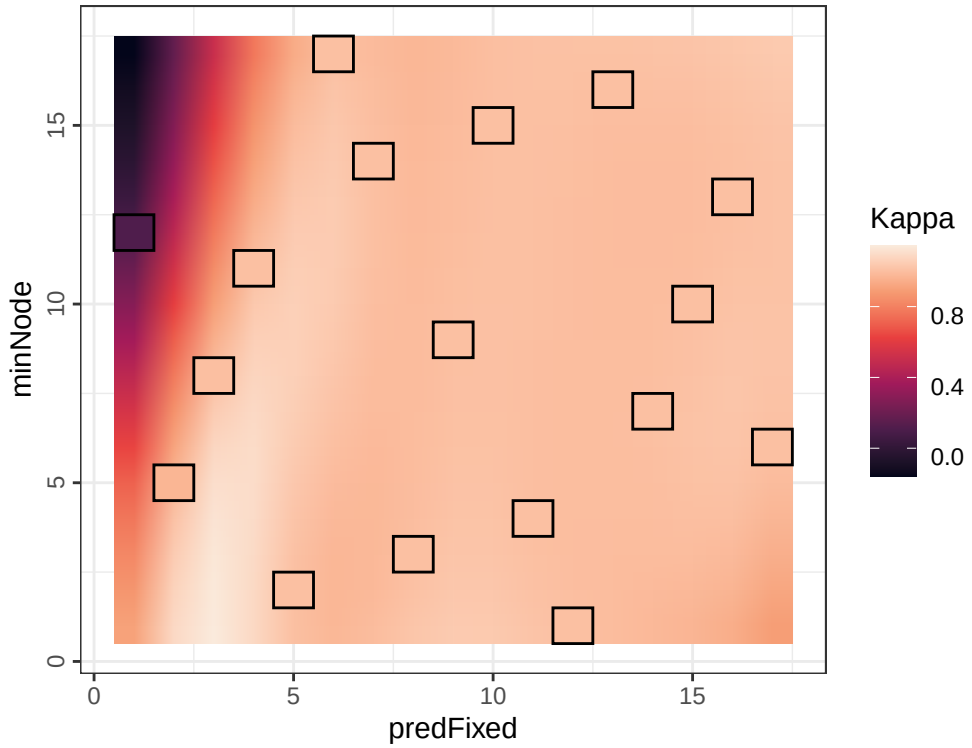


Figure 34: Performances du modèle Rborist dans une classification par familles, en fonction de ses deux hyperparamètres (interpolation quadratique, points expérimentaux encadrés en noir)

Table 8: Performances du modèle Rborist (hyperparamètres optimaux)

predFixed	minNode	Accuracy	Kappa
8	3	0.99999	0.99999

Le modèle Rborist a donné des résultats similaires à ceux du modèle Ranger, avec d'excellentes performances.

6.1.3 Résultats

Les critères et le protocole de l'évaluation sont les mêmes que ceux évoqués précédemment.

L'évaluation finale du modèle ranger donne la matrice de confusion de la table 9.

La précision finale est égale à $R = 1$, avec un intervalle de confiance à 95% de [0.9996 ; 1]. La forêt aléatoire de type Ranger a donné d'excellents résultats, en un temps très raisonnable (1.74 min).

La forêt aléatoire de type Rborist a donné des résultats similaires, avec une précision finale égale à 1, avec un intervalle de confiance à 95% de [0.9996 ; 1]. Le modèle Rborist, donnant des résultats sensiblement identiques à Ranger, s'est avéré plutôt efficient sur le plan calculatoire (9.66 min).

Table 9: Matrice de confusion du modèle Ranger (prédictions à gauche, références en haut)

	Amanita	Bolbitius	Bolete	Bracket_Fungi	Chanterelle	Cortinarius	Crepidotus	Ear_Pick	Entoloma	Hydnum	Ink_Cap	Jelly_Discs	Lepiota	Morel	Mushroom	Oyster_Mushroom	Paxillus	Pluteus	Russula	Saddle_Cup	Stropharia	Tricholoma	Wax_Gill
Amanita	381	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bolbitius	0	142	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bolete	0	0	666	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bracket_Fungi	0	0	0	335	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Chanterelle	0	0	0	0	142	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cortinarius	0	0	0	0	0	524	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Crepidotus	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ear_Pick	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Entoloma	0	0	0	0	0	0	0	0	334	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hydnum	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0
Ink_Cap	0	0	0	0	0	0	0	0	0	0	621	0	0	0	0	0	0	0	0	0	0	0	0
Jelly_Discs	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	0	0
Lepiota	0	0	0	0	0	0	0	0	0	0	0	0	143	0	0	0	0	0	0	0	0	0	0
Morel	0	0	0	0	0	0	0	0	0	0	0	0	0	48	0	0	0	0	0	0	0	0	0
Mushroom	0	0	0	0	0	0	0	0	0	0	0	0	0	0	238	0	0	0	0	0	0	0	0
Oyster_Mushroom	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	95	0	0	0	0	0	0	0
Paxillus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	143	0	0	0	0	0	0
Pluteus	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0
Russula	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1285	0	0	0	0
Saddle_Cup	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0
Stropharia	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	381	0	0
Tricholoma	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2046	0
Wax_Gill	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	382

Nous pouvons noter que le modèle ranger s'est ici avéré plus rapide que Rborist.

Table 10: Performances des modèles Ranger et Rborist (évaluation)

	Précision	Kappa	Durée (min)
Ranger	1	1	1.74
Rborist	1	1	9.66

6.2 Classification par espèce

Dans cette partie, la difficulté de la classification augmente sensiblement, les modèles ne sont plus chargés de classifier les champignons par familles, mais de déterminer précisément l'espèce de chaque spécimen du lot de données. Les modèles utilisés dans cette partie sont les mêmes que ceux de la classification par familles.

6.2.1 Modèles d'arbres de décision

Comme précédemment, le modèle d'arbre de décision retenu pour la classification par espèces est rpart.

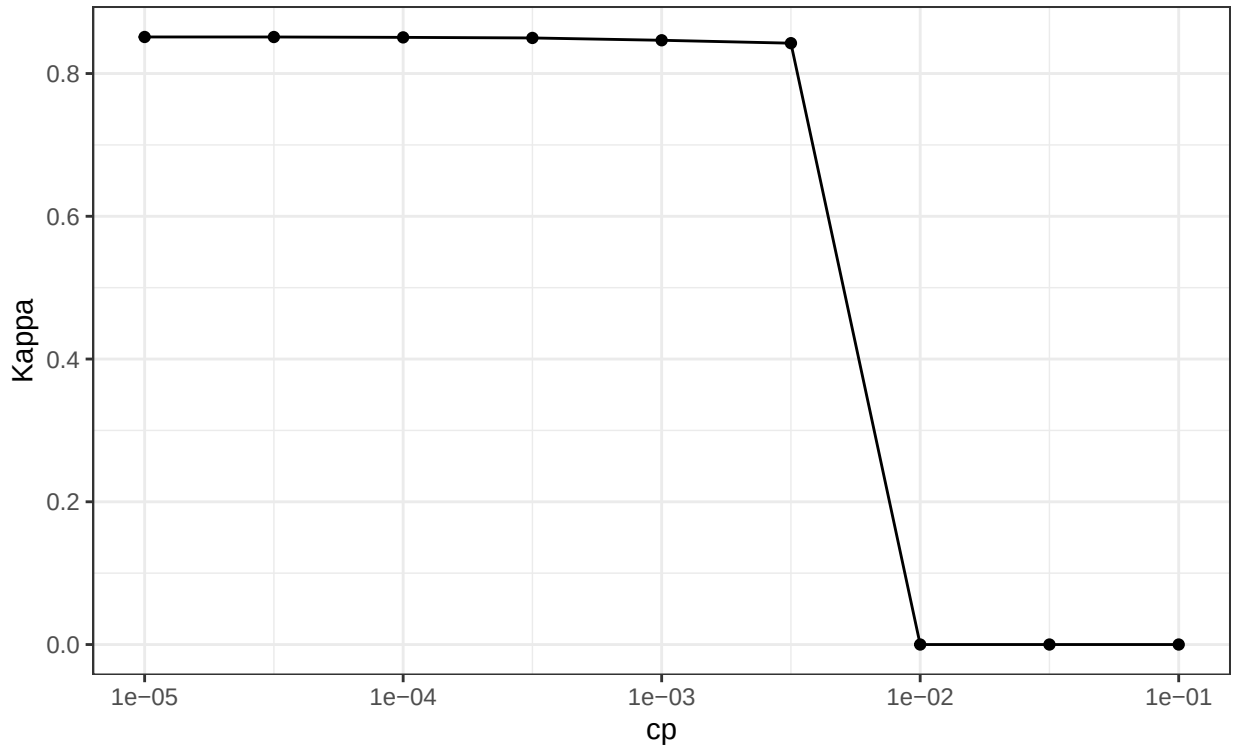
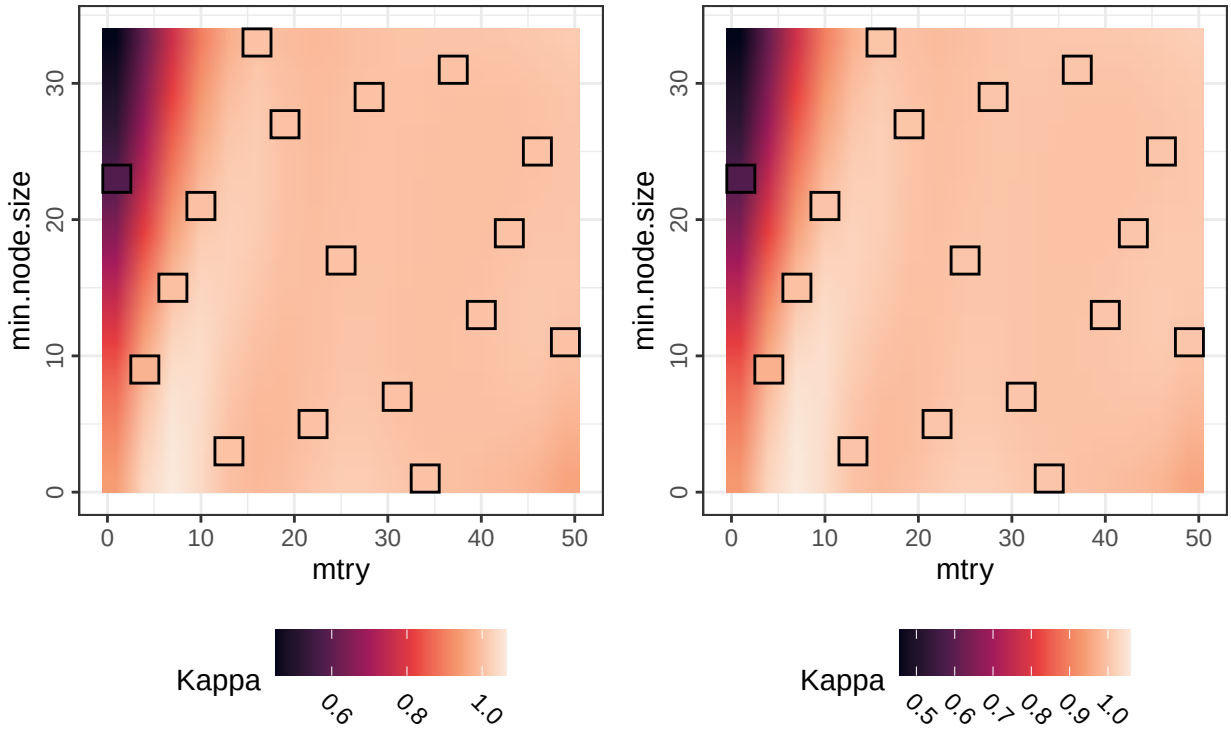


Figure 35: Performances du modèle CART (rpart) dans une classification par espèces, en fonction du paramètre de complexité

Ce modèle, bien que relativement simple, donne encore des résultats honorables, bien que le kappa ($\kappa_{max} = 0.851$) comme la une précision ($R_{max} = 0.852$) n'atteignent pas les objectifs de cette étude.

6.2.2 Forêts aléatoires

Comme lors de la classification par familles, notre premier modèle de forêt aléatoire évalué dans cette partie est ranger. L'exploration de l'espace expérimental des hyperparamètres de ce modèle laisse entrevoir de très bonnes performances sur une large plage d'hyperparamètres.

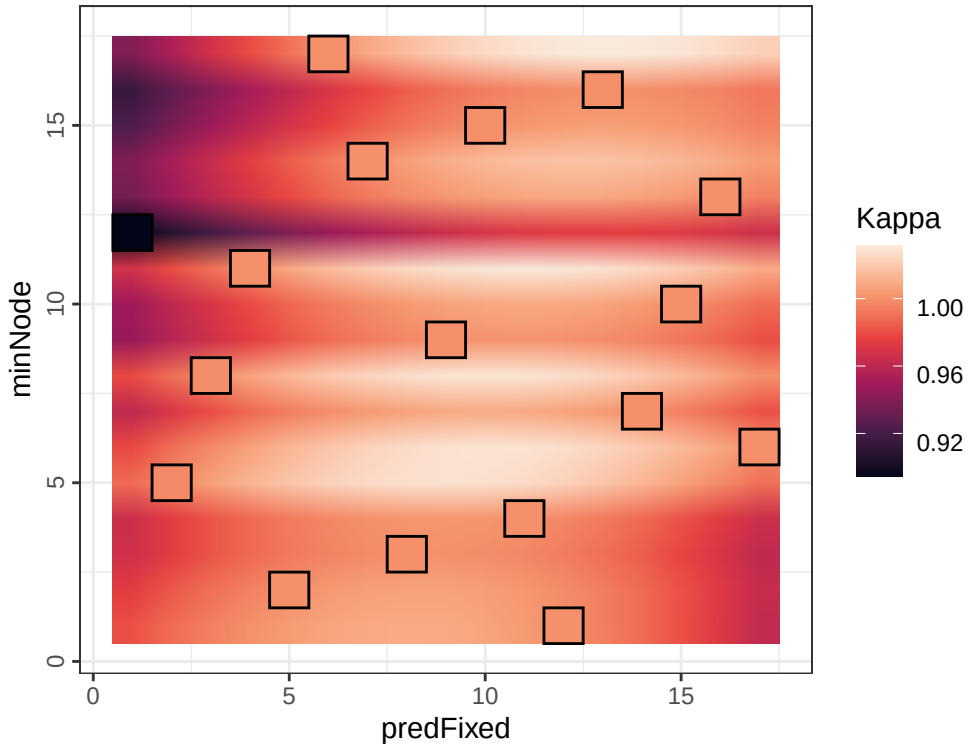


Après optimisation des hyperparamètres (*min.node.size* = 25, *mtry* = 46 et *splitrule* = gini), ce modèle a donné, comme lors des tests précédents, d'excellents résultats, malgré la complexité accrue du problème.

Table 11: Performances du modèle Ranger (hyperparamètres optimaux)

mtry	min.node.size	splitrule	Accuracy	Kappa
46	25	gini	0.99974	0.99974

Le dernier modèle de forêt aléatoire exploité dans cette tâche de classification par espèces est Rborist.



Avec des paramètres optimaux (*predFixed* = 7 et *minNode* = 14), la performance est estimée à :

Table 12: Performances du modèle Rborist (hyperparamètres optimaux)

predFixed	minNode	Accuracy	Kappa
7	14	0.99987	0.99987

Le modèle Rborist a donné des résultats similaires à ceux du modèle Ranger, avec d'excellentes performances en phase d'apprentissage et optimisation.

6.2.3 Résultats

Les critères et le protocole de l'évaluation sont les mêmes que ceux mentionnés pour les autres tâches de classification.

La précision finale du modèle Ranger est égale à $R = 0.9996$, avec un intervalle de confiance à 95% de [0.9996 ; 0.9999]. La forêt aléatoire de type Ranger a donné d'excellents résultats, en un temps très raisonnable (1.74 min).

La matrice de confusion, ne reprenant ici que les espèces ayant posé problème à notre modèle (3:8239), est résumée par la table 13.

Table 13: Matrice de confusion des erreurs de Ranger, (prédictions à g., références en h.)

	Ear_pick_Fungus	Milky_Bell_Cap	Oak_Milk_Cap	Red_cracked_Bolete	Rufous_Milk_Cap	Yellow_cracked_Bolete
Milky_Bell_Cap	1	48	0	0	0	0
Rufous_Milk_Cap	0	0	1	0	48	0
Yellow_cracked_Bolete	0	0	0	1	0	48

La forêt aléatoire de type Rborist a donné des résultats très proches de ceux obtenus par le modèle ranger, avec un taux d'erreur de 1:8239.

La précision finale de Rborist est ainsi égale à 0.9999, avec un intervalle de confiance à 95% de [0.9993 ; 1]. Le modèle Rborist, donnant des résultats proches à Ranger, s'est de plus avéré assez efficient sur le plan calculatoire (12.96 min).

Nous pouvons noter que si Rborist affiche des performances marginalement supérieures, le modèle ranger s'est encore une fois avéré sensiblement plus rapide que Rborist sur des problèmes complexes.

Table 14: Performances des modèles Ranger et Rborist (évaluation)

	Précision	Kappa	Durée (min)
Ranger	0.99964	0.99963	6.01
Rborist	0.99988	0.99988	12.96

7 Robustesse de la classification

Partie facultative, assez ambitieux, voir si cette partie est compatible (en temps) avec la création du lot de données, qui va être assez chronophage. . .

Evaluation de la robustesse, avec deux stratégies envisageables :

1. Robustesse "intrinsèque" : lot d'entraînement standard, lot d'évaluation avec variations. Robustesse des modèles seuls.

2. Robustesse "intégrée" : intégration de la robustesse dès le départ, avec lot d'entraînement intégrant des variations représentatives du lot d'évaluation.

7.1 Robustesse face aux déviations

Robustesse face à des données hors-normes, sans être extrêmes pour autant.

Facile à coder pour valeurs quantitatives (simplement augmenter la SD du générateur. . .)

DIFFICILE à coder pour les valeurs qualitatives (i.e. trouver des proximités, soit manuellement, soit via clustering / KNN)

Robustesse "intrinsèque" et "intégrée"

7.2 Robustesse face aux erreurs

Robustesse face à des données complètement aberrantes.

Facile à coder. Robustesse "intrinsèque" uniquement.

48. Genuer R, Poggi JM. Random Forests with R [Internet]. Cham: Springer International Publishing; 2020 [cité 19 mai 2023]. (Use R!). Disponible sur: http://link.springer.com/10.1007/978-3-030-56485-8

49. Santiago J, Claeys-Bruno M, Sergent M. Construction of space-filling designs using WSP algorithm for high dimensional spaces. Chemometrics and Intelligent Laboratory Systems [Internet]. avr 2012 [cité 4 mars 2023];113:26-31. Disponible sur: https://linkinghub.elsevier.com/retrieve/pii/S0169743911001195

50. Youden WJ. Index for rating diagnostic tests. Cancer [Internet]. 1950 [cité 4 mars 2023];3(1):32-5. Disponible sur: https://onlinelibrary.wiley.com/doi/abs/10.1002/1097-0142%281950%293%3A1%3C32%3A%3AAID-CNCR2820030106%3E3.0.CO%3B2-3

51. Rücker G, Schumacher M. Summary ROC curve based on a weighted Youden index for selecting an optimal cutpoint in meta-analysis of diagnostic accuracy. Statistics in Medicine [Internet]. 2010 [cité 4 mars 2023];29(30):3069-78. Disponible sur: http://onlinelibrary.wiley.com/doi/abs/10.1002/sim.3937

52. Cohen J. A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement [Internet]. avr 1960 [cité 12 mars 2023];20(1):37-46. Disponible sur: https://doi.org/10.1177/001316446002000104

53. Landis JR, Koch GG. The Measurement of Observer Agreement for Categorical Data. Biometrics [Internet]. 1977 [cité 12 mars 2023];33(1):159-74. Disponible sur: http://www.jstor.org/stable/2529310

A Annexe : développement d'un algorithme de génération de lot synthétique

Plus de détails concrets sur la méthode utilisée en pratique, avec extraits de code...

Pour l'instant, l'algorithme est écrit et fonctionne, à partir des données primaires du Secondary Dataset de Dennis Wagner. C'est cet algorithme qui a servi à créer le lot de données de la classification multiclasse.

Pour avoir mes propres lots de données, il ne me reste plus qu'à avoir mes propres données primaires, c'est à dire entrer manuellement les caractéristiques clés de mes 400+ champignons, un par un...

La seule bibliothèque utilisée lors de la création du lot de données synthétique est le *tidyverse*,[7] collection de bibliothèques spécialisées dans le domaine de la *data science* et notamment dédiées au traitement, au nettoyage et à la visualisation de données.

```
library(tidyverse)
```

Notre algorithme charge ensuite le fichier zip, incluant le fichier csv contenant les caractéristiques typiques des macromycètes (type de sporophore, dimensions maximales du stipe, du chapeau, type de lames, couleur de sporée, etc.), qui est lu et attribué à un *dataframe*. Les lignes commentées correspondent à l'utilisation d'un fichier identique hébergé à distance sur un dépôt GitHub.

```
fichier_data <- tempfile()
#URL <- "https://github.com/EKRihani/mushrooms/raw/master/MushroomDataset.zip"
#download.file(URL, fichier_data)
fichier_data <- "~/projects/champis/MushroomDataset.zip"
fichier_data <- unzip(fichier_data, "MushroomDataset/primary_data.csv")
data_champis <- read.csv(fichier_data,
                         header = TRUE,
                         sep = ";",
                         stringsAsFactors = TRUE)
```

etc etc.

B Annexe : outils d'analyse exploratoire des données (EDA)

Code EDA... A développer une fois le jeu de données primaires créé...

C Annexe : développement des algorithmes d'apprentissage machine

Nous détaillerons ici principalement les algorithmes utilisés pour le classifieur binaire. Les particularités d'intérêt des classifieurs multiples seront évoquées brièvement lors des développements de cette section.

C.1 Initialisation

Les bibliothèques utilisées lors des étapes d'apprentissage machine sont :

- tidyverse,[7] collection de bibliothèques spécialisées dans le domaine de la *data science*,
- DiceDesign,[22] bibliothèque spécialisée dans la création de plans d'expériences hypercubiques,
- DiceEval,[23] bibliothèque spécialisée dans la modélisation des résultats de plans d'expériences hypercubiques,
- caret,[10] collection d'outils dédiés à l'apprentissage machine.
- twinning,[12] outils dédiés à la génération de jeux de données d'entraînement, optimisation, validation équilibrés.

```
library(tidyverse)
library(DiceDesign)
library(DiceEval)
library(caret)
library(twinning)
```

Le chargement des données s'effectue de la même façon que lors des sections précédentes. L'argument *stringsAsFactors* = *TRUE* revêt une importance particulière, car la classe *factor* est essentielle au bon fonctionnement des classifieurs. Dans le cadre d'une classification binaire, nous définissons arbitrairement, à l'aide de la fonction *relevel*, la classe *"toxique"* comme étant la valeur positive. Cette définition n'est pas nécessaire pour les classifieurs multiclasses.

```
fichier_data <- tempfile()
fichier_data <- "~/projects/champis/MushroomDataset.zip"
fichier_data <- unzip(fichier_data, "MushroomDataset/secondary_data.csv")
dataset <- read.csv(fichier_data,
                    header = TRUE,
                    sep = ";",
                    stringsAsFactors = TRUE)
dataset$class <- recode_factor(dataset$class, e = "comestible", p = "toxique")
# /!\ recode utilisé uniquement pour le Wagner !!!
dataset$class <- relevel(dataset$class, ref = "toxique")
```

C.2 Création des jeux d'entraînement, optimisation et évaluation

La création du jeu d'évaluation s'effectue en deux étapes. La première est la définition des rapports des dichotomies entre jeux d'entraînement et optimisation d'une part, et d'évaluation d'autre part. Cette définition implique l'évaluation du nombre d'individus, le calcul du nombre de coefficients p, puis du rapport de dichotomie $f = \sqrt(p) + 1$.[x]

```
BI_n_champis <- nrow(dataset)
BI_split_p <- sqrt(BI_n_champis)
BI_split_facteur <- round(sqrt(BI_split_p)+1)
```

La seconde partie consiste à effectuer la scission proprement dite. Cette scission implique la définition d'une liste d'index, de fraction 1 : f du nombre d'individus, qui servira via inclusion (jeu d'évaluation) ou exclusion (jeu d'apprentissage et d'évaluation) booléennes des lignes correspondantes, à de constituer chaque jeu de données. La fonction *set.seed* assure la reproductibilité.

```
set.seed(7)
index1 <- twin(data = dataset, r = BI_split_facteur)
BI_lot_appr_opti <- dataset[-index1,]
BI_lot_evaluation <- dataset[index1,]
```

Les lots ainsi obtenus seront ensuite utilisés pour l'entraînement, l'optimisation et l'évaluation finale des performances des modèles.

C.3 Entraînement et optimisation des modèles

Cette partie ne prétend pas à l'exhaustivité, elle se limitera à la présentation de l'entraînement, l'optimisation et la génération de graphiques pour deux modèles : un modèle CART (rpart) et un modèle RF (Rborist). Un certain nombre de tâches telles que l'entraînement du modèle ou la génération de graphiques sont en réalité attribuées à des fonctions créées *adhoc* dans le but de clarifier l'organisation du code de l'algorithme, car elles sont effectuées à de nombreuses reprises. Nous décrirons ici le code source sans faire appel à ces fonctions.

C.3.1 Arbre de classification et régression

La première étape est de définir l'indice J de Youden,[y] et les pondérations respectives de la sensibilité et de la spécificité. Cette définition n'est pas nécessaire pour les classifieurs multiclasse, le kappa (κ) et l'indice de Rand (R) étant des métriques évaluées nativement par la librairie *caret*.

```
BI_w <- 10
BI_RatioSens <- 2*BI_w/(BI_w+1)
BI_RatioSpec <- 2*(1-BI_w/(BI_w+1))
```

[x] cf. section 4.2, page 31.
[y] cf. section 4.6, page 44

La seconde définition à préciser est celle de l'espace expérimental des hyperparamètres. En l'espèce le seul hyperparamètre du modèle rpart est la variable *cp*.

```
BI_grid_rpart_cp <- data.frame(cp = 10^seq(from = -5, to = -1, by = .5))
```

L'étape suivante est de définir les paramètres d'entraînement et d'évaluation des performances du modèle en vue de son optimisation. La fonction *trainControl* permet ici de préciser les principaux paramètres régissant cette étape :

- *classProbs*, *[TRUE indispensable au fonctionnement...]*
- *summaryFunction*, afin d'indiquer que les métriques de performance à utiliser sont celles d'un classifieur binaire (l'argument *multiClassSummary* sera utilisé pour un classifieur multiclasse)
- *method*, afin de préciser la méthode de construction des jeux d'entraînement et d'optimisation, ici validation croisée (*CV : cross-validation*)
- *number*, afin d'indiquer le nombre de blocs de la validation croisée, calculé précédemment.

Ici aussi, la fonction *set.seed* assure la reproductibilité du processus.

```
set.seed(1)
tr_ctrl <- trainControl(classProbs = TRUE,
                        summaryFunction = twoClassSummary,
                        method = "cv",
                        number = BI_split_facteur)
```

L'entraînement du modèle peut avoir lieu. En l'espèce, le modèle mathématique retenu est l'attribution d'une prédiction sur *class* en fonction de toutes les autres variables (*class* $\sim$.). Les arguments *data*, *trControl*, *tuneGrid* font appel aux éléments décrits dans les paragraphes qui précèdent.

```
BI_fit_rpart_cp <- train(class ~ .,
                         method = "rpart",
                         data = BI_lot_appr_opti,
                         trControl = tr_ctrl,
                         tuneGrid  = BI_grid_rpart_cp)
```

L'objet résultant est d'une structure relativement complexe. Notre algorithme peut notamment en extraire les résultats relatifs aux performances du modèle, et y adjoindre le calcul du J de Youden pondéré J_w. Dans le cadre des classifieurs multiclasses, ce calcul est inutile, l'objet généré à l'étape précédente contenant déjà les indicateurs de performance que nous utilisons : kappa (κ) et indice de Rand (*R*, *accuracy*).

```
BI_fit_rpart_cp_resultats <- BI_fit_rpart_cp$results %>%
   mutate(Jw = Sens*BI_RatioSens + Spec*BI_RatioSpec - 1)
```

Table 15: Tableau des résultats de l'entraînement de rpart

cp	ROC	Sens	Spec	ROCSD	SensSD	SpecSD	Jw
0.0000100	0.9992475	0.9977113	0.9974202	0.0005176	0.0014069	0.0014071	0.9953696
0.0000316	0.9992344	0.9975859	0.9974593	0.0005374	0.0014608	0.0012481	0.9951487
0.0001000	0.9992092	0.9975545	0.9973029	0.0005223	0.0014356	0.0012967	0.9950632
0.0003162	0.9987400	0.9956733	0.9963646	0.0007393	0.0026099	0.0012678	0.9914724
0.0010000	0.9972132	0.9904371	0.9894068	0.0007871	0.0031615	0.0027640	0.9806869
0.0031623	0.9848444	0.9641001	0.9557501	0.0042545	0.0104682	0.0095868	0.9266819
0.0100000	0.8983138	0.9286069	0.7720671	0.0141701	0.0192717	0.0380337	0.8287520
0.0316228	0.7011303	0.5726145	0.8032593	0.0142681	0.0320282	0.0150998	0.1871645
0.1000000	0.6176520	0.5355555	0.6997485	0.0111304	0.0146717	0.0167098	0.1009643

L'objet de type *dataframe* ainsi créé peut être appelé afin d'en extraire des résultats d'intérêt ou d'en inclure le tableau dans le rapport :

L'algorithme génère également un graphique synthétisant les performances du modèle (sensibilité, spécificité, J_w, κ, R ou autre indicateur d'intérêt) en fonction de son hyperparamètre :

- *ggplot* est la fonction de génération du graphique, et permet d'appeler l'objet servant à générer le graphique, ainsi que certains paramètres complémentaires via *aes*. Ici, la variable servant d'abscisse.
- *geom_point* permet de tracer le nuage de points. Ici encore, *aes* permet de préciser, pour chaque nuage de points, la variable d'ordonnée (*Sens*, *Spec* ou *Jw*), ainsi que la légende associée à la couleur des points (*Sensibilité*, *Spécificité* ou *Jw*).
- *geom_line* permet de tracer les lignes correspondant au nuage de points.
- *labs* permet de légender correctement l'attribut *color* de notre légende.
- *ylab* permet de définir la légende l'axe des ordonnées. Ici, de la supprimer, car nous avons trois variables différentes en ordonnées.
- *scale_x_log10* nous permet ici de définir un axe logarithmique décimal en abscisse.
- *theme_bw* attribue le thème(couleur de fond, d'axes, grilles) de type *bw* (*black and white*) à notre graphique.

```
BI_fit_rpart_cp_graphe <- ggplot(data = BI_fit_rpart_cp_resultats, aes(x = cp)) +
   geom_point(aes(y = Sens, color = "Sensibilité")) +
   geom_line(aes(y = Sens, color = "Sensibilité")) +
   geom_point(aes(y = Spec, color = "Spécificité")) +
   geom_line(aes(y = Spec, color = "Spécificité")) +
   geom_point(aes(y = Jw, color = "Jw")) +
   geom_line(aes(y = Jw, color = "Jw")) +
   labs(color = "Performance") +
   ylab(NULL) +
   scale_x_log10() +
   theme_bw()
```

Le graphique ainsi généré peut être intégré dans notre rapport :

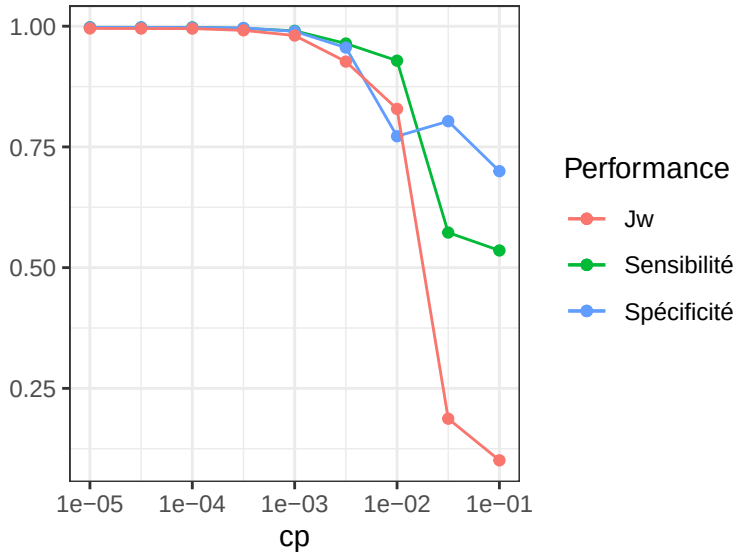


Figure 36: Graphique des performances de rpart

C.3.2 Rborist

La première étape est, comme précédemment, de définir l'espace expérimental. Ici, s'agissant d'un modèle à plusieurs hyperparamètres, l'algorithme utilisera un plan d'expériences basé sur les hypercubes latins. La fonction *nolhDesign* permet de créer un hypercube latin quasi-orthogonal (NOLH : *Near Orthogonal Latin Hypercube*), ici paramétré avec 2 dimensions, dans l'espace $[0;1]^2$. Le plan d'expérience en est extrait, puis inséré dans un objet de type *dataframe*, avec colonnes nommées d'après nos variables réduites X_1 et X_2.

```
BI_LHS <- nolhDesign(dimension = 2, range = c(0, 1))$design
BI_LHS <- data.frame(BI_LHS)
colnames(BI_LHS) <- c("X1", "X2")
```

L'hypercube latin des hyperparamètres (*predFixed* et *minNode*) est généré à partir de l'hypercube latin des paramètres réduits. Ces hyperparamètres sont des valeurs entières. L'hypercube latin quasi-orthogonal de dimension 2 possédant 17 expériences équitablement réparties dans l'espace $[0;1]^2$, il apparaît souhaitable, pour des raisons d'homogénéité dans l'espace expérimental, que $F_n = k \times X_n$ avec k multiple de 16. En pratique, nous n'avons jamais rencontré d'erreurs d'arrondi lors de cette étape mais l'usage de la fonction *round* constitue une précaution supplémentaire garantissant que le produit sera bien un entier.

```
BI_grid_Rborist <- data.frame(BI_LHS) %>%
   mutate(predFixed = round(1+X1*16,0)) %>%
   mutate(minNode = round(1+X2*16,0))
```

Table 16: Plan d'expériences d'entraînement et optimisation du modèle Rborist

X1	X2	predFixed	minNode
0.3125	1.0000	6	17
0.0625	0.2500	2	5
0.1250	0.4375	3	8
0.1875	0.6250	4	11
0.7500	0.9375	13	16
1.0000	0.3125	17	6
0.6250	0.1875	11	4
0.5625	0.8750	10	15
0.5000	0.5000	9	9
0.6875	0.0000	12	1
0.9375	0.7500	16	13
0.8750	0.5625	15	10
0.8125	0.3750	14	7
0.2500	0.0625	5	2
0.0000	0.6875	1	12
0.3750	0.8125	7	14
0.4375	0.1250	8	3

L'entraînement du modèle se déroule de la même façon que pour le modèle rpart.[z]

```
set.seed(1)
tr_ctrl <- trainControl(classProbs = TRUE,
                        summaryFunction = twoClassSummary,
                        method = "cv",
                        number = BI_split_facteur)
BI_fit_Rborist <- train(class ~ .,
                        method = "Rborist",
                        data = BI_lot_appr_opti,
                        trControl = tr_ctrl,
                        tuneGrid  = BI_grid_Rborist[c('predFixed', 'minNode')])
```

L'algorithme extrait les résultats relatifs aux performances du modèle et y adjoint le calcul de J_w (ou κ pour les classifieurs multiclasses), comme précédemment. L'objet obtenu ne contenant que les facteurs expérimentaux (non réduits) des hyperparamètres, il convient d'y adjoindre les facteurs réduits. La table *BI_grid_Rborist* générée précédemment contient toutes les informations qui permettent, via une jonction, de lier facteurs réduits et facteurs expérimentaux.

[z] cf. section C.3.1, page 83.

```
BI_fit_Rborist_resultats <- BI_fit_Rborist$results %>%
   mutate(Jw = Sens*BI_RatioSens + Spec*BI_RatioSpec - 1) %>%
   left_join(x = .,
             y = BI_grid_Rborist,
             by = c("predFixed", "minNode"))
```

Nous chargeons ensuite notre algorithme de calculer le modèle quadratique avec interactions permettant d'évaluer, à partir des résultats obtenus suite à l'entraînement, la réponse J_w en fonction des X_1 et X_2, suivant la formule :

$$Y = b_0 + b_1.X_1 + b_2.X_2 + b_{12}.X_1.X_2 + b_{11}.X_1^2 + b_{22}.X_2^2$$

```
BI_mod_Rborist_jw <-  modelFit(X = BI_fit_Rborist_resultats[,c("X1", "X2")],
                               Y = BI_fit_Rborist_resultats$Jw,
                               type="Kriging",
                               formula= Y ~ X1 + X2 + X1:X2 + I(X1^2) + I(X2^2))
```

Ces résultats permettent notamment de modéliser les performances sur la totalité de l'espace expérimental des hyperparamètres. A cette fin, l'algorithme est chargé de générer l'ensemble des couples de valeurs (X_1, X_2) possibles, à l'aide de la fonction *expand.grid*, avant de calculer la valeur J_w correspondante à chaque couple de points.

```
CodBI_pred_Rborist <- expand.grid(CodBI_fit_Rborist_resultats[,c("X1","X2")]) %>%
   mutate(Jw = modelPredict(CodBI_mod_Rborist_jw, .[,c("X1","X2")]))
```

L'ensemble des données expérimentales et modélisées obtenues permettent de générer un graphique bidimensionnel des performances en fonction des hyperparamètres. Pour des raisons didactiques, nous séparerons ici le graphique résultant de l'expérimentation de celui résultant de la modélisation quadratique.

La génération du graphique reprend des principes similaires à ceux présentés précédemment pour le graphique des performances de rpart, notamment au niveau des arguments utilisés dans *aes*. Les seules fonctions appelant à commentaires sont l'utilisation de *geom_raster* pour la modélisation, à laquelle nous superposons le graphique généré via *geom_tile* pour les points expérimentaux. L'utilisation de la gamme de couleurs proposée par *viridis* permet une visualisation plus aisée des résultats obtenus.

```
BI_pred_Rborist %>% ggplot() +
   geom_raster(data = BI_pred_Rborist,
               aes(x = X1, y = X2, fill = Jw), interpolate = TRUE) +
   geom_tile(data = BI_fit_Rborist_resultats,
             aes(x = X1, y = X2, fill = Jw), color = 'black', linewidth =.5) +
   scale_fill_viridis_c(option = "D", direction = 1) +
   theme(axis.text.y = element_text(angle=90, vjust=.5, hjust=.5)) +
   theme_bw()
```

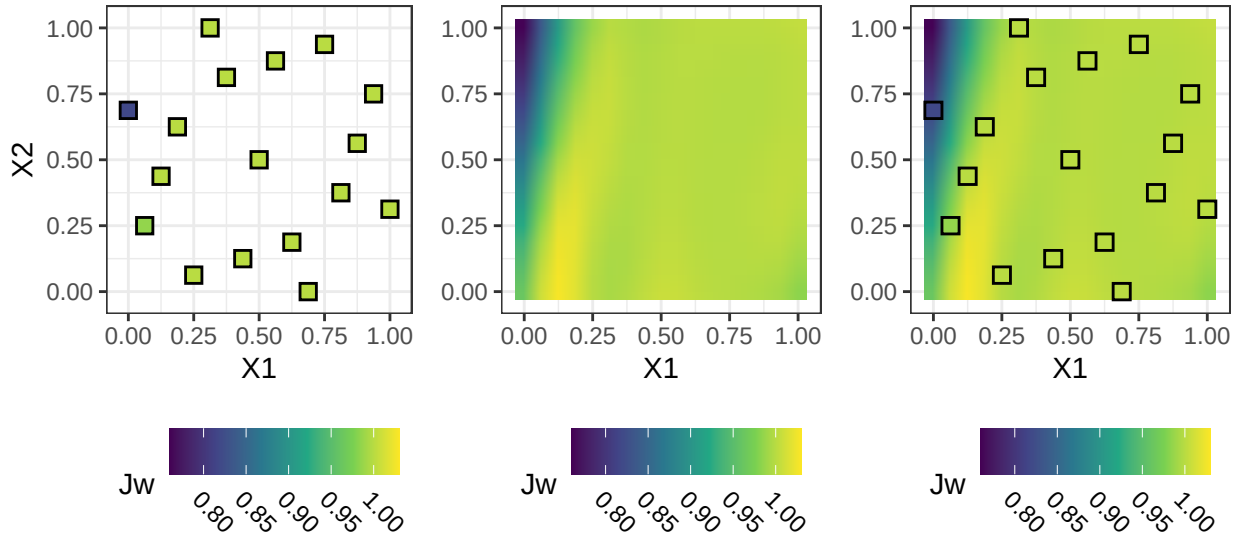


Figure 37: Points expérimentaux (à gauche), modélisation (au centre) et superposition (à droite) des résultats obtenus avec Rborist

L'étape suivante est d'exploiter ce modèle quadratique pour évaluer les hyperparamètres permettant de maximiser J_w. A cet effet, une table de l'ensemble des points de l'espace expérimental des facteurs réduits est générée, puis la fonction quadratique évaluée précédemment est appliquée, afin d'obtenir une prédiction approximative de J_w. Les hyperparamètres sont également calculés, afin de pouvoir être appliqués par la suite.

```
BI_modelquad_Rborist <- expand.grid(X1 = seq(from = 0, to = 1, length.out = 17),
                                    X2 = seq(from = 0, to = 1, length.out = 17)) %>%
   mutate(Jw = BI_mod_Rborist_jw$model@trend.coef[1] +
               BI_mod_Rborist_jw$model@trend.coef[2]*X1 +
               BI_mod_Rborist_jw$model@trend.coef[3]*X2 +
               BI_mod_Rborist_jw$model@trend.coef[4]*X1^2 +
               BI_mod_Rborist_jw$model@trend.coef[5]*X2^2 +
               BI_mod_Rborist_jw$model@trend.coef[6]*X1*X2) %>%
   mutate(predFixed = round(1+X1*16,0)) %>%
   mutate(minNode = round(1+X2*16,0))
```

Le J_w théorique maximal est ensuite évalué, ainsi que les hyperparamètres qui y sont associés.

```
BI_modelquad_Rborist_top <- BI_modelquad_Rborist[which.max(BI_modelquad_Rborist$Jw),
                                                 c("predFixed", "minNode")]
```

Il est ensuite possible de relancer un entraînement, comme précédemment, avec les paramètres optimisés, et en extraire les indicateurs de performances (J_w).

```
set.seed(1)
BI_fit_Rborist_best <- train(class ~ .,
                  method = "Rborist",
```

```
                  data = BI_lot_appr_opti,
                  trControl = tr_ctrl,
                  tuneGrid  = BI_modelquad_Rborist_top[c('predFixed', 'minNode')])

BI_fit_Rborist_best_resultats <- BI_fit_Rborist_best$results %>%
      mutate(Jw = Sens*CodBI_RatioSens + Spec*CodBI_RatioSpec - 1)
```

C.4 Évaluation des performances des modèles

L'étape finale est celle de l'évaluation des performances du modèle, ici présentée pour Rborist. Cette évaluation commence par l'extraction des valeurs réelles de comestibilité – c'est à dire des réponses attendues de la part de notre modèle – à partir du jeu de données d'évaluation, ainsi que leur conversion en valeurs booléennes, à fins de comparaison avec les valeurs qui seront prédites. Cette étape n'est évidemment pas nécessaire lors d'une classification multiclasse.

```
BI_evaluation <- BI_lot_evaluation %>%
   mutate(reference = as.factor(case_when(class == "toxique" ~ TRUE,
                                          class == "comestible" ~ FALSE)))
```

Les performances du modèle peuvent être évaluées, en termes d'efficience calculatoire, par son temps d'éxécution. Un moyen de mesurer le temps d'éxécution de n'importe quelle portion de code est de mesurer l'heure de début et de fin d'éxécution du code à l'aide de *Sys.time*, puis d'en mesurer la différence via la fonction *difftime*.

Le code exécuté correspond ici à l'entraînement du modèle (cf. *supra*) et à la prédiction sur le lot d'évaluation, enregistré dans un objet dédié.

```
chrono_debut <- Sys.time()
BI_fit_Rborist_final <- train(class ~ .,
                   method = 'Rborist',
                   data = BI_lot_appr_opti,
                   trControl = tr_ctrl,
                   tuneGrid  = BI_modelquad_Rborist_top[c('predFixed', 'minNode')])
BI_pred_Rborist_final <- predict(object = BI_fit_Rborist_final,
                                 newdata = BI_lot_evaluation)
chrono_fin <- Sys.time()
CodBI_temps_Rborist <- difftime(chrono_fin, chrono_debut) %>%
   as.numeric %>%
   round(.,2)
```

L'objet correspondant aux prédictions est ensuite comparé aux valeurs références que notre modèle devait prédire, ce qui permet notamment d'obtenir la matrice de confusion associée aux prédictions.

```
BI_CM_Rborist_final <- confusionMatrix(data = BI_pred_Rborist_final,
                                       reference = BI_lot_evaluation$class)
BI_CM_Rborist_final$table
```

Nous pouvons également extraire de cette comparaison des indicateurs de performances : sensibilité, spécificité, J_w dans le cas d'une classification binaire, kappa et indice de Rand pour la classification multiclasse, ainsi que le temps de calcul.

```
BI_resultats_Rborist <- BI_CM_Rborist_final$byClass %>%
   t(.) %>%
   as.data.frame(.) %>%
   select(c(Sensitivity, Specificity)) %>%
   mutate(Jw = Sensitivity*BI_RatioSens + Specificity*BI_RatioSpec - 1) %>%
   mutate(temps = BI_temps_Rborist)
```

Enfin, les principaux objets volumineux qui n'ont pas vocation à être exploités par la suite – c'est-à-dire les jeux de données, ainsi que les objets issus des fonctions *train* – sont retirés de l'environnement, qui est ensuite sauvegardé. Cette sauvegarde contient les graphiques, tableaux et valeurs d'intérêt, à fins d'insertion automatisée dans le fichier Rmarkdown qui constitue le corps de texte de cette thèse.

```
rm(dataset, BI_evaluation, BI_lot_appr_opti, BI_lot_evaluation,
   BI_fit_rpart_cp, BI_fit_Rborist, BI_fit_Rborist_best, BI_fit_Rborist_final)

save.image(file = "EKR-Champis-CodeSourceBi.RData")
```