

UNIVERSITÉ D'ANGERS

MASTER MATHÉMATIQUES ET APPLICATIONS
M1 DATA SCIENCE
ANNÉE ACADEMIQUE 2023-2024

TRAVAIL ENCADRÉ DE RECHERCHE

Calcul bayésien approché basé sur les Forêts Aléatoires : la méthode ABC-RF

Etudiant :

Mahamat MAHAMAT NOUR
BACHAR
Maxime CHARBONNEAU
Rayane JAFFAL

Tuteur Enseignant :

Charles RABIER

Engagement de non plagiat

Nous, soussignés

déclarons être pleinement conscients que le plagiat de documents ou d'une partie d'un document publiés sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée. En conséquence, nous nous engageons à citer toutes les sources que nous avons utilisées pour écrire ce rapport.

Table des matières

1	Introduction	1
1.1	Motivation	1
1.2	Objectifs	1
2	Calcul Bayésien Approché (ABC)	3
2.1	Statistique bayésienne	3
2.1.1	Loi a priori et loi a posteriori	3
2.1.2	Méthodes d'inférence bayésienne	3
2.2	ABC	4
2.2.1	Motivations	4
2.2.2	Principe et Algorithme des méthodes ABC	4
2.2.3	Calibration des méthodes ABC	5
2.2.4	Exemple	6
2.2.5	Limitations	9
3	Les Forêts Aléatoires (RF)	10
3.1	Les arbres CART	10
3.2	Les forêts aléatoires	11
3.2.1	Définition	11
3.2.2	Erreur Out-Of-Bag	12
3.2.3	Importance des variables	12
3.2.4	Ajustement des paramètres	13
4	Une nouvelle approche : ABC et Forêts Aléatoires	14
4.1	Choix du modèle	14
4.2	Une méthode de classification	15
4.3	ABC-RF	16
4.4	Illustration de la méthodologie ABC-RF	18
5	Application sur des données génétiques du riz d'Asie	20
5.1	Les réseau phylogénétiques	20
5.2	Analyse des données génétiques du riz par la méthode ABC-RF	22
5.3	Interprétation des résultats	22
5.3.1	Prédiction	23
5.3.2	Variables importantes	23
5.3.3	Optimisation du paramètre	24
6	Conclusion	26
	Annexes	28

Chapitre 1

Introduction

1.1 Motivation

Les méthodes bayésiennes, qui consistent à imposer des distributions de probabilités sur les paramètres d'un modèle statistique, s'avèrent très populaires en biologie. Elles peuvent, par exemple, permettre d'inférer des scénarios évolutifs responsables de la différenciation des génomes. L'émergence des méthodes dites ABC, pour Calcul Bayésien Approximé (Approximate Bayesian Computation), a connu un véritable engouement ces dernières années (e.g. Marin et al., 2012 ; Del Moral et al., 2012 ; Fearnhead et Prangle, 2012 ; Boitard et al. 2016 ; Saulnier et al. 2017 ; Jay et al. 2019). Ces méthodes ABC n'existeraient pas si la biologie n'avait pas introduit des problèmes de dimension si élevée. Typiquement, celles-ci sont employées lorsqu'il n'existe pas de formule analytique pour le calcul de vraisemblance, ou lorsque la vraisemblance demeure trop coûteuse en temps de calcul. Elles s'affranchissent du calcul de la vraisemblance, tout en restant fondées mathématiquement.

Des approches ABC-Forêts aléatoires (ABC-RF) ont été proposées par Pudlo et al (2015) [6], Raynal et al. (2019) [8], Estoup et al. (2018) [2] et ont déjà montré tout leur potentiel applicatif (e.g. Chapuis et al. 2020 ; Gilabert et al. 2023). ABC-RF se distingue de l'ABC classique car il ne prend pas en compte les distributions *a posteriori* traditionnellement utilisées. ABC-RF consiste à construire un classifieur sur la base du concept très célèbre des forêts aléatoires (Breiman 2001) et d'une table de référence constituée d'un grand nombre de statistiques résumées. Par la suite, il est également possible d'approximer la probabilité *a posteriori* du modèle sélectionné.

ABC-RF s'avère bien moins difficile qu'ABC à mettre en œuvre en terme de calibration. En effet, le choix de statistiques résumées n'est plus aussi crucial. Dans les méthodes ABC classiques, un mauvais choix de statistique résumées conduit à un très grand nombre de rejet (non proximité entre statistiques résumées observées et simulées), d'où la nécessité d'effectuer un nombre considérable de simulations afin de pouvoir approximer la distribution *a posteriori*. Au contraire, ABC-RF nécessite beaucoup moins de simulations à partir de la loi *a priori* car il ne repose pas sur une méthode d'acceptation/rejet : on peut ainsi remplir aisément une table de référence basée sur un très grand nombre de statistiques résumées.

1.2 Objectifs

Afin de pouvoir utiliser les approches ABC-RF, il est nécessaire de comprendre les principes sur lesquels celles-ci sont basés : les méthodes ABC et les Forêts aléatoires. Nous allons donc, dans un premier temps, nous familiariser avec le concept de l'ABC : ses avantages, comment calibrer ses algorithmes, ainsi que ses limitations. Nous allons ensuite nous attarder sur les Forêts aléatoires en décrivant, comme pour l'ABC, ses avantages ainsi que la calibration de ses algorithmes. Après la présentation de ces deux méthodes, nous pourrons enfin nous concentrer

sur notre centre d'intérêt principal : les méthodes ABC-RF. Outre les aspects théoriques de celles-ci, que nous présenterons en détail pour commencer, nous prendrons en main le package R abcrf afin de tester l'ABC-RF en cherchant à reproduire les résultats obtenus dans Pudlo et al (2015) [6] à des fins de recherche reproductible. Pour finir, nous illustrerons les résultats présentés dans la partie sur l'ABC-RF en étudiant à l'histoire évolutive du riz en Asie, notamment son processus de domestication. Les performances de ces méthodes dans le cadre de réseaux phylogénétique (graphe dirigé acyclique) n'étant pas encore présentes dans la littérature, nous accorderons notre attention à la présenterons cette histoire évolutive sous cette forme.

Chapitre 2

Calcul Bayésien Approxché (ABC)

2.1 Statistique bayésienne

L'inférence bayésienne est une méthode d'inférence statistique qui repose sur le théorème de Bayes à savoir :

Théorème 2.1.1. *de Bayes. On a la formule suivante :*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)}$$

Une version continue de ce résultat est la suivante :

$$\pi(\theta|y) = \frac{f(y|\theta)\pi(\theta)}{\int_{\Theta} f(y|\theta)\pi(\theta) d\theta}$$

2.1.1 Loi a priori et loi a posteriori

Définition 2.1.1. *On nomme loi a priori d'une variable aléatoire la connaissance que l'on a sur la distribution de la probabilité d'un événement avant d'observer les données. On nomme loi a posteriori la connaissance que l'on a sur la distribution de la probabilité d'un événement après avoir observé les données.*

Maintenant que nous avons défini les notions de loi a priori et de loi a posteriori, nous pouvons donc expliquer plus clairement la version continue du théorème de Bayes :

$$\underbrace{\pi(\boldsymbol{\theta} | y)}_{\text{a posteriori}} = \frac{\overbrace{f(y | \boldsymbol{\theta})}^{\text{vraisemblance a priori}} \overbrace{\pi(\boldsymbol{\theta})}^{\text{a priori}}}{\underbrace{\int_{\Theta} f(y | \boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}}_{\text{constante de normalisation}}}$$

où y sont les données observées.

2.1.2 Méthodes d'inférence bayésienne

La densité des observations s'avère, en pratique, très souvent difficile à déterminer. Afin de pouvoir résoudre ce problème, l'inférence bayésienne offre une panoplie de méthodes pour approximer la loi a posteriori. Parmi les plus populaires, on trouve :

Markov Chain Monte Carlo (MCMC) : Ces méthodes explorent l'espace dans lequel se trouve les paramètres en simulant des chaînes de Markov, permettant ainsi d'échantillonner la loi a posteriori.

Approximate Bayesian Computation (ABC) : Cette méthode approchée de calcul bayésien consiste à simuler des données à partir d'un modèle statistique, voir 2.2 pour plus de détails sur cette méthode.

Le choix de la méthode (MCMC vs ABC, cf 2.2.1) utilisée dépend du modèle et des données à disposition. L'inférence bayésienne permet de quantifier l'incertitude et d'intégrer des connaissances préexistantes dans l'analyse de données complexes.

2.2 ABC

2.2.1 Motivations

La statistique bayésienne s'appuie sur deux éléments clés pour calculer la loi a posteriori : la loi a priori et la vraisemblance. Cependant, dans certains cas, il peut être difficile voire impossible d'exprimer cette vraisemblance :

- pour des raisons mathématiques → la vraisemblance peut être difficile à formuler sous forme d'une fonction de paramètres.
- pour des raisons computationnelles → le calcul de la vraisemblance peut être trop coûteux en temps et ressources de calcul.

Les méthodes MCMC peuvent rencontrer des difficultés de convergence, en particulier pour des modèles complexes ou des données de grande dimension, modèles et données que l'on rencontre très souvent dans le domaine de la biologie. Ces limitations entravent l'application de l'inférence bayésienne classique à certains problèmes. C'est dans ce contexte que l'Approximate Bayesian Computation (ABC) prend tout son sens.

L'ABC propose une alternative à l'inférence bayésienne classique en contournant le problème de la vraisemblance inaccessible. Cette méthode a été initialement développée pour la génétique des populations (Tavaré et al. 1997), mais a depuis été appliquée à divers domaines, notamment la biologie, l'écologie, l'économie et la physique.

2.2.2 Principe et Algorithme des méthodes ABC

Les méthodes ABC sont également désignées sous le terme de méthodes sans vraisemblance (likelihood-free methods). Elles sont utilisées lorsque l'expression de la vraisemblance explicite est inaccessible, mais qu'un simulateur est disponible pour générer des données synthétiques. C'est ce simulateur ainsi que ses différents paramètres et les manières de les calibres que nous allons décrire dans cette section. Rappelons et introduisons tout d'abord quelques notations. Notons y l'échantillon des données observées, z les données simulées, θ le paramètre de la loi a priori π et posons \mathbf{D} un ensemble fini ou dénombrable dans lequel y prend ses valeurs. Regardons maintenant de plus près l'algorithme de l'ABC.

Algorithm 1 ABC

```

1: for  $i = 1$  to  $N_{ref}$  do
2:   repeat
3:     Générer  $\theta'$  à partir de la loi a priori  $\pi(\cdot)$ 
4:     Générer  $z$  à partir de la vraisemblance  $f(\cdot|\theta')$ 
5:   until  $\rho\{\eta(z), \eta(y)\} \leq \varepsilon$ 
6:    $\theta_i = \theta'$ 
7: end for
```

Les paramètres utilisés dans cet algorithme que nous n'avons pas déjà définis sont les suivants :

- N_{ref} , le nombre de θ' et de z qui seront simulés par l'algorithme.
- η , une statistique résumée définie sur D .
- ρ , une distance sur $\eta(D)$.
- $\epsilon > 0$, un niveau de tolérance.

Cet algorithme ABC estime la distribution a posteriori d'un paramètre en simulant les paramètres de la loi a priori $\pi(\cdot)$ pour produire des ensembles de données artificielles \mathbf{z} . Une comparaison de la similarité entre l'ensemble de données simulées et l'ensemble de données réelles observées est ensuite réalisée en calculant une distance entre z et y . Cette comparaison est effectuée grâce à une fonction ρ servant à calculer une distance, ou plus généralement une forme de discordance, entre les deux jeux données. Si la distance entre ceux-ci est nulle, nous pouvons alors sauvegarder la valeur de θ' car celle-ci décrit correctement la distribution a posteriori. C'est ici qu'une première problématique apparaît : il est virtuellement impossible de générer un ensemble de données artificielles z égal aux données observées y , la probabilité de générée de telles données est nulle. Afin de pallier ce problème, nous introduisons un nouvel élément dans l'algorithme : le paramètre de tolérance ϵ . Ce paramètre va nous permettre d'accepter des valeurs de θ' si les deux jeux de données sont suffisamment proches l'un de l'autre.

En pratique, à mesure que nous augmentons la taille de l'échantillon des données, il devient de plus en plus difficile d'obtenir des valeurs de ρ assez petites. Une solution naïve serait d'augmenter la valeur de ϵ , mais cela signifierait augmenter l'erreur de notre approximation, ce que nous ne voulons pas. Une meilleure solution, celle qui est utilisée dans l'algorithme 1, est d'utiliser une ou plusieurs statistiques résumées η et de calculer ensuite la distance entre les statistiques résumées des deux jeux de données. Il faut cependant être conscient qu'utiliser une (ou plusieurs) statistique(s) résumée(s) introduit une source supplémentaire d'erreur dans l'approximation ABC, des informations sur les données peuvent être perdues lors du calcul de celles-ci. Les $\theta_1, \theta_2, \dots, \theta_N$ ainsi produits par cet algorithme forment une approximation de la loi a posteriori.

L'idée derrière les méthodes ABC est qu'en utilisant des statistiques résumées η suffisamment représentatives avec une tolérance ϵ suffisamment petite, nous obtiendrons une approximation assez correcte de la loi a posteriori, à savoir [4] :

$$\pi_\epsilon(\theta|y) = \int \pi_\epsilon(\theta, z|y) dz \approx \pi(\theta|y).$$

2.2.3 Calibration des méthodes ABC

La statistique résumée : Plusieurs publications se sont déjà attardées sur la difficulté associée au choix de la ou des statistiques résumées $\eta(y)$, que l'on aimerait considérer comme une statistique quasi-suffisante, c'est à dire qui n'engendre quasiment pas de perte d'informations. Pour la majorité des problèmes concrets, il est impossible de trouver une ou des statistiques résumées suffisantes. Les statistiques résumées d'intérêt sont généralement déterminées par le problème en question et choisies après de nombreux essais. Le choix de celles-ci constitue une partie importante du travail de calibration de l'algorithme, elles ont un impact significatif sur celui-ci et un mauvais choix de peut rapidement mener à une forte baisse des performances de l'algorithme, comme nous allons le voir dans l'exemple la section suivante 2.2.4. Un point important à noter lors de la calibration de ce paramètre est qu'il ne faut pas négliger les corrélations entre les statistiques résumées lors de l'ajout ou du retrait de celles-ci de l'algorithme.

La tolérance : Le choix du niveau de tolérance ϵ dépend principalement de la puissance de calcul disponible et de la précision des résultats souhaitée : des valeurs plus petites de ϵ sont associées à des coûts de calcul plus élevés, plus d'essais seront nécessaires afin de trouver

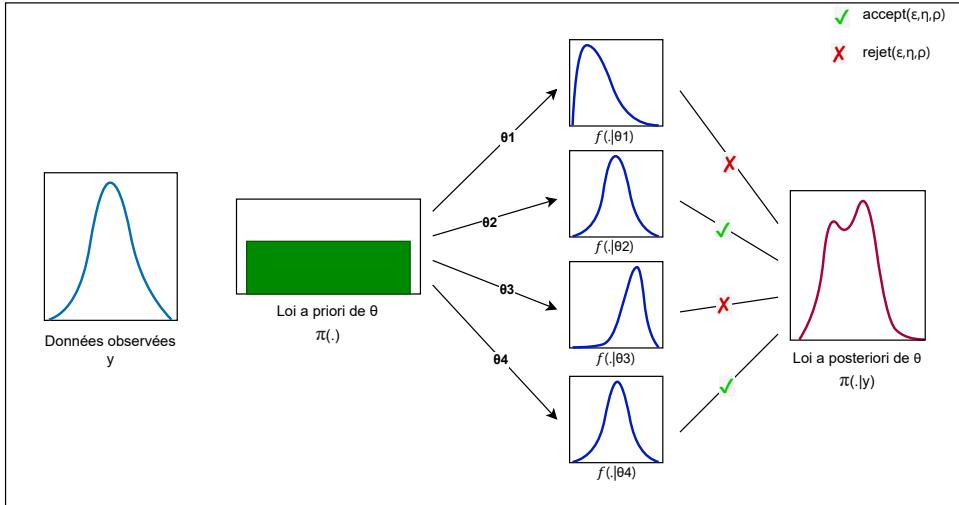


FIGURE 2.1 – Représentation graphique du fonctionnement des algorithmes ABC. Nous prélevons un ensemble de valeurs de θ à partir de la distribution a priori. Chaque valeur est ensuite transmise au simulateur (le modèle), qui génère des ensembles de données synthétiques (simulées). Nous comparons ensuite les données simulées avec celles observées. Dans cet exemple, seule les valeurs θ_2 et θ_4 ont pu générer un ensemble de données synthétique suffisamment proche des données observées, les valeurs θ_1 et θ_3 ont donc été rejetées. L'accumulation des θ acceptées produit une approximation de la loi a posteriori.[5]

des données simulées assez proches des données observées. La pratique standard consiste à sélectionner ϵ comme un petit pourcentage des distances simulées. Au contraire, plus la valeur de ϵ est grande, plus nous serons tolérants quant à la proximité entre $\eta(z)$ et $\eta(y)$, moins d'essais seront donc nécessaires. En général, pour un problème donné, une valeur plus grande de implique une approximation plus grossière de la loi a posteriori. [4]

La distance : La distance est un élément crucial des méthodes ABC car elle détermine comment quantifier la similarité entre les données simulées et observées. Les choix de distance les plus populaires dans la littérature sont la distance euclidienne et la distance quadratique. Nous utiliserons dans l'exemple 2.2.4 la distance euclidienne. D'autres distances telles que la norme 1, la norme infini, la divergence de Kullback–Leibler (KL) et la distance de Mahalanobis sont également utilisées. [5] Le choix de la distance dépend de plusieurs facteurs : la nature des données, leur dimension, les propriétés du modèle, ainsi que l'objectif de l'inférence. [1]

2.2.4 Exemple

Afin d'illustrer la méthode ABC, nous allons maintenant présenter un exemple concret d'application de cette méthode sur les séries temporelles à moyenne mobile (MA).

Définition 2.2.1. *Le processus MA(q) est un processus stochastique $(y_k)_{k \in \mathbb{N}^*}$ défini par :*

$$y_k = u_k + \sum_{i=1}^q \theta_i u_{k-i}$$

où $(u_k)_{k \in \mathbb{Z}}$ est une séquence iid de lois normales centrées réduites $N(0, 1)$.

Bien que l'analyse bayésienne puisse gérer l'inférence de loi a posteriori dans le cas des moyennes mobiles, les calculs et les calibrations de l'algorithme ABC restent complexes à effectuer et ce n'est pas l'objectif de cette section. Nous allons imposer la condition d'identifiabilité

sur ce modèle que les racines du polynôme :

$$Q(x) = 1 - \sum_{i=1}^q \theta_i x^i$$

sont toutes à l'extérieur du cercle unité dans le plan complexe. Une distribution a priori simple des paramètres est donc la distribution uniforme sur les régions correspondante des θ_i , particulièrement lorsque q est petit.

Afin de simplifier la visualisation des résultats et la calibration des algorithmes, nous allons traiter dans cet exemple le cas $q = 2$. L'expression de la moyenne mobile est donc donnée par :

$$y_k = u_k + \theta_1 u_{k-1} + \theta_2 u_{k-2}.$$

Après calculs, on obtient comme distribution a priori pour les deux paramètres θ_1 et θ_2 le triangle :

$$-2 < \theta_1 < 2, \quad \theta_1 + \theta_2 > -1, \quad \theta_1 - \theta_2 < 1.$$

Bien que la loi a priori sur θ soit simple, et malgré la nature gaussienne des variables aléatoires, la vraisemblance associée à une série $(y_k)_{1 \leq k \leq n}$ est reste tout de même complexe en raison de la nécessité d'intégrer les $u_{-q+1}, \dots, u_{-1}, u_0$.

Une itération de l'algorithme ABC dans ce cadre nécessite simplement :

1. de simuler θ_1 et θ_2 sur triangle défini précédemment,
2. de générer une séquence iid $(u_k)_{-2 < k \leq n}$,
3. et de produire une série simulée $(z_k)_{1 \leq k \leq n}$.

Pour pouvoir illustrer les résultats de l'algorithme, nous avons simulé une série de longueur 100 en utilisant les paramètres $(\theta_1, \theta_2) = (0.6, 0.2)$. L'objectif est de retrouver ces deux paramètres en utilisant notre algorithme ABC.

Pour les paramètres ρ et η nous allons ici utiliser la distance quadratique et les 2 premières autocovariances :

$$\tau_1 = \sum_{k=2}^n y_k y_{k-1}, \quad \tau_2 = \sum_{k=3}^n y_k y_{k-2}$$

Le graphique 2.2 illustre comment la distribution des paramètres conservés par l'algorithme ABC s'éloigne de la vraie densité a posteriori. Nous avons choisi le niveau de tolérance ϵ de telle sorte que 0.1% des $N_{ref} = 10^6$ jeux de données que nous avons simulés pour cet exemple soient acceptés, nous allons donc ici accepter les 100 jeux de données les plus proches du jeu de données simulé avec $(\theta_1, \theta_2) = (0.6, 0.2)$. On remarque que les paramètres des données qui ont été acceptées ne se concentrent pas parfaitement autour de $(0.6, 0.2)$. Diminuer ϵ conduirait à une meilleure concentration de ceux-ci, mais au détriment de la taille de l'échantillon résultant après filtrage par ϵ ou à un coût de calcul plus élevé si l'on souhaite garder le même nombre de jeux de données acceptés car il faudrait alors augmenter le nombre de simulations.

Pour obtenir le graphique 2.3 nous avons utilisé comme statistique descriptive la moyenne des jeux de données simulés et comme distance le distance euclidienne. Nous pouvons très clairement voir que l'un de ces deux choix, ou les deux, était très peu approprié à notre problème. Bien que les prédictions choisies soient les 100 meilleures prédictions effectuées, la grande majorité d'entre elles sont très éloignées des vrai valeurs des paramètres. Ce graphique nous montre bien qu'un mauvais choix de statistiques résumées et de distance peut conduire à un très gros impact sur les performances de l'algorithme.

Tous les codes pour obtenir ces graphiques ainsi que d'autres graphiques représentant les variations des performances de l'algorithme en fonction des choix des paramètres peuvent être retrouvés dans la section 6 de l'annexe.

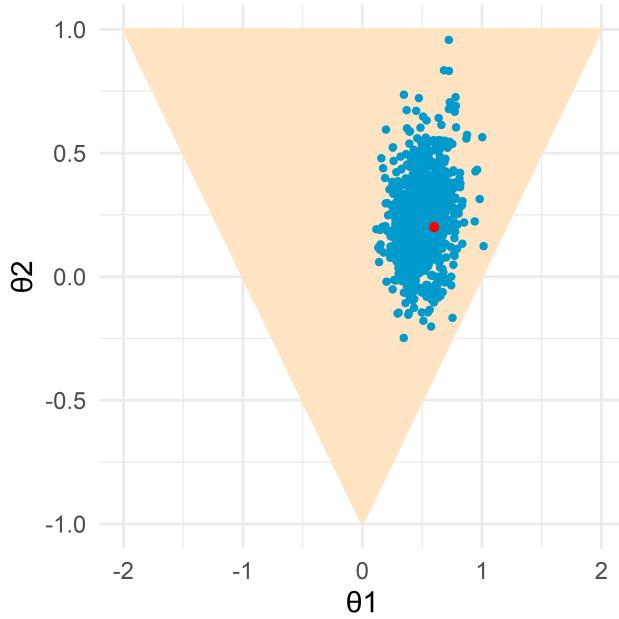


FIGURE 2.2 – Comparaison des vraies valeurs de θ_1 et θ_2 (en rouge sur le graphique) avec l'approximation ces valeurs (en bleu) par ABC, avec comme statistiques résumées les deux premières auto-covariances et comme distance la distance quadratique. Le triangle représente l'ensemble de valeurs acceptables de θ_1 et θ_2 .

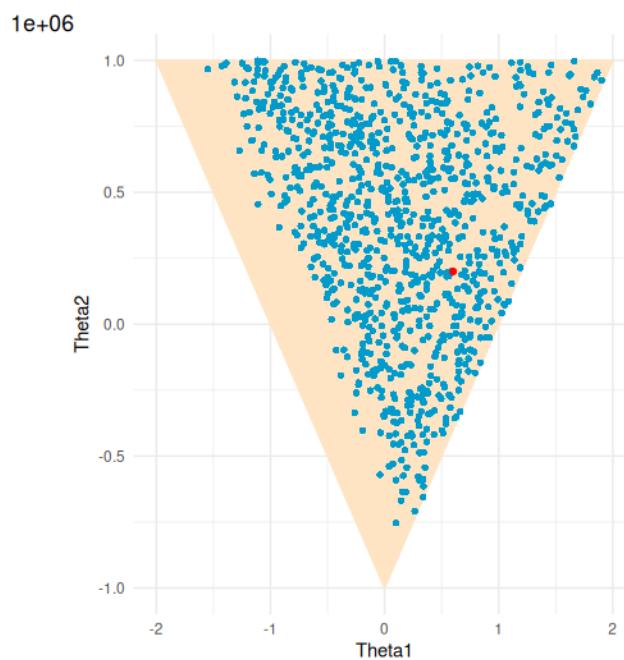


FIGURE 2.3 – Comparaison des vraies valeurs de θ_1 et θ_2 (en rouge sur le graphique) avec l'approximation ces valeurs (en bleu) par ABC avec comme statistique résumée la moyenne des jeux de données simulées et comme distance la distance euclidienne. Le triangle représente l'ensemble de valeurs acceptables de θ_1 et θ_2 .

2.2.5 Limitations

1. Perte de précision : L'approximation introduite par ABC peut entraîner une perte de précision par rapport à l'inférence bayésienne classique. Comme nous avons pu le voir dans l'exemple précédent, bien que les valeurs retournées par l'algorithme ABC soient concentrées autour des vrai valeurs des paramètres, il est important de noter une certaine dispersion de ces approximations. L'impact sur la précision dépend de plusieurs facteurs :

- La complexité de la distribution a posteriori. Des distributions a posteriori complexes peuvent grandement impacter les performances de l'algorithme et la qualité des prédictions.
- La qualité des simulations. La précision des simulations est essentielle lors de l'étape de la comparaison avec les vrai données afin d'éviter des prédition avec une variance trop grande ou afin d'éviter une temps de calcul trop long pour que l'algorithme soit d'une quelconque utilité.
- Le choix de la distance. Une distance mal choisie peut introduire un biais dans les résultats.
- Le nombre de simulations. Un nombre de simulation insuffisant peut limiter la précision de l'approximation si le niveau de tolérance est trop bas.

2. Calibrage de l'algorithme : Le calibrage des simulations vise à garantir que les simulations reflètent correctement la distribution a posteriori. Nous avons pu constater dans l'exemple précédent que l'algorithme est très sensible aux variations des statistiques résumées, de la distance ou du niveau de tolérance. Le choix de ces différents paramètres fait parti d'une des difficultés majeurs des algorithmes ABC. L'exemple que nous avons pris étant simple, la recherche et le choix de ceux-ci n'a pas été une très grande difficulté mais pour les problèmes les plus complexes, très souvent rencontrés dans les cas pratiques, il est nécessaire d'avoir par exemple des statistiques résumées beaucoup plus nombreuses et beaucoup plus complexes, ce qui augmente grandement le temps nécessaire à rechercher les bonnes statistiques à choisir ainsi que de la bonne distance à choisir afin de pouvoir efficacement les comparer.

3. Limites computationnelles : Bien que généralement plus efficace que MCMC, ABC reste gourmand en ressources computationnelles pour des modèles complexes et des ensembles de données volumineux. La réduction du temps de calcul des algorithmes ABC est un domaine de recherche actif.

Pour conclure, l'ABC est un outil de prédiction de loi a posteriori puissant mais il n'est pas une solution miracle. Le choix de l'ABC doit être justifié par les caractéristiques du modèle et des données afin de s'assurer que cet algorithme soit adapté au problème posé. Une attention particulière doit être portée aux aspects délicats que nous avons mentionnés tout au long de cette partie. [4]

Chapitre 3

Les Forêts Aléatoires (RF)

Nous avons expliqué, dans un premier temps, les méthodes ABC afin de pouvoir comprendre les méthodes ABC-RF. Nous allons maintenant pouvoir nous attarder sur la deuxième méthodes nécessaire afin de pouvoir travailler sur celles-ci : les Forêts Aléatoires (RF).

3.1 Les arbres CART

Afin de pouvoir décrire et comprendre les RF, nous allons avoir besoin de détailler la méthode sur laquelle ceux-ci reposent : les arbres de classification et de régression ou arbres CART.

L'algorithme des arbres CART produit un arbre binaire, c'est à dire un arbre ayant deux branches à chaque nœuds, qui établit des règles d'allocation, nommées étiquettes, sur les nœuds internes de l'arbre permettant de diriger des valeurs d'entrée X vers une branche de l'arbre ou une autre en fonction de certaines conditions afin de les classifier ou de prédire des valeurs Y lorsque ces valeurs X arrivent dans nœud terminal, appelé feuille. À un nœud interne donné, la règle binaire compare une covariable X_j , déterminée lors de la création de l'arbre, avec une limite t . La branche émergeant à gauche de ce nœud interne est définie par $X_j < t$ et la branche émergeant à droite est définie par $X_j \geq t$. Prédire la valeur de Y étant donné une observation X implique de suivre un chemin à partir de la racine de l'arbre qui est défini en appliquant ces règles binaires. Le résultat de la prédiction est la valeur trouvée à la feuille atteinte à la fin du chemin.[6]

Pour trouver la meilleure variable X^j afin d'effectuer la meilleur division $\{X^j \leq d\} \cup \{X^j > d\}$ à chaque nœud de l'arbre où $j \in \{1, \dots, p\}$ et $d \in \mathbb{R}$, nous allons minimiser l'un des critères suivant :

- en classification, avec un ensemble des classes $\{1, \dots, C\}$, on définit l'impureté des noeuds fils par le biais de l'indice de Gini. L'indice de Gini d'un noeud t est défini par $\Phi(t) = \sum_{c=1}^C \hat{p}_t^c(1 - \hat{p}_t^c)$ où \hat{p}_t^c est la proportion d'observations de la classe c dans le noeud t . On est alors conduit, pour tout noeud t et toute séparation admissible, on cherche à maximiser :

$$\Phi(t) - \left(\frac{\#t_L}{\#t} \Phi(t_L) + \frac{\#t_R}{\#t} \Phi(t_R) \right);$$

- en régression, on cherche à minimiser la variance intra-groupes entre deux noeuds fils t_L et t_R résultants de la découpe d'un noeud t . La variance d'un noeud t est définie par $V(t) = \frac{1}{\#t} \sum_{i:x_i \in t} (y_i - \bar{y}_t)^2$ où \bar{y}_t et $\#t$ sont la moyenne et le nombre d'observations présentes dans le noeud t . On cherche donc à maximiser :

$$V(t) - \left(\frac{\#t_L}{\#t} V(t_L) + \frac{\#t_R}{\#t} V(t_R) \right).$$

En régression, on recherche donc les découpes qui tendent à diminuer la variance des noeuds obtenus. En classification, on cherche à diminuer la fonction de pureté de Gini, et donc à augmenter l'homogénéité des noeuds obtenus. Un noeud est parfaitement homogène s'il ne contient que des observations de la même classe. Dans le cas d'une variable explicative X^j catégorielle, ce qui précède demeure valide sauf que, dans ce cas, la coupure est de la forme : $\{X^j \in d\} \cup \{X^j \in \bar{d}\}$ où $j \in \{1, \dots, p\}$ et d et \bar{d} sont non vides et constituent une partition de l'ensemble des modalités de la variable X^j .

La recherche de la meilleure coupure se fait parmi toutes les variables à chaque fois. Ainsi, une variable peut être utilisée dans plusieurs coupures de l'arbre. Dans la version aléatoire de l'algorithme CART seul un sous-ensemble de covariables de taille n_{try} est choisi aléatoirement et est considéré à chaque nœud de l'arbre.

3.2 Les forêts aléatoires

Maintenant que nous avons fini de définir ce que sont les arbres CART, nous allons pouvoir passer à l'étude des Forêts aléatoires.

3.2.1 Définition

Le principe général des forêts aléatoires est d'agréger une collection d'arbres de décision aléatoires. L'idée est de mettre en commun un ensemble de prédicteurs pour profiter d'une plus grande exploration de l'espace et ainsi des meilleures performances prédictives.

Définition 3.2.1. Soient $(\hat{h}(\cdot, \Theta_1), \dots, \hat{h}(\cdot, \Theta_q))$ une collection d'arbres de décision, avec $\Theta_1, \dots, \Theta_q$ q variables aléatoires i.i.d. indépendantes de L_n , l'échantillon d'apprentissage. Le prédicteur des forêts aléatoires \hat{h}_{RF} est obtenu en agrégeant cette collection d'arbres aléatoires. L'agrégation se fait de la façon suivante :

- En régression, $\hat{h}_{RF}(x) = \frac{1}{q} \sum_{l=1}^q \hat{h}(x, \Theta_l)$, la moyenne des prédictions individuelles des arbres ;
- En classification, $\hat{h}_{RF}(x) = \underset{1 \leq c \leq C}{\operatorname{argmax}} \sum_{l=1}^q \mathbb{I}_{\{\hat{h}(x, \Theta_l)=c\}}$, le vote majoritaire parmi les prédictions individuelles des arbres.

Dans l'algorithme des forêts aléatoires, chaque arbre de décision est construit grâce aux données d'entraînement, mais avec une petite variation cruciale : au lieu d'utiliser l'ensemble complet de données, chaque arbre est entraîné sur un échantillon bootstrap de la base de données initiale. Ce processus, connu sous le nom de "bootstrapping", implique la création d'échantillons bootstrap en tirant aléatoirement, avec remise, des observations de la base de données d'entraînement. Chaque observation (X_i, Y_i) de la base de données d'entraînement a une probabilité de $1/n$ d'être sélectionnée à chaque tirage, où n est la taille de l'échantillon. Ces échantillons bootstrap servent ensuite à entraîner chaque arbre de décision de la forêt.

Le processus de bootstrap permet d'introduire de la variabilité dans les échantillons d'entraînement utilisés pour chaque arbre, ce qui contribue à réduire la corrélation entre les arbres individuels. Cette diversité accrue est cruciale pour améliorer les performances prédictives globales de la forêt aléatoire. De plus, lors de la construction de chaque arbre, seul un sous-ensemble aléatoire de covariables de taille n_{try} est considéré à chaque nœud, ce qui ajoute une autre dimension de variabilité et permet d'augmenter la robustesse de l'algorithme.

Sur **R**, il est possible de construire une forêt aléatoire grâce à la fonction **randomForest()** implémentée dans le package **randomForest**. En ajustant les paramètres de cette fonction, tels que le nombre N_{tree} d'arbres dans la forêt, le nombre n_{try} de covariables échantillonées à

Algorithm 2 Forêts aléatoires

- 1: **for** $i = 1$ to N_{tree} **do**
- 2: Tirer un échantillon bootstrap de taille N_{boot}
- 3: Crée un arbre de décision CART
- 4: **for** $b = 1$ to B **do** (avec B le nombre des noeuds)
- 5: Sélectionner n_{try} variables au hasard
- 6: Déterminer la meilleure scission parmi ces prédicteurs
- 7: **end for**
- 8: **end for**
- 9: Prédire de nouvelles données en agrégeant les prédictions des N_{tree} arbres

chaque nœud de l'arbre CART aléatoire, et la taille N_{boot} des sous-échantillons bootstrap, les forêts aléatoires peuvent être adaptées à différentes tâches d'apprentissage et types de données, offrant ainsi une méthode flexible et puissante d'apprentissage statistique. En agrégeant les prédictions de ces arbres aléatoires, les forêts aléatoires bénéficient de la diversité induite par le bootstrap et de l'agrégation des prédictions pour améliorer les performances prédictives, faisant d'elles une approche très efficace dans de nombreux domaines d'application.

3.2.2 Erreur Out-Of-Bag

L'estimation de l'erreur de prédiction des forêts aléatoires est donné par l'erreur OOB dans la sortie de la fonction **randomForest**, où "OOB" signifie "Out Of Bag" et doit se comprendre comme "en dehors du Bootstrap". Cet estimateur est de type validation croisée et permet de se dispenser d'avoir recours à un échantillon test. Il utilise les observations qui n'ont pas été sélectionnées dans un échantillon bootstrap comme des données tests. Pour calculer l'erreur OOB, on agrège uniquement les prédicteurs construits sur des échantillons bootstrap ne contenant pas Y_i , ce qui fournit une prédiction \hat{Y}_i pour la sortie de la $i^{\text{ème}}$ observation. L'erreur OOB est :

- l'erreur quadratique moyenne en régression, $\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$
- la proportion de d'observation mal classées en classification, $\frac{1}{n} \sum_{i=1}^n \mathbb{I}_{\{Y_i \neq \hat{Y}_i\}}$

3.2.3 Importance des variables

Dans les forêts aléatoires, il est possible de construire une hiérarchie des variables explicatives basée sur une quantification de l'importance des effets de la variable sur la réponse. Cette quantification est donnée par l'accroissement moyen de l'erreur d'un arbre dans la forêt lorsque les valeurs observées de cette variable sont permutées au hasard dans les échantillons OOB.

- Posons $j \in \{1, \dots, p\}$ et calculons $VI(X^j)$ l'indice d'importance de la variable X^j .
- On considère un échantillon bootstrap $L_n^{\Theta_l}$ et l'échantillon OOB_l associé qui représente l'ensemble des observations n'apparaissant pas dans $L_n^{\Theta_l}$.
 - On calcule ensuite $errOOB_l$, l'erreur commise sur OOB_l par l'arbre construit sur $L_n^{\Theta_l}$.
 - On permute ensuite aléatoirement les valeurs de la $j^{\text{ème}}$ variable dans l'échantillon OOB_l , ce qui nous donne un échantillon perturbé, noté \widetilde{OOB}_l^j .
 - On calcule enfin \widetilde{errOOB}_l^j , l'erreur sur l'échantillon \widetilde{OOB}_l^j .
 - On effectue ces opérations pour tous les échantillons bootstrap. L'importance de la variable X^j , $VI(X^j)$, est alors définie par la différence entre l'erreur moyenne d'un arbre sur l'échantillon OOB perturbé et celle sur l'échantillon OOB :

$$VI(X^j) = \frac{1}{q} \sum_{l=1}^q \left(\widetilde{errOOB}_l^j - errOOB_l \right).$$

Plus l'augmentation de l'erreur engendrée par des permutations aléatoires de la $j^{\text{ème}}$ variable explicative est forte, plus la variable est importante. Et inversement, si les permutations n'affecte pas l'erreur ou ont peu d'effet, la variable est considérer comme très peu importante, (VI peut être légèrement négative). On peut retrouver l'importance VI des variables à l'aide de la fonction **varImpPlot()** du package **randomForest** sur **R**. Cette fonction renvoie directement un graphe qui montre l'importance des variables VI classées par ordre décroissant d'importance.

3.2.4 Ajustement des paramètres

Le nombre d'arbres : **ntree**

Pour la méthode RF le paramètre $ntree$ (le nombre d'arbres CART dans une forêt aléatoire) n'est pas le plus crucial. En règle générale, plus le nombre d'arbres est grand, meilleur sera la forêt aléatoire. Dans le package **randomForest**, la fonction **plot()** trace l'évolution de l'erreur OOB en fonction du nombre d'arbres dans la forêt, ce qui permet de déterminer le nombre d'arbres dès lequel l'ajout de nouveaux arbres dans la forêt n'apporte pas d'amélioration en terme de qualité de prédiction. L'erreur OOB varie très peu dès lors que le nombre d'arbres est suffisamment grand. La valeur par défaut de $ntree$ dans la fonction **randomForest()** responsable de la construction des forêts aléatoires dans **R** est 500 ce qui est généralement dans la plupart des situations.

Le nombre des variables choisies à chaque noeud : **mtry**

Le paramètre le plus important de la fonction **randomForest()** est le paramètre $mtry$. Choisir différentes valeurs de ce paramètre peut conduire à des performances en prédiction très différentes. Le paramètre $mtry$ permet de déterminer le nombre de variables X^j choisies aléatoirement afin de construire chaque noeud. Ce paramètre dépend grandement du nombre totale des variables explicatives p disponibles. La valeur par défaut de ce paramètre dans la fonction **randomForest** est $mtry = p/3$ pour les forêts aléatoires de destinées à faire de la régression et est égale à \sqrt{p} pour celles destinées à faire de la classification. [3]

Chapitre 4

Une nouvelle approche : ABC et Forêts Aléatoires

Maintenant que nous avons fini de présenter les méthodes ABC et les Forêts Aléatoires, nous allons pouvoir passer à notre principal sujet d'intérêt : les méthodes ABC-RF.

4.1 Choix du modèle

En plus de l'inférence des paramètres de modèles, la méthode ABC permet de prédire le modèle \mathcal{M} associé à la loi a priori $\pi(\mathcal{M} = m)$ où $m = 1, \dots, M$ sont les différents modèles possibles pour celle-ci. Cette méthode permet aussi de prédire une loi a priori sur les paramètres conditionnelle à la valeur m de l'indice de modèle, $\pi_m(\theta_m)$, définie sur l'espace des paramètres Θ_m . L'algorithme du choix du modèle avec ABC suit le même principe que l'algorithme 1, où $\eta(z) = (\eta_1(z), \dots, \eta_M(z))$ est la concaténation des statistiques résumées utilisées pour toutes les modèles.

Algorithm 3 ABC - Choix du modèle

```
1: for  $i = 1$  to  $N_{ref}$  do
2:   (A)
3:   repeat
4:     Générer  $m^{(i)}$  selon la loi a priori  $\pi(\mathcal{M} = m)$ 
5:     Générer  $\theta'_{m^{(i)}}$  selon la loi a priori  $\pi_{m^{(i)}}(\theta_m)$ 
6:     Générer  $z$  à partir du modèle  $f_m(z|\theta_{m^{(i)}})$ 
7:   until  $\rho\{\eta(z), \eta(y)\} < \epsilon$ 
8:   (B)
9:   Définir  $m^{(i)} = m$  et  $\theta'_{m^{(i)}} = \theta_m$ 
10: end for
```

L'estimation de la probabilité a posteriori $\pi(\mathcal{M} = m | \eta(y))$ par la méthode ABC est alors la moyenne d'acceptation du modèle m :

$$\frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\{m^{(i)}=m\}}.$$

Cela correspond également à la proportion d'ensembles de données simulés qui sont plus proches des données y que la tolérance ϵ . [4]

Les difficultés associées à la calibration de la méthode ABC, qui entravent l'obtention d'outils fiables pour la sélection du modèle, nous conduisent à opter pour une approche de classification au lieu de l'estimation de la probabilité a posteriori $\pi(\mathcal{M} = m \mid \eta(y))$, **laquelle est très sensible aux calibrations réalisées par la méthode ABC**, comme nous avons pu le voir dans le chapitre 2. L'objectif de la méthode de classification est alors de prédire l'indice du modèle m correspondant le mieux aux informations que nous avons sur le vecteur des statistiques résumées $\eta(z)$.

Une façon pratique d'évaluer les performances d'un algorithme de choix de modèle avec ABC associé à un classifieur donnée et à un ensemble donné de statistiques résumées est de vérifier s'il fournit une meilleure réponse que d'autres algorithmes ayant le même objectif mais utilisant d'autres outils. L'objectif est de se rapprocher du classificateur bayésien, qui, pour des données y observées, sélectionne le modèle ayant la plus grande probabilité a posteriori $\pi(\mathcal{M} = m \mid \eta(y))$. Il est bien connu que le classificateur bayésien minimise la perte ou l'erreur intégrée de 0-1. Dans le cadre des méthodes ABC, nous appelons la perte intégrée le taux d'erreur a priori, car il fournit une indication de la qualité globale d'un classificateur \hat{m} donné sur tout l'espace pondéré par la loi a priori. Ce taux est la valeur attendue de l'erreur de mauvaise classification sur la loi a priori hiérarchique :

$$\sum_m \pi(m) \int \mathbb{I}_{\{\hat{m}(\eta(z)) \neq m\}} f(z|\theta, m) \pi(\theta|m) dz d\theta$$

Il peut être évalué à partir de simulations $(\theta, m, \eta(z))$ tirées à l'étape (A) de l'algorithme 3, indépendamment de la table de référence. Les classificateurs peuvent donc être comparés via cette échelle d'erreur : la paire qui minimise le taux d'erreur a priori obtient la meilleure approximation du classificateur bayésien idéal. En ce sens, elle se rapproche le plus de la décision que nous prendrions si nous étions en mesure de calculer le vrai $\pi(\mathcal{M} = m \mid \eta(y))$. [6]

4.2 Une méthode de classification

La classification constitue un paradigme d'apprentissage supervisé, où un algorithme apprend des données fournies en entrée, puis applique ce qu'il a appris pour classer de nouvelles observations. De nombreux algorithmes d'apprentissage impliquent le réglage de paramètres qui doivent être déterminés avec soin afin d'obtenir des résultats satisfaisants. Généralement, les performances prédictives des classificateurs sont évaluées sur des données n'ayant pas servies lors de l'étape d'apprentissage. Dans notre contexte, pour évaluer la performance d'un algorithme ABC de sélection de modèle qui utilise un classifieur, on compare s'il minimise le taux d'erreur de prédiction a priori (calculé à partir d'une table de référence remplie grâce à des simulations générées à l'aide de la loi a priori). Nous cherchons un classifieur qui puisse gérer un nombre arbitraire de statistiques résumées et extraire l'information maximale de la table de référence obtenue à la première étape (A) de l'algorithme 3.

Afin d'illustrer ceci, nous allons reproduire les résultats de Pudlo et al [6] sur la comparaison des erreurs de prédiction des différentes méthodes de classification (les forêts aléatoires, la méthode des k plus proches voisins, la méthode bayésienne naïve, la régression logistique et l'analyse linéaire discriminante) sur un exemple de séries temporelles. Étant donnée une série temporelle (y_t) de longueur $n = 100$, nous comparons les taux d'erreurs de classification des modèles de moyenne mobile d'ordre 1 et 2, MA(1) et MA(2), respectivement définis par les équations :

$$y_t = u_t - \theta_1 u_{t-1} \quad \text{et} \quad y_t = u_t - \theta_1 u_{t-1} - \theta_2 u_{t-2} \quad ; \quad u_t \sim N(0, 1) \quad .$$

Pour ces deux modèles, les loi a priori des θ s, sous les conditions que nous avons posées dans la section 2.2.4, sont des distributions uniformes sur les domaines de stationnarité :

- pour MA(1), le paramètre unique θ_1 est tiré uniformément du segment $(-1, 1)$;
- pour MA(2), le couple (θ_1, θ_2) est tiré uniformément sur le triangle défini par $-2 < \theta_1 < 2$, $\theta_1 + \theta_2 > 1$, et $\theta_1 - \theta_2 < 1$.

Nous allons considérer ici comme statistiques résumées les sept premières autocorrélations, une ensemble de statistiques résumées qui donne une table de référence ABC de taille $N_{\text{ref}} = 10^4$ avec sept covariables.

La table 4.1 établit une comparaison entre les performances de différents classifieurs en considérant les sept premières autocorrélations comme statistiques résumées. Nous avons utilisé 80% de la table de référence $(m, \eta(z))$ tiré de la première étape de l'algorithme ABC comme données d'entraînement et 20% de cette même table comme données test. **On peut remarquer que parmi les méthodes de classifications appliquées, la méthode qui atteint le minimum taux d'erreur a priori est les forêts aléatoires avec taux d'erreur d'environ 10% sur cette exemple.** [6]

Les codes utilisés pour obtenir cette tables sont disponibles dans la partie 6 de l'annexe.

Taux d'erreur de classification: MA(1) vs MA(2)	
classifieur	erreur
Random Forest	10.025
knn (k = 50)	10.500
knn (k = 100)	10.500
Naive Bayes	11.675
Logistic Regression	13.600
LDA	13.975

FIGURE 4.1 – Comparaison du taux d'erreur de classification des différentes méthodes de classification

4.3 ABC-RF

La méthode de classification des forêts aléatoires, comme introduit précédemment, permet de gérer un nombre arbitraire de statistiques et d'extraire le maximum d'informations de la table de référence. Elle s'est avérée être insensible à la fois aux fortes corrélations entre les covariables (ici, les statistiques résumées) et à la présence de variables bruits, même en grand nombre. Ce type de robustesse justifie l'adoption d'un algorithme RF pour inférer, à partir d'une table de référence générée grâce à une ABC, la sélection de modèles.

Algorithm 4 ABC-RF

- 1: Générer une table de référence comprenant N_{ref} simulations $\{m, \eta(z)\}$ à partir de $\pi(m)\pi(\theta|m)f(z|m, \theta)$.
 - 2: Construire N_{tree} arbres CART aléatoires permettant de prédire m en utilisant $\eta(z)$.
 - 3: **for** $b = 1$ à N_{tree} **do**
 - 4: Tirer un échantillon bootstrap de taille N_{boot} à partir de la table de référence.
 - 5: Faire croître un arbre CART randomisé T_b
 - 6: **end for**
 - 7: Déterminer les indices prédits pour $\eta(y)$ et les arbres $\{T_b; b = 1, \dots, N_{\text{tree}}\}$.
 - 8: Sélectionner l'indice des $\eta(y)$ en fonction d'un vote majoritaire parmi les modèles prédits.
-

La RF est entraînée sur les simulations produites par la première étape de l'algorithme de choix du modèle 3, ce qui permet de constituer la table de référence composée de N_{ref} éléments, chacun étant composé de l'indice du modèle m et d'un vecteur $\eta(z)$ composé de d statistiques résumées. Une fois que le modèle d'indice m est sélectionné, nous choisissons d'approximer $\pi(\mathcal{M} = m | \eta(y))$ par une autre RF, obtenue à partir de la régression de la probabilité d'erreur sur les mêmes covariables, comme expliqué dans la section suivante.

$$\begin{pmatrix} m^{(1)} & \eta_1(z^{(1)}) & \eta_2(z^{(1)}) & \cdots & \eta_d(z^{(1)}) \\ m^{(2)} & \eta_1(z^{(2)}) & \eta_2(z^{(2)}) & \cdots & \eta_d(z^{(2)}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m^{(N_{\text{ref}})} & \eta_1(z^{(N_{\text{ref}})}) & \eta_2(z^{(N_{\text{ref}})}) & \cdots & \eta_d(z^{(N_{\text{ref}})}) \end{pmatrix}$$

Par suite un classifieur $\hat{m}(.)$ est construit pour inférer l'indice du modèle. Ce classifieur est ensuite appliquée aux données observées $\eta(y)$ afin de prédire l'indice du modèle.

Le résultat du calcul de RF appliqué à un ensemble de données observés donné est un vote de classification pour chaque modèle, qui représente le nombre de fois qu'un modèle est sélectionné dans une forêt de N_{tree} arbres. Le modèle avec le vote de classification le plus élevé correspond au modèle le mieux adapté à l'ensemble de données observés. Ainsi le classifieur $\hat{m}(.)$ est suffisamment bon pour sélectionner le modèle le plus probable, mais pas pour dériver directement les probabilités a posteriori associées. En effet, la fréquence des arbres associés au modèle majoritaire n'est pas un substitut adéquat à la véritable probabilité a posteriori.

D'après Robert et al. (2001), la probabilité a posteriori d'un modèle est la quantification de l'incertitude bayésienne car elle est le complément de l'erreur a posteriori associée à la perte : $\mathbb{I}(\hat{m}(\eta(y)) \neq \mathcal{M})$. D'après Stoehr et al.(2015), on peut estimer ensuite le taux de l'erreur conditionnel par :

$$\begin{aligned} \mathbb{E}[\mathbb{I}\{(\hat{m}(\eta(y)) \neq \mathcal{M})|\eta(y)\}] &= \mathbb{P}[\hat{m}(\eta(y)) \neq \mathcal{M}|\eta(y)] \\ &= 1 - \mathbb{P}[\hat{m}(\eta(y)) = \mathcal{M}|\eta(y)] \end{aligned}$$

ce qui nous donne le complément de la probabilité a posteriori que le vrai modèle soit le modèle sélectionné $\mathbb{P}[\mathcal{M} = \hat{m}(\eta(y))|\eta(y)]$. Cette estimation efficace taux d'erreur conditionnel nous fournit ainsi une estimation non paramétrique de la probabilité a posteriori, qu'on l'estime ensuite par une forêt aléatoire de régression.

Pour produire notre estimation de la probabilité a posteriori $\mathbb{P}[\mathcal{M} = \hat{m}(\eta(y))|\eta(y)]$, nous procérons comme suit :

1. Nous calculons la valeur de $\mathbb{I}(\hat{m}(\eta(y)) \neq \mathcal{M})$ pour la RF entraînée \hat{m} et pour tous les termes dans la table de référence ABC ; pour éviter le surapprentissage, nous utilisons l'erreur OOB.
2. Nous entraînons une RF de régression $\hat{\mathbb{E}}(.)$ qui infère $\mathbb{E}[\mathbb{I}\{(\hat{m}(\eta(y))) \neq \mathcal{M}|\eta(y)\}]$. Elle estime la variabilité de $\mathbb{I}(\hat{m}(\eta(y)) \neq \mathcal{M})$ en fonction du même ensemble de statistiques résumées, en se basant sur la même table de référence.
3. Nous appliquons cette fonction RF aux observations réelles résumées $\eta(y)$ et retournons $1 - \hat{\mathbb{E}}(.)$ comme estimation de $\mathbb{P}[\hat{m}(\eta(y)) = \mathcal{M}|\eta(y)]$.

Cela correspond à la représentation de l'algorithme 5 :

Algorithm 5 Estimation de la probabilité a posteriori du modèle sélectionné

- 1: **Étape (a) :** Utiliser la RF produit par l'algorithme 4 pour calculer les classificateurs OOB de tous les termes dans la table de référence et déduire l'erreur de prédiction binaire du modèle associé.
 - 2: **Étape (b) :** Utiliser la table de référence pour construire une fonction de régression RF $\hat{\mathbb{E}}(.)$ régressant l'erreur de prédiction du modèle sur les statistiques résumées.
 - 3: **Étape (c) :** Retourner la valeur de $1 - \hat{\mathbb{E}}(.)$ comme estimation de régression RF de $\mathbb{P}[\mathcal{M} = \hat{m}(\eta(y))|\eta(y)]$.
-

Ces algorithmes, 4 et 5 sont implémentés dans le package **abcrf** sur **R**, développé par Pudlo et al. [6]. Dans ce package, la fonction **abcrf()** est la fonction implémentée pour la construction des forêts aléatoires à partir d'une table de référence pour le choix du modèle par ABC-RF. La table de référence doit comprendre une colonne pour les indices des modèles à comparer, et des colonnes pour les statistiques résumées simulées. Dans la sortie, cette fonction affiche l'erreur OOB de notre RF. Une fois la forêt aléatoire construite, on peut afficher l'évolution de l'erreur OOB en fonction de nombre *ntree* d'arbres choisi en appliquant la fonction **err.abcrf()** afin de choisir le nombre d'arbres optimal qui minimise l'erreur OOB et le temps de construction de notre RF pour pouvoir ensuite améliorer la prédiction.

La fonction **predict.abcrf()** permet ensuite de prédire le vrai modèle des données observées, tout évaluant la probabilité a posteriori à partir d'un arbre aléatoire de régression $\mathbb{P}[\hat{m}(\eta(y)) = \mathcal{M} | \eta(y)]$. [Package abcrf](#)

4.4 Illustration de la méthodologie ABC-RF

Pour illustrer cette nouvelle approche ABC-RF pour le choix du modèle, nous allons reprendre l'exemple de classification entre les modèles moyennes mobiles MA(1) vs MA(2) 6. Nous utilisons une table de référence $(m, \eta(z))$ de taille $N_{\text{ref}} = 10^4$, où z représente la série de moyennes mobiles, m l'indice du modèle, et $\eta(z) = (\eta_1(z), \dots, \eta_7(z))$ est le vecteur des sept premières autocorrelations.

Une forêt aléatoire de classification est entraînée sur la table de référence $(m, \eta(z))$ pour prédire les indices des modèles via un vote majoritaire. La forêt aléatoire de classification est construite à l'aide de la fonction **abcrf()**, et la prédiction des indices du modèle réel est illustrée comme sur la figure 4.2 :

```
library(abcrf)

abcrf <- abcrf(Model ~ ., data= table_reference, ntree=500, lda= FALSE )
predict <- predict(abcrf, obs = Donnees_observe, training = table_reference, ntree=500)

predict$vote

##      MA(1) MA(2)
## [1,]   498     2
## [2,]   188   312
```

FIGURE 4.2

Ensuite, pour estimer la probabilité a posteriori que le modèle sélectionné soit le vrai modèle, nous calculons tout d'abord l'erreur locale binaire $\mathbb{I}(\hat{m}(\eta(z)) \neq \mathcal{M})$, en comparant l'indice du modèle réel et l'indice du modèle prédit dans le tableau d'entraînement. Une forêt aléatoire de régression est ensuite construite sur l'ensemble de données $(m, \mathbb{I}(\hat{m}(\eta(z)) \neq \mathcal{M}))$ pour prédire le taux d'erreur estimé, et par la suite la probabilité a posteriori que le modèle sélectionné soit le modèle réel. Cette forêt aléatoire de régression est implémentée dans la fonction **predict()** du package **abcrf** et retourne la probabilité a posteriori comme illustré ci-dessous 4.3 :

```

library(abcrf)

abcrf <- abcrf(Model ~ ., data= table_reference, ntree=500, lda= FALSE )
predict <- predict(abcrf, obs = Donnees_observe, training = table_reference, ntree=500)

predict

##   selected model votes model1 votes model2 post.proba
## 1       MA(1)      498        2     0.9774
## 2       MA(2)      188       312     0.6668

```

FIGURE 4.3

Sélection des statistiques résumées La fonction d'autocorrélation pour un processus MA(q) est donnée par :

$$\rho(k) = \frac{\sum_{i=0}^{q-k} \theta_i \theta_{i+k}}{1 + \sum_{j=1}^q \theta_j^2}$$

pour $k \leq q$ et $\rho(k) = 0$ pour $k > q$ où q est le lag ou "décalage", représentant la différence temporelle entre deux observations de la série.

Ainsi :

- pour la moyenne mobile d'ordre 1, MA(1), l'autocorrélation pour lag > 1 vaut zero. (numériquement très proche du zéro)
- pour la moyenne mobile d'ordre 2, MA(2), l'autocorrélation pour lag > 2 vaut zero. (numériquement très proche du zéro)

Les statistiques résumées qui sont significatives pour nos forêts aléatoires devraient donc inclure les premières et deuxièmes autocorrelations. En appliquant la fonction **VariableImpPlot()** du package **abcrf** sur notre objet "abcrf", nous obtenons le graphique ci-dessous, où les statistiques résumées les plus importantes sont bien les deux premières autocorrelations.

```
variableImpPlot(abcrf)
```

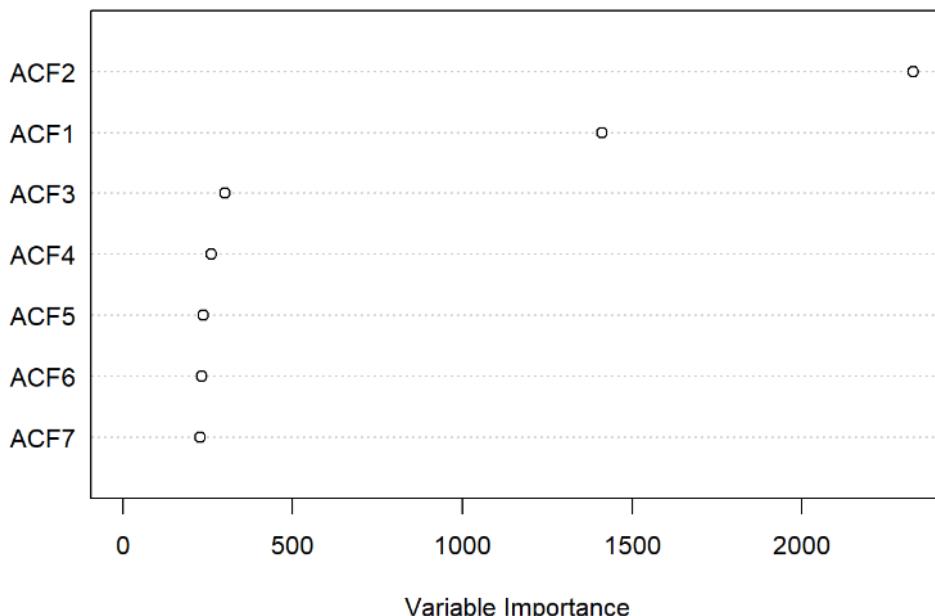


FIGURE 4.4

Chapitre 5

Application sur des données génétiques du riz d'Asie

Nous allons maintenant pouvoir appliquer les méthodes ABC-RF sur un problème concret de prédiction de scénario évolutif.

5.1 Les réseau phylogénétiques

Les réseaux phylogénétiques sont des représentations graphiques utilisées pour illustrer les relations évolutives complexes entre différentes espèces ou populations d'organismes. De plus, ils permettent de représenter des événements de réticulation tels que l'hybridation.

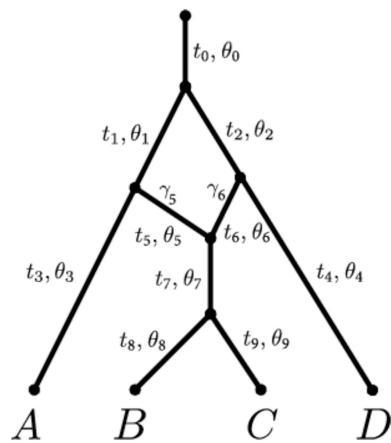


FIGURE 5.1 – Exemple de réseau phylogénétique illustrant l'hybridation entre quatre espèces. Les espèces B et C sont issues de l'hybridation entre A et D. Chaque branche i est associée à un temps de coalescence t_i et une taille de population effective θ_i [7]. De plus, les branches i au sommet d'un nœud de réticulation ont une probabilité d'héritage γ_i représentant leur probabilité d'avoir contribué à un espèce situé au sommet de la branche juste en dessous.

L'hybridation est particulièrement intéressante puisqu'elle aboutit à la formation de génomes mosaïques, c'est-à-dire des génomes qui combinent des éléments génétiques provenant des deux parents d'espèces différentes.

Par la suite, un scénario représente un réseau phylogénétique. Dans le cadre de notre étude, nous nous intéressons à l'évolution du riz asiatique cultivé. Nous considérons quatre groupes de riz cultivé (Indica, cAus, cBasmati et Japonica) et trois groupes de riz sauvage (Or1I, Or1A et Or3), tous issus d'un ancêtre commun *Oryza*.

Nous explorons trois scénarios évolutifs distincts, chacun représenté par un réseau phylogénétique spécifique :

Scénario 1 : une hybridation récente

Dans ce scénario, l'évolution génétique de la population de riz est caractérisée par une hybridation récente entre proto-Or1A (ancêtre de Or1A) et Japonica, donnant naissance à cBasmati.

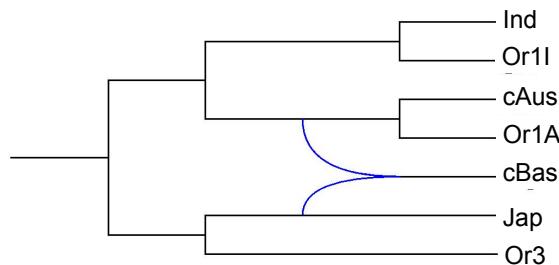


FIGURE 5.2 – Scénario 1 représenté par un réseau phylogénétique

Scénario 9 : une ancienne hybridation, puis une récente

Dans ce scénario, l'évolution génétique de la population de riz est caractérisée par une ancienne hybridation entre Japonica et proto-Or1I, donnant naissance à cAus et Or1A, suivie d'une hybridation récente entre le riz Japonica et cAus, issu de l'ancienne hybridation.

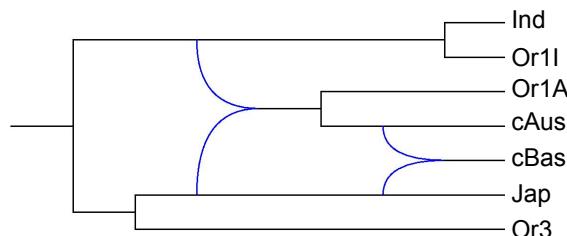


FIGURE 5.3 – Scénario 9 représenté par un réseau phylogénétique

Scénario 15 : une hybridation récente et une ancienne

Dans ce scénario, l'évolution génétique de la population de riz est caractérisée par une première hybridation entre Or1A et Japonica, donnant naissance à cAus, suivie d'une seconde hybridation entre les mêmes espèces, donnant naissance à cBasmati.

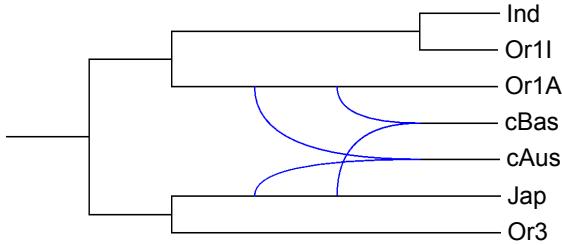


FIGURE 5.4 – Scénario 15 représenté par un réseau phylogénétique

5.2 Analyse des données génétiques du riz par la méthode ABC-RF

Nous allons maintenant utiliser la méthode ABC-RF pour prédire le scénario évolutif (le modèle) le plus probable d'un ensemble de données génétiques du riz. Ces données, similaires à celles utilisées dans Rabier et al. [7], se distinguent par une plus grande diversité intraspécifique et un nombre de simulations plus importants que dans leur étude basée sur MCMC [7].

Notre objectif est d'identifier, grâce à l'ABC-RF, le scénario évolutif le plus pertinent parmi un ensemble de scénarios prédéfinis (1, 9 et 15).

L'ABC-RF est particulièrement adaptée à l'analyse de données génétiques complexes car elle nécessite moins de simulations que les méthodes ABC classiques [4], tout en offrant une précision élevée dans la classification des scénarios évolutifs. De plus, elle permet d'identifier les variables importantes influençant l'évolution génétique, ce qui enrichit l'interprétation des résultats.

Notons que, par la suite, nous nous référerons au modèle ABC-RF entraîné en tant qu'algorithme.

Pour ce faire, nous disposons d'une table de référence contenant 10 000 simulations par scénario. Chaque simulation est caractérisée par 562 statistiques résumées, qui capturent les caractéristiques essentielles des données génétiques simulées. Cet algorithme apprend à partir des caractéristiques des simulations pour distinguer les différents scénarios évolutifs. Nous appliquons ensuite l'algorithme entraîné à un ensemble de données réelles de riz, dont les statistiques résumées ont été calculées de manière similaire.

5.3 Interprétation des résultats

Après avoir appliqué la méthode abcrf du package abcrf sur les données d'apprentissage, **nous avons obtenu une erreur Out-of-Bag (OOB) de 0.00804** (voir Figure 5.5), indiquant une bonne capacité de généralisation de l'algorithme à des nouvelles données.

```

Call:
abcrf(formula = Model ~ ., data =
Tableau_de_reference, lda = FALSE, ntree = 500)
Number of simulations: 24626
Out-of-bag prior error rate: 0.804%

Confusion matrix:
      1     9    15 class.error
1 7557   29   15 0.005788712
9 130  8526    1 0.015132263
15   22     1 8345 0.002748566

```

FIGURE 5.5 – La figure représente les sorties du modèle après apprentissage.

Pour évaluer les performances de l'algorithme, nous allons soumettre une observation réelle à la classification.

5.3.1 Prédiction

L'algorithme a prédict avec une probabilité à posteriori supérieure à 0,7 que cette observation appartenait au scénario d'évolution 15, suggérant que ce scénario d'évolution est le plus probable compte tenu des caractéristiques génétiques observées.

En résumé, ces résultats soulignent la capacité de l'algorithme à classifier efficacement les données génétiques du riz selon les différents scénarios d'hybridation. La combinaison d'une faible erreur OOB et d'une probabilité postérieure élevée confère une robustesse accrue aux classifications par rapport à l'approche ABC classique.

Ces résultats ouvrent des perspectives prometteuses pour de futures analyses visant à mieux comprendre l'évolution génétique du riz et à affiner les modèles prédictifs.

5.3.2 Variables importantes

En appliquant la méthode *varImpPlot* sur l'algorithme entraîné, nous avons identifié plusieurs variables importantes. Ces variables jouent un rôle crucial dans la classification des scénarios évolutifs du riz, et leur identification fournit des informations précieuses sur les facteurs qui influencent son évolution génétique.

La figure 5.6 présente ces variables importantes en fonction de leur Importance de Variable (VI), définie comme la moyenne de la différence des erreurs Out-of-Bag (OOB).

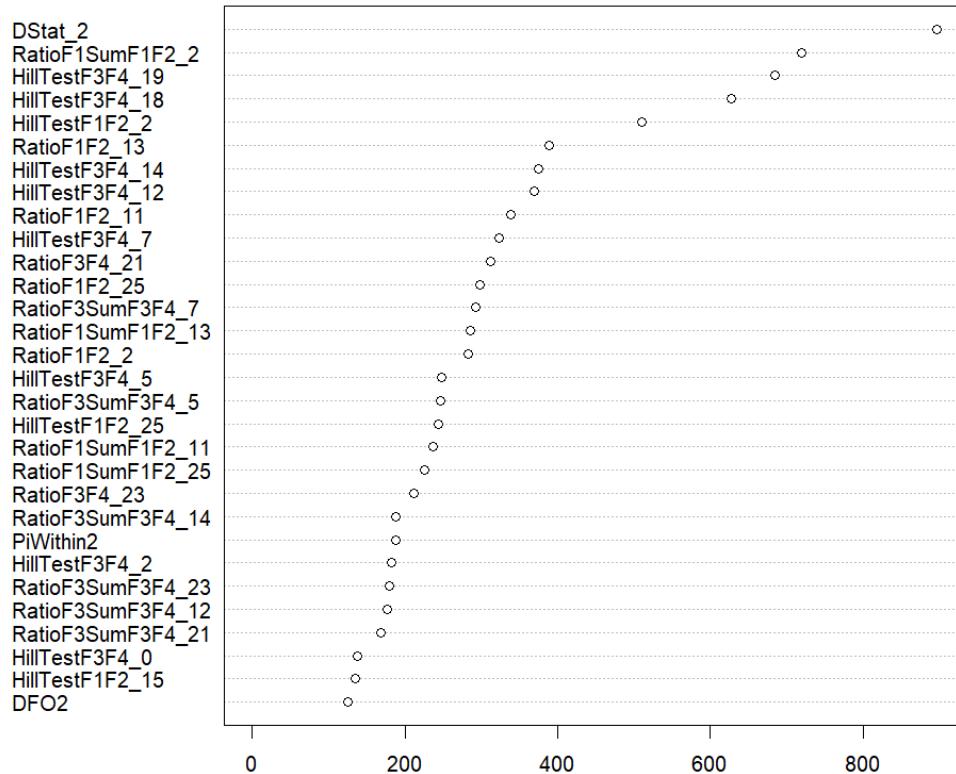


FIGURE 5.6 – Variables importantes présentées en fonction de leur VI (Importance de Variable).

5.3.3 Optimisation du paramètre

Dans l'objectif de minimiser l'erreur out-of-bag (OOB), nous avons étudié l'impact du nombre d'arbres sur les performances du modèle de forêt aléatoire. La figure 5.7 illustre l'évolution de l'erreur OOB en fonction du nombre d'arbres.

Nous constatons que l'erreur OOB diminue rapidement pour approcher son minimum dès 100 arbres, puis se stabilise. Le nombre optimal d'arbres, correspondant à l'erreur OOB minimale (environ 10^{-4}), se situe autour de 170. la réduction de l'erreur OOB au-delà de ce point étant très légère, nous choisissons de conserver le modèle à 500 arbre.

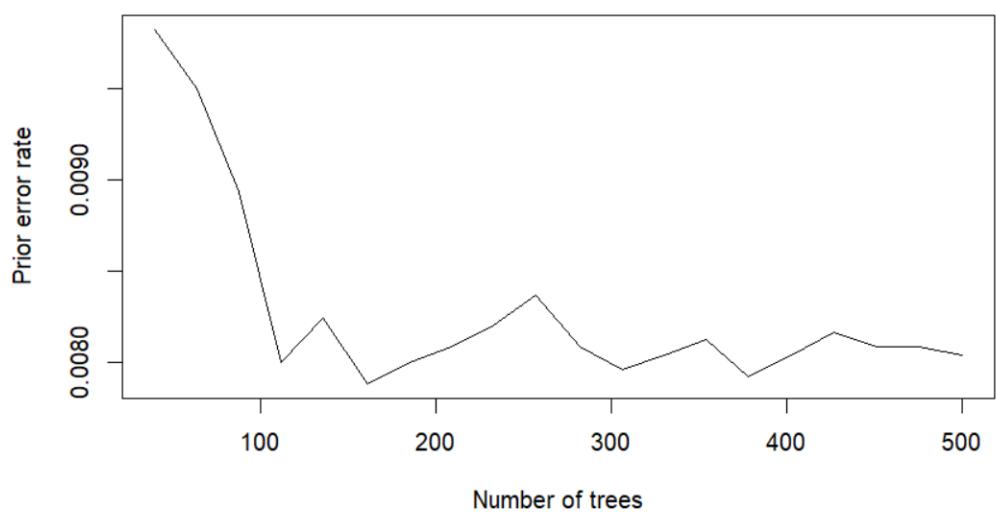


FIGURE 5.7 – Évolution de l’erreur out-of-bag (OOB) en fonction du nombre d’arbres dans le modèle de forêt aléatoire.

Chapitre 6

Conclusion

Le choix parmi un ensemble de modèles (scénarios individuels) constitue une question inférentielle cruciale, car il permet d'identifier les principaux événements historiques et évolutifs formalisés dans un ensemble de scénarios comparés, constitués comme une combinaison de tels événements évolutifs.

L'application à l'histoire évolutive du riz en Asie a permis d'évaluer les performances de l'ABC-RF dans le contexte de réseaux phylogénétiques, un domaine encore peu exploré dans la littérature, en surmontant plusieurs limitations des méthodes traditionnelles telles que l'Approximate Bayesian Computation (ABC) classique et les Markov Chain Monte Carlo (MCMC).

ABC-RF réalise deux avancées majeures : l'utilisation optimale des statistiques résumées et l'identification du modèle le plus probable. Concernant les statistiques résumées, ABC-RF extrait le maximum d'informations de l'ensemble des métriques disponibles, évitant ainsi le choix arbitraire d'un sous-ensemble de statistiques, une pratique courante dans les analyses ABC. Pour l'identification du modèle le plus probable, ABC-RF utilise un système de vote de classification plutôt que les probabilités postérieures traditionnellement employées. Pudlo et al.[\[6\]](#) démontrent que par rapport aux méthodes ABC classiques, ABC-RF offre au moins trois avantages : (i) une réduction significative de l'erreur de classification des modèles, mesurée par le taux d'erreur a priori ; (ii) une robustesse face au nombre et au choix des statistiques résumées, grâce à la capacité de RF à gérer de nombreuses statistiques superflues et/ou fortement corrélées sans impact sur la performance ; (iii) une diminution considérable de l'effort de calcul, RF nécessitant une table de référence beaucoup plus petite que les approches ABC plus standard. Par exemple, les traitements ABC-RF sur les populations de riz en Asie ont requis beaucoup moins de temps de calcul que l'approche ABC standard appliquée sur l'exemple des moyennes mobiles MA(2) [2.2.4](#), nécessitant une table de référence de 30 000 jeux de données simulés contre un million pour l'approche traditionnelle. Ces améliorations majeures en termes de calcul et d'inférence augmentent la possibilité d'utiliser des méthodes basées sur la simulation dans des études de cas exigeantes en calcul.

En somme, la méthode ABC-RF fournit un cadre robuste pour la sélection de modèles et l'évaluation de la confiance, étendant l'applicabilité et l'efficacité de l'ABC dans l'analyse statistique. Les algorithmes ABC-RF seront d'un grand intérêt pour le traitement statistique de jeux de données massifs dont la disponibilité augmente rapidement dans divers domaines de recherche, y compris, mais sans s'y limiter, la génétique des populations.

Annexes

Les figures en fonction des différentes calibrations de l'ABC

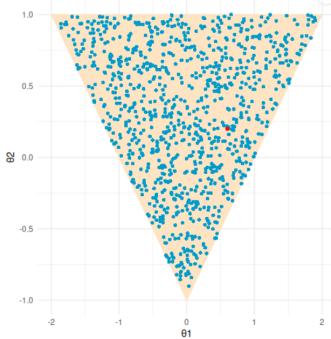


FIGURE 6.1 – On prend ici comme statistiques résumées les autocorrélations pour $\text{lag} = \text{c}(1,2)$. Le nombre de simulations est égal à $N_{ref} = 10^6$ et la tolérance est choisie de telle sorte que 0.1% des N_{ref} jeux de donnés que nous avons simulés soient acceptés.

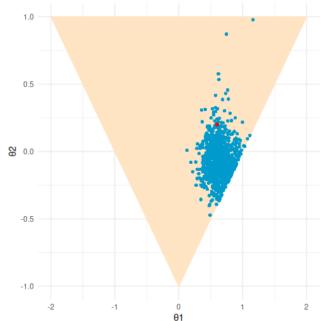


FIGURE 6.2 – On prend ici comme statistiques résumées les autocorrélations pour $\text{lag} = \text{c}(0,1)$. Le nombre de simulations est égal à $N_{ref} = 10^6$ et la tolérance est choisie de telle sorte que 0.1% des N_{ref} jeux de donnés que nous avons simulés soient acceptés.

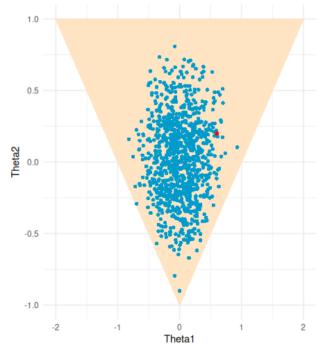


FIGURE 6.3 – On prend ici comme statistique résumée les autocovariances pour $\text{lag} = 0$ (la variance) et $\text{lag} = 1$. Le nombre de simulations est égal à $N_{ref} = 10^6$ et la tolérance est choisie de telle sorte que 0.1% des N_{ref} jeux de donnés que nous avons simulés soient acceptés.

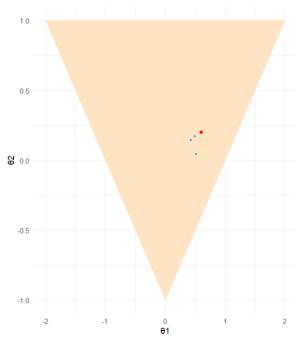


FIGURE 6.4 – On prend ici comme statistique résumée les autocovariances pour $\text{lag} = (0,1)$. Le nombre de simulations est égal à $N_{ref} = 30000$ et la tolérance est choisie de telle sorte que 0.1% des N_{ref} jeux de donnés que nous avons simulés soient acceptés.

1. Le code de l'exemple d'application de la méthode ABC sur les moyennes mobiles

```
library(ggplot2)
library(MASS)
library(gridExtra)
```

La fonction moyenne mobile d'ordre 2:

```
moving_average_2 <- function(theta1, theta2, n_obs=100) {
  lambda <- rnorm(n_obs + 2)
  y <- lambda[3:(n_obs+2)] + theta1*lambda[2:(n_obs+1)] + theta2*lambda[1:n_obs]
  return(y)
}
```

Les vraies thetas:

```
theta1_true <- 0.6
theta2_true <- 0.2
```

Les données observées:

```
y_obs <- moving_average_2(theta1_true, theta2_true)
```

Simulation des theta': pour q = 2: $-2 < \theta_1 < 2$, $\theta_1 + \theta_2 > -1$, $\theta_1 - \theta_2 < 1$.

```
grid <- function() {
  while (TRUE) {
    # Générer points aléatoires suivant loi uniforme
    theta1 <- runif(1, min = -2, max = 2)
    theta2 <- runif(1, min = -1, max = 1)

    # vérifier les conditions
    if (theta1 + theta2 > -1 && theta1 - theta2 < 1 && theta1 > -2 && theta1 < 2) {
      return(c(theta1, theta2))
    }
  }
}
```

Nous définissons la distance qui sera utilisée par la suite pour calculer celle entre les statistiques résumées:

```
#Euclidien
distance_euc <- function(y, z){
  return(sqrt(sum((y - z)^2)))}
```

La métrique de Mahalanobis sert à détecter les outliers ou pour déterminer la cohérence de données simulées par le modèle:

```

mahalanobis_distance <- function(y,z) {

  #calcul de la matrice de cov
  cov = cov(y,z)

  carre_diff <- y - z

  # Calcul de distance Mahalanobis
  dista <- sqrt(sum(t(carre_diff) %*% carre_diff)/cov)

  return(dista)
}

```

La statistique autocovariance:

```

autocov <- function(x, q = 2) { #q = max_lag

  n <- length(x)
  tau <- rep(NA, q) # vecteur des autocovariances

  # Calculer les autocovariances pour lags de 0 à q
  for (lag in 0:q) {
    if (lag == 0) {
      # Autocovariance pour lag 0 est la variance
      tau[lag + 1] <- var(x)
    } else {
      # Calculer pour lag = 1 et lag=2
      #tau[lag + 1] <- sum((x[1:(n - lag)]-mean(x)) * (x[(lag + 1):n] - mean(x)))/(n-lag)

      #autocov sur les data non centrées comme dans Marin et al.(2010)
      tau[lag + 1] <- sum(x[1:(n - lag)] * x[(lag + 1):n])/(n-lag)
    }
  }
  return(tau)
}

```

l'autocorrelation est utilisée dans Pudlo et al.(2016)

```

autocorr <- function(x, q=2){

  return(autocov(x,q)/var(x))
}

```

L'algorithme ed l'ABC:

```

ABC <- function(N, tol, eta = autocorr, dist = distance) {
  #N est le nombre des estimations générées
  #tol est la dégré de tolérance
  #eta est la statistique définie sur l'ensemble des data simulées
  #dist : La métrique

  theta_values <- matrix(nrow = N, ncol = 3) # Stocker les valeurs de theta dans une matrice
  i = 1
  for (i in 1:N) {
    #générer des theta'
    thetas_sim <- grid()
    #simulation des points c'est à dire la vraisemblance f(.|theta')
    z_sim <- moving_average_2(thetas_sim[1], thetas_sim[2])
    #calcul de la distance entre les statistiques des vraies données et celles simulées
    d <- dist(eta(y_obs), eta(z_sim))
    theta_values[i, ] <- c(thetas_sim[1], thetas_sim[2], d)
    i <- i+1
  }

  #on va utiliser comme critère de comparaison pour fixer la tol comme quantile
  sorted_thetas<-theta_values[order(theta_values[,3],decreasing=FALSE),]

  return(sorted_thetas[1:N*tol,1:2])
}

```

Execution de l'algorithme ABC:

```

set.seed(1)
N = 10^6
tol = 0.001
sampled_thetas <- ABC(N, tol, eta = autocov, dist = distance_euc)

```

La moyenne des estimations des thetas: (Remarque: La moyenne est assez proche des vrais θ avec les bonnes calibrations)

```

sampled_thetas_df <- as.data.frame(sampled_thetas)
theta_moy = c(mean(sampled_thetas_df[,1]), mean(sampled_thetas_df[,2]))
theta_moy

## [1] 0.6373197 0.3582331

```

Graphe:

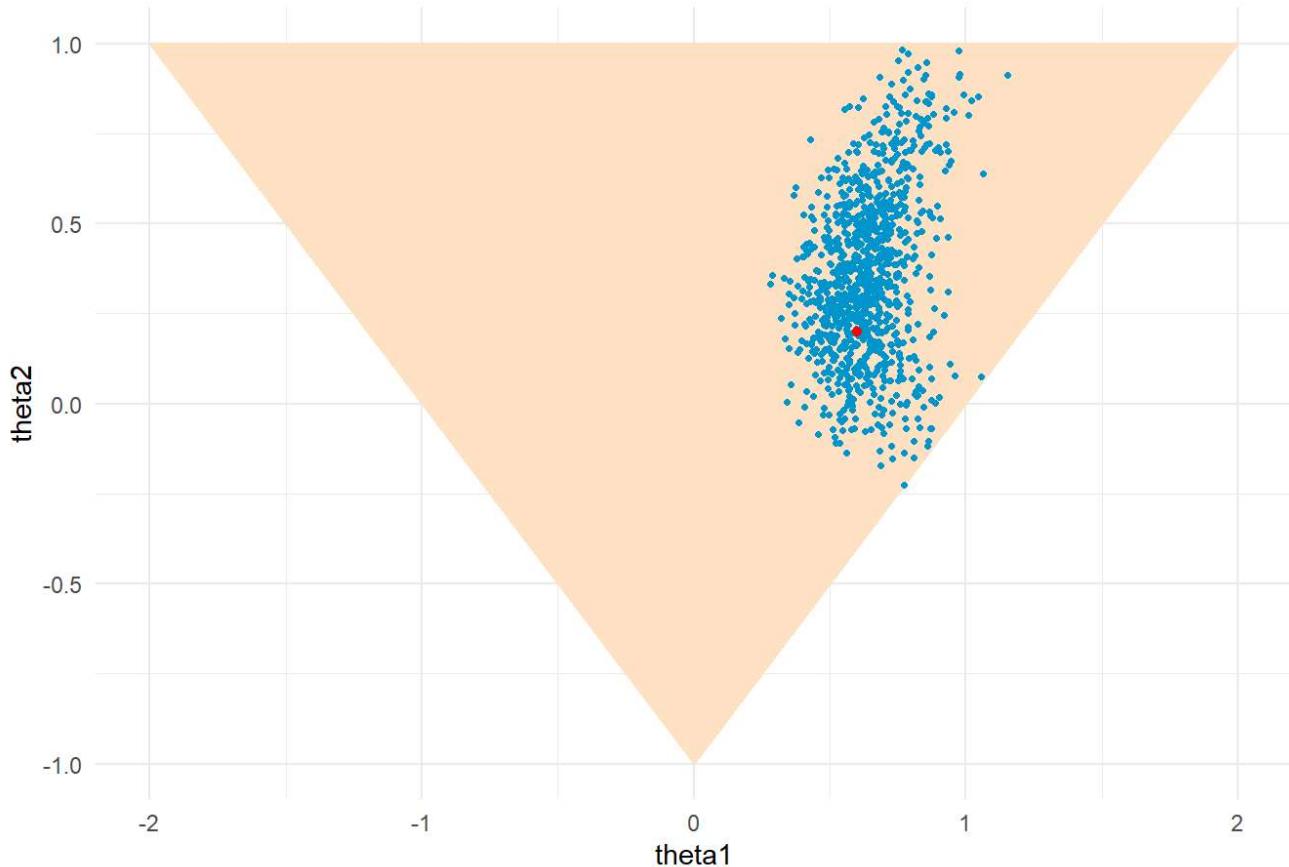
```

#Convertir la matrice des thetas à un data.frame
sampled_thetas_df <- as.data.frame(sampled_thetas)
colnames(sampled_thetas_df) <- c("x", "y")

ggplot() +
  geom_polygon(data = data.frame(u = c(-2, 0, 2), v = c(1, -1, 1)), aes(x = u, y = v), fill = "bisque1", color = "bisque1") +
  labs(x = "theta1", y = "theta2") +
  geom_point(data = sampled_thetas_df, aes(x = x, y = y), color = "deepskyblue3", shape = 20) +
  geom_point(data = data.frame(x = theta1_true, y = theta2_true), aes(x = x, y = y), color = "red", shape = 19) +
  theme_minimal()+
  labs(title = paste0("eta= autocov, la distance euclidienne, tol = 0.001 et N = ", N))

```

eta= autocov, la distance euclidienne, tol = 0.001 et N = 1e+06



Preuve de l'impact de la tolérance:

```

#Simu of ABC for different tol
N = 10**6
tol = c(0.1,0.001,0.001)
sampled_thetas1 <- ABC(N, tol[1],eta = autocov, dist = distance_euc)
sampled_thetas2 <- ABC(N, tol[2],eta = autocov, dist = distance_euc)
sampled_thetas3 <- ABC(N, tol[3],eta = autocov, dist = distance_euc)

```

```

# Extract the columns of sorted_thetas for each tol
theta11_values <- sampled_thetas1[, 1]
theta21_values <- sampled_thetas2[, 1]
theta31_values <- sampled_thetas3[, 1]

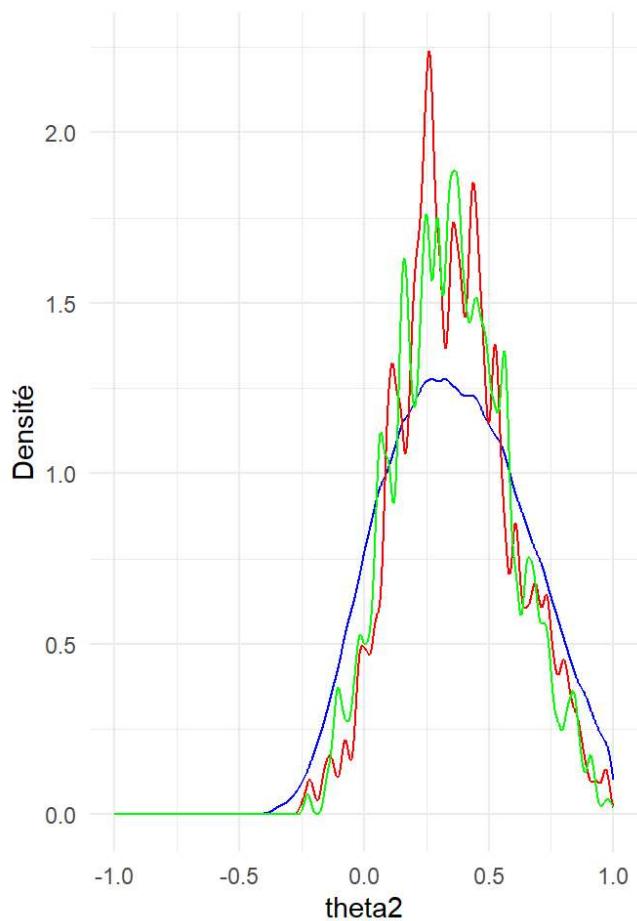
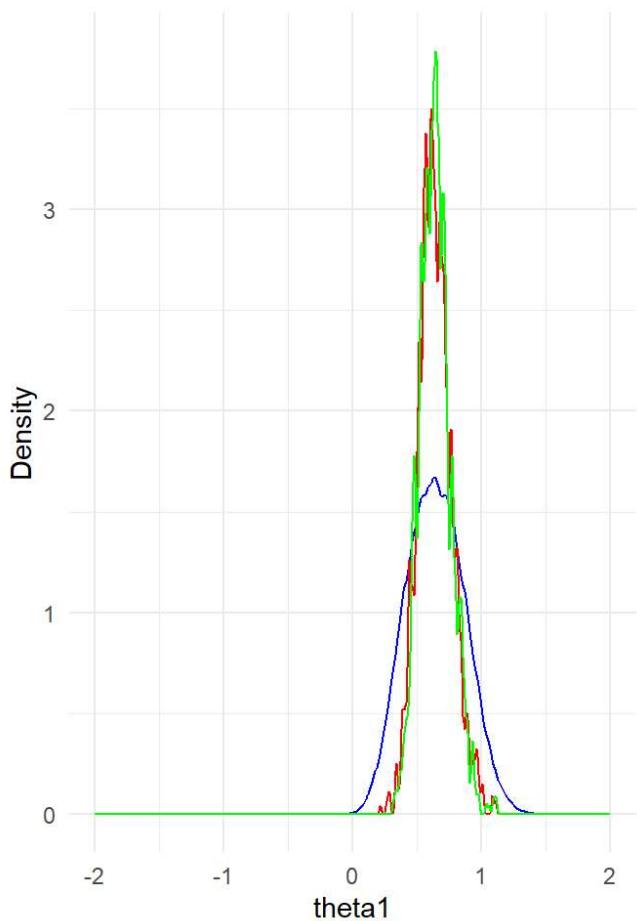
theta12_values <- sampled_thetas1[, 2]
theta22_values <- sampled_thetas2[, 2]
theta32_values <- sampled_thetas3[, 2]

library(gridExtra)
# Create plots for each column
graphique1 <- ggplot() +
  geom_density(color = "black") +
  geom_density(data = data.frame(theta = theta11_values), aes(x = theta), color = "blue") +
  geom_density(data = data.frame(theta = theta21_values), aes(x = theta), color = "red") +
  geom_density(data = data.frame(theta = theta31_values), aes(x = theta), color = "green") +
  labs(x = "theta1", y = "Density") +
  theme_minimal() +
  xlim(-2, 2)

graphique2 <- ggplot(data = data.frame(theta = theta12_values), aes(x = theta)) +
  geom_density(color = "black") +
  geom_density(data = data.frame(theta = theta12_values), aes(x = theta), color = "blue") +
  geom_density(data = data.frame(theta = theta22_values), aes(x = theta), color = "red") +
  geom_density(data = data.frame(theta = theta32_values), aes(x = theta), color = "green") +
  labs(x = "theta2", y = "Densité") +
  theme_minimal() +
  xlim(-1, 1)

grid.arrange(graphique1, graphique2, nrow = 1)

```



Code ▾

2. Le code pour les méthodes de classification: MA(1) vs MA(2)

la loi a priori des paramètres: Pour MA(1): simulation du pair (θ_1, θ_2) sur le triangle défini par $-2 < \theta_1 < 2$, $\theta_1 + \theta_2 > 1$, $\theta_1 - \theta_2 < 1$ Pour MA(2): simulation du θ_1 sur le segment $-1 < \theta_1 < 1$

Hide

```
theta_M1 <- function() {  
  runif(1, min = -1, max = 1)  
}  
  
theta_M2 <- function() {  
  while (TRUE) {  
    # Générer points aléatoires suivant loi uniform  
    theta1 <- runif(1, min = -2, max = 2)  
    theta2 <- runif(1, min = -1, max = 1)  
  
    # vérifier les conditions  
    if (theta1 + theta2 > 1 && theta1 - theta2 < 1) {  
      return(c(theta1, theta2))  
    }  
  }  
}
```

Générer MA(1) et MA(2) pour n=100

Hide

```
# Function to generate an MA(1) time series  
moving_average_1 <- function(theta1, n = 100) {  
  epsilon <- rnorm(n + 1)  
  y <- epsilon[1:n] - theta1 * epsilon[2:(n + 1)]  
  return(y)  
}  
  
# Function to generate an MA(2) time series  
moving_average_2 <- function(theta1, theta2, n = 100) {  
  epsilon <- rnorm(n + 2)  
  y <- epsilon[3:(n + 2)] - theta1 * epsilon[2:(n + 1)] - theta2 * epsilon[1:n]  
  return(y)  
}
```

Tableau de référence MA(1) et MA(2) et les sept premiers autocorrelations:

Hide

```

N_ref <- 10^4 # Define the number of simulations pour chaque MA

# Create an empty data frame to store results
reference_table <- data.frame(model = character(), theta1 = numeric(), theta2 = numeric(),
                                ACF1 = numeric(), ACF2 = numeric(), ACF3 = numeric(),
                                ACF4 = numeric(), ACF5 = numeric(), ACF6 = numeric(),
                                ACF7 = numeric(), stringsAsFactors = FALSE)

# Simulate and store results
set.seed(123)
for (i in 1:N_ref) {
  # Parameters for MA(1)
  theta1 <- theta_M1()
  series_MA1 <- moving_average_1(theta1)
  acf_values_MA1 <- acf(series_MA1, lag.max = 7, plot = FALSE)$acf[2:8]

  # Parameters for MA(2)
  theta_pair <- theta_M2()
  series_MA2 <- moving_average_2(theta_pair[1], theta_pair[2])
  acf_values_MA2 <- acf(series_MA2, lag.max = 7, plot = FALSE)$acf[2:8]

  # Append to the reference table
  reference_table <- rbind(reference_table, c("MA(1)", theta1, NA, acf_values_MA1))
  reference_table <- rbind(reference_table, c("MA(2)", theta_pair[1], theta_pair[2], acf_values_MA2))
}

# Set proper column names for clarity
colnames(reference_table) <- c("Model", "Theta1", "Theta2", "ACF1", "ACF2", "ACF3", "ACF4", "ACF5",
                               "ACF6", "ACF7")

# View the first few rows of the reference table
head(reference_table)

```

Mo...	Theta1	Theta2	ACF1	ACF2
	<chr>	<chr>	<chr>	<chr>
1 MA(1)	-0.424844959750772	NA	0.32153418225028	0.00176239188827978
2 MA(2)	1.87742564454675	0.934796741232276	-0.191337597017996	0.0529922609961209
3 MA(1)	-0.88361640740186	NA	0.522634430049573	-0.0554170464550251
4 MA(2)	1.38222449179739	0.612870327662677	-0.261520018060584	-0.218841606889112
5 MA(1)	-0.685409426689148	NA	0.308853949327986	-0.200742832122003
6 MA(2)	1.39565633703023	0.756083549465984	-0.140021995725897	-0.136395469569733

6 rows | 1-6 of 10 columns

Estimation des taux d'erreur de qq méthodes de classifications de MA(1) et MA(2):

Préparation des données:

Hide

```
str(reference_table) #inspecter les types des variables
```

```
'data.frame': 20000 obs. of 10 variables:  
 $ Model : chr "MA(1)" "MA(2)" "MA(1)" "MA(2)" ...  
 $ Theta1: chr "-0.424844959750772" "1.87742564454675" "-0.88361640740186" "1.38222449179739"  
 ...  
 $ Theta2: chr NA "0.934796741232276" NA "0.612870327662677" ...  
 $ ACF1 : chr "0.32153418225028" "-0.191337597017996" "0.522634430049573" "-0.261520018060584"  
 ...  
 $ ACF2 : chr "0.00176239188827978" "0.0529922609961209" "-0.0554170464550251" "-0.21884160688  
 9112" ...  
 $ ACF3 : chr "-0.0105990231989412" "0.062273010409666" "-0.194226242850955" "0.05173383113256  
 44" ...  
 $ ACF4 : chr "0.0306891807767046" "-0.261438163202731" "-0.180358910557505" "0.13277776591847  
 7" ...  
 $ ACF5 : chr "0.0624232941625019" "0.0862977194450783" "-0.12974569809809" "-0.09324912000532  
 92" ...  
 $ ACF6 : chr "0.183510423483166" "-0.15844324064886" "-0.0925970660927555" "-0.12525864483297  
 3" ...  
 $ ACF7 : chr "-0.0133123874857743" "-0.0868650219989817" "-0.0122538964309709" "0.08935537292  
 81109" ...
```

Hide

```
reference_table$Model <- as.factor(reference_table$Model)  
reference_table[,4:10] <- sapply(reference_table[,4:10], as.numeric)  
  
Data <- subset(reference_table, select = -c(Theta1, Theta2))  
head(Data)
```

Mo... <fctr>	ACF1 <dbl>	ACF2 <dbl>	ACF3 <dbl>	ACF4 <dbl>	ACF5 <dbl>	ACF6 <dbl>
1 MA(1)	0.3215342	0.001762392	-0.01059902	0.03068918	0.06242329	0.18351042
2 MA(2)	-0.1913376	0.052992261	0.06227301	-0.26143816	0.08629772	-0.15844324
3 MA(1)	0.5226344	-0.055417046	-0.19422624	-0.18035891	-0.12974570	-0.09259707
4 MA(2)	-0.2615200	-0.218841607	0.05173383	0.13277777	-0.09324912	-0.12525864
5 MA(1)	0.3088539	-0.200742832	-0.06012473	0.13480093	0.21567625	0.14585807
6 MA(2)	-0.1400220	-0.136395470	0.08009883	-0.14479407	0.12864134	0.13057084

6 rows

échantillonnage:

[Hide](#)

```
set.seed(123) # for reproducibility

# Define the proportion of data to use for training (e.g., 80%)
train_prop <- 0.8

# Randomly sample row indices for the training set
train_indices <- sample(1:nrow(Data), train_prop * nrow(Data))

# Create the training and testing sets
train_data <- Data[train_indices, ]
test_data <- Data[-train_indices, ]
```

Random Forest:

[Hide](#)

```
library(randomForest)
```

```
randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.
```

[Hide](#)

```
set.seed(123)
# Train Random Forest on all data
rf_model <- randomForest(Model ~ ., data = train_data, ntree = 500)

#prediction
predictions <- predict(rf_model, newdata = test_data[, -1])

#Evaluate Performance
error_rate_rf <- mean(predictions != test_data$Model) * 100 # Misclassification error rate

#error rate
error_rate_rf
```

```
[1] 10.025
```

[Hide](#)

```
library(class)

# Perform KNN with k = 50
knn_model <- knn(train = train_data[, -1], test = test_data[, -1], cl = train_data$Model, k = 50)

# Evaluate Performance
error_rate_knn_50 <- mean(knn_model != test_data$Model) * 100 # Misclassification error rate

# Display the error rate
error_rate_knn_50
```

```
[1] 10.5
```

[Hide](#)

```
library(class)

# Perform KNN with k = 50
knn_model <- knn(train = train_data[, -1], test = test_data[, -1], cl = train_data$Model, k = 100)

# Evaluate Performance
error_rate_knn_100 <- mean(knn_model != test_data$Model) * 100 # Misclassification error rate

# Display the error rate
error_rate_knn_100
```

```
[1] 10.575
```

[Hide](#)

```
library(e1071)

# Train Naive Bayes model
nb_model <- naiveBayes(train_data[, -1], train_data$Model)

# Make predictions
nb_predictions <- predict(nb_model, test_data[, -1])

# Evaluate performance
error_rate_nb <- mean(nb_predictions != test_data$Model) * 100 # Misclassification error rate

# Display the error rate
error_rate_nb
```

```
[1] 11.675
```

[Hide](#)

```
# Train logistic regression model
logit_model <- glm(Model ~ ., data = train_data, family = binomial)

# Make predictions
logit_predictions <- predict(logit_model, newdata = test_data[, -1], type = "response")

# Convert predicted probabilities to class labels
logit_predictions <- ifelse(logit_predictions > 0.5, "MA(2)", "MA(1)")

# Evaluate performance
error_rate_logit <- mean(logit_predictions != test_data$Model) * 100 # Misclassification error rate

# Display the error rate
error_rate_logit
```

```
[1] 13.6
```

[Hide](#)

```
library(MASS)

# Train LDA model
lda_model <- lda(Model ~ ., data = train_data)

# Make predictions
lda_predictions <- predict(lda_model, newdata = test_data[, -1])

# Extract predicted classes
lda_classes <- lda_predictions$class

# Evaluate performance
error_rate_lda <- mean(lda_classes != test_data$Model) * 100 # Misclassification error rate

# Display the error rate
error_rate_lda
```

```
[1] 13.975
```

[Hide](#)

```

library(knitr)

# Create a data frame with classifiers and error rates
classifier_error <- data.frame(classifieur = c("Random Forest", "knn (k = 50)", "knn (k = 100)",
"Naive Bayes", "Logistic Regression", "LDA"),
                                erreur = c(error_rate_rf, error_rate_knn_50, error_rate_knn_100,
error_rate_nb, error_rate_logit, error_rate_lda))

# Display the table
kable(classifier_error, caption = "Taux d'erreur de classification: MA(1) vs MA(2) ")

```

Taux d'erreur de classification: MA(1) vs MA(2)

classifieur	erreur
Random Forest	10.025
knn (k = 50)	10.500
knn (k = 100)	10.500
Naive Bayes	11.675
Logistic Regression	13.600
LDA	13.975

3. Le code de l'application sur les données génétiques du riz d'Asie

```
setwd("C:/Users/jaffa/Documents/DS - Sem2/TER")
```

```
tableref <- read.csv("MyReferenceTable", sep = " ")
tablereal <- read.csv("MyTableREALDataSecondSampling", sep = " ")
```

Données réelles

```
tablereal
```

Mo...	PiAll	PiWithin0	PiWithin1	PiWithin2	PiWithin3	PiWithin4	PiWithin5	PiWithin6	Pi
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
100	0.227337	0.095737	0.0970333	0.126604	0.110978	0.202202	0.229068	0.247482	

1 row | 1-10 of 563 columns

Nettoyage des tableaux

```
#Vérifier s'il y a des valeurs manquantes dans Les données
any(is.na(tableref))
```

```
## [1] TRUE
```

Nettoyage des lignes avec des valeurs manquantes

```
tab_sans_na <- tableref[complete.cases(tableref), ]
```

Afficher le nombre des lignes par scénario du tableau nettoyé

```
View(tab_sans_na)
table(tab_sans_na$Model)
```

```
##
##    1     9    15
## 7601 8657 8368
```

```
#str(Data) #vérifier Les types des variables
```

Categorisation du colonne des modèles

```
tab_sans_na$Model <- as.factor(tab_sans_na$Model)

tablereal$Model <- as.factor(tablereal$Model)
```

```
library(abcrf)
Tableau_de_reference <- tab_sans_na
model <- abcrf(Model ~ ., data= Tableau_de_reference, ntree=500, lda= FALSE )
```

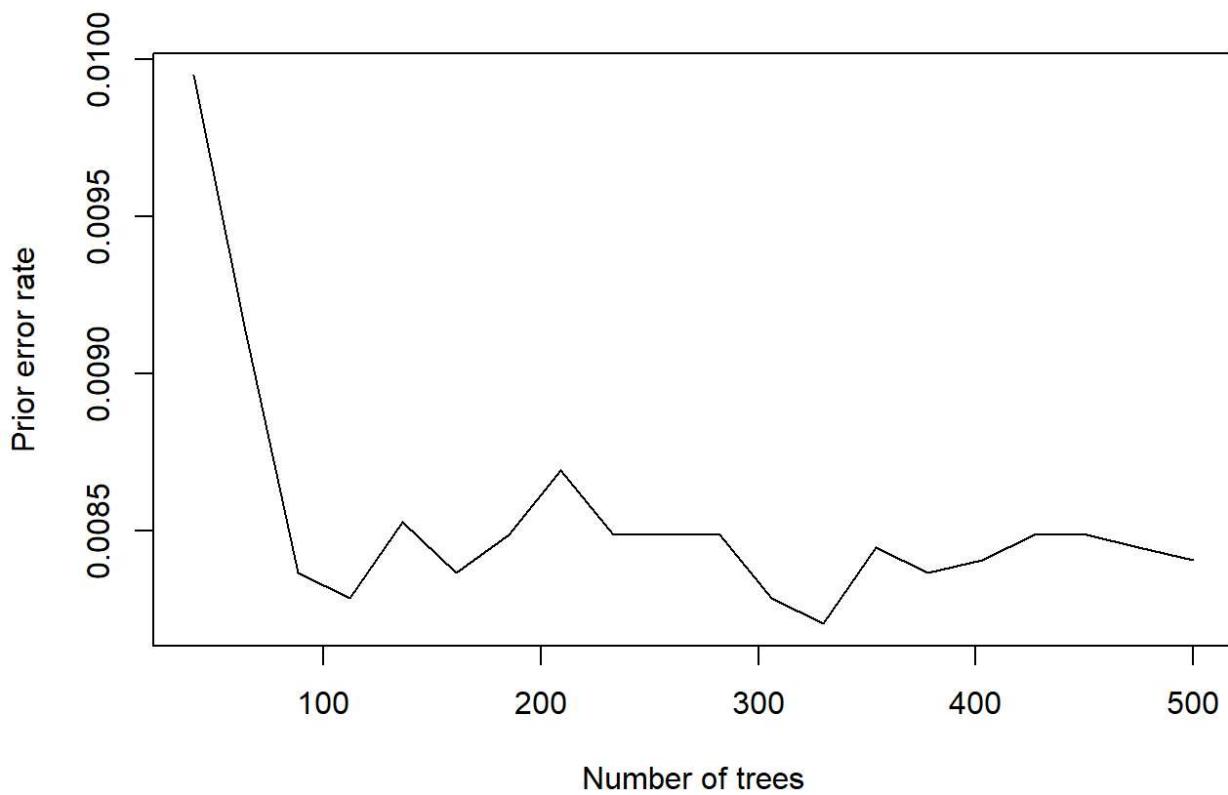
```
## Growing trees.. Progress: 23%. Estimated remaining time: 1 minute, 43 seconds.
## Growing trees.. Progress: 46%. Estimated remaining time: 1 minute, 13 seconds.
## Growing trees.. Progress: 69%. Estimated remaining time: 42 seconds.
## Growing trees.. Progress: 92%. Estimated remaining time: 11 seconds.
```

```
model
```

```
##
## Call:
## abcrf(formula = Model ~ ., data = Tableau_de_reference, lda = FALSE, ntree = 500)
## Number of simulations: 24626
## Out-of-bag prior error rate: 0.8406%
##
## Confusion matrix:
##      1     9    15 class.error
## 1 7553   31   17 0.006314959
## 9 134 8522    1 0.015594317
## 15  24    0 8344 0.002868069
```

Evolution de l'erreur OOB en fonction du nombre des arbres

```
err.abcrf(model, training = Tableau_de_reference)
```



```
##      ntree  error.rate
## [1,]    40 0.009948835
## [2,]    64 0.009136685
## [3,]    88 0.008365143
## [4,]   112 0.008283928
## [5,]   136 0.008527572
## [6,]   161 0.008365143
## [7,]   185 0.008486965
## [8,]   209 0.008690002
## [9,]   233 0.008486965
## [10,]  257 0.008486965
## [11,]  282 0.008486965
## [12,]  306 0.008283928
## [13,]  330 0.008202713
## [14,]  354 0.008446358
## [15,]  378 0.008365143
## [16,]  403 0.008405750
## [17,]  427 0.008486965
## [18,]  451 0.008486965
## [19,]  475 0.008446358
## [20,]  500 0.008405750
```

Application sur les données réelles

```
predict0 = predict(model, obs = tablereal[,-1], training = Tableau_de_reference)
```

```

## Growing trees.. Progress: 12%. Estimated remaining time: 3 minutes, 51 seconds.
## Growing trees.. Progress: 24%. Estimated remaining time: 3 minutes, 19 seconds.
## Growing trees.. Progress: 36%. Estimated remaining time: 2 minutes, 45 seconds.
## Growing trees.. Progress: 48%. Estimated remaining time: 2 minutes, 15 seconds.
## Growing trees.. Progress: 59%. Estimated remaining time: 1 minute, 47 seconds.
## Growing trees.. Progress: 70%. Estimated remaining time: 1 minute, 21 seconds.
## Growing trees.. Progress: 81%. Estimated remaining time: 52 seconds.
## Growing trees.. Progress: 92%. Estimated remaining time: 21 seconds.

```

```
print(predict0)
```

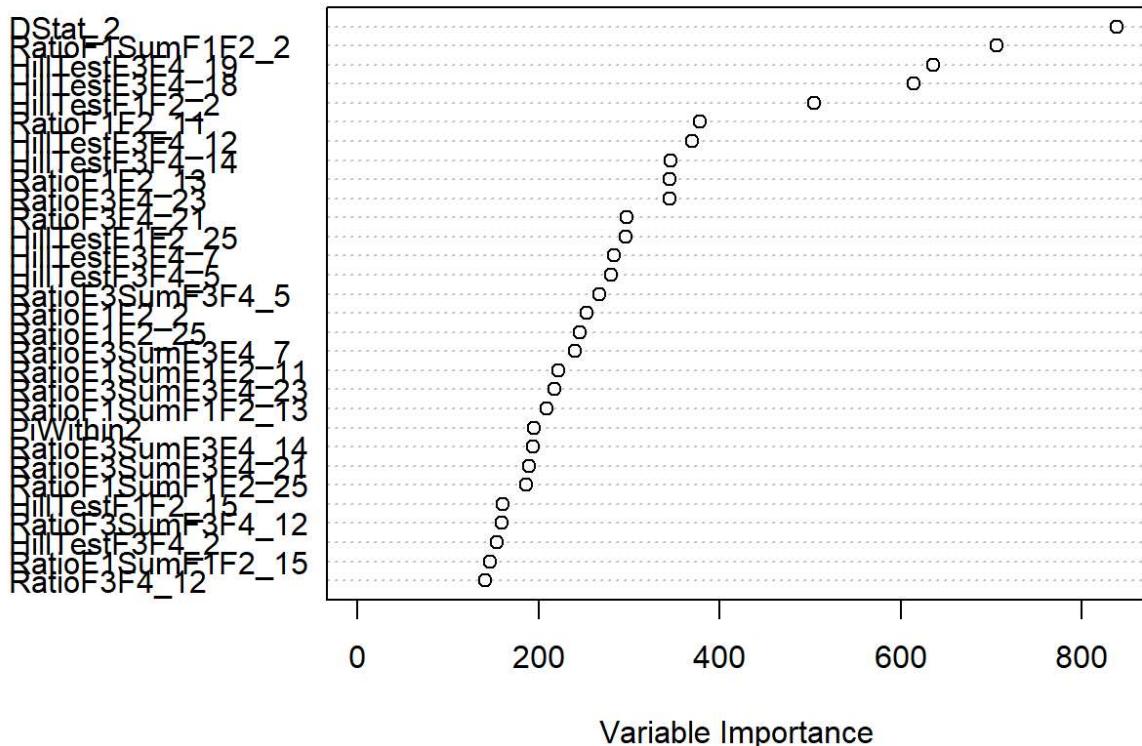
```

## selected model votes model1 votes model2 votes model3 post.proba
## 1           15          370         176        454     0.70095

```

Visualisation des statistiques résumées importantes dans l'abcrf

```
variableImpPlot(model)
```



```
model2 <- abcrf(Model ~ ., data= Tableau_de_reference, ntree=300, lda= FALSE )
```

```

## Growing trees.. Progress: 37%. Estimated remaining time: 52 seconds.
## Growing trees.. Progress: 75%. Estimated remaining time: 21 seconds.

```

```
model2
```

```
##  
## Call:  
## abcrf(formula = Model ~ ., data = Tableau_de_reference, lda = FALSE, ntree = 300)  
## Number of simulations: 24626  
## Out-of-bag prior error rate: 0.8568%  
##  
## Confusion matrix:  
##      1     9    15 class.error  
## 1 7554    34    13 0.006183397  
## 9 137 8520     0 0.015825344  
## 15    25     2 8341 0.003226577
```

```
predict2 = predict(model2, obs = tablereal[,-1], training = Tableau_de_reference)
```

```
## Warning in pred.all[nobs, ] == rep(i, ntree): longer object length is not a  
## multiple of shorter object length
```

```
## Warning in pred.all[nobs, ] == rep(i, ntree): longer object length is not a  
## multiple of shorter object length
```

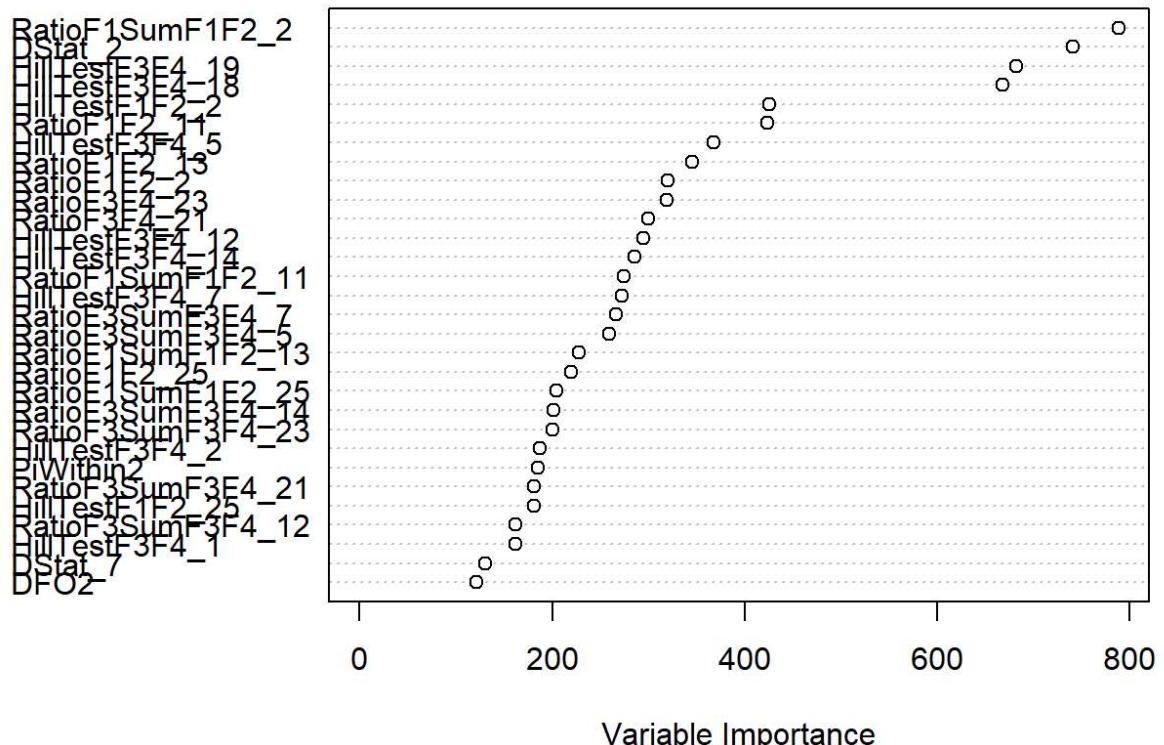
```
## Warning in pred.all[nobs, ] == rep(i, ntree): longer object length is not a  
## multiple of shorter object length
```

```
## Growing trees.. Progress: 13%. Estimated remaining time: 3 minutes, 29 seconds.  
## Growing trees.. Progress: 26%. Estimated remaining time: 2 minutes, 59 seconds.  
## Growing trees.. Progress: 39%. Estimated remaining time: 2 minutes, 27 seconds.  
## Growing trees.. Progress: 52%. Estimated remaining time: 1 minute, 56 seconds.  
## Growing trees.. Progress: 64%. Estimated remaining time: 1 minute, 25 seconds.  
## Growing trees.. Progress: 77%. Estimated remaining time: 54 seconds.  
## Growing trees.. Progress: 90%. Estimated remaining time: 24 seconds.
```

```
print(predict2)
```

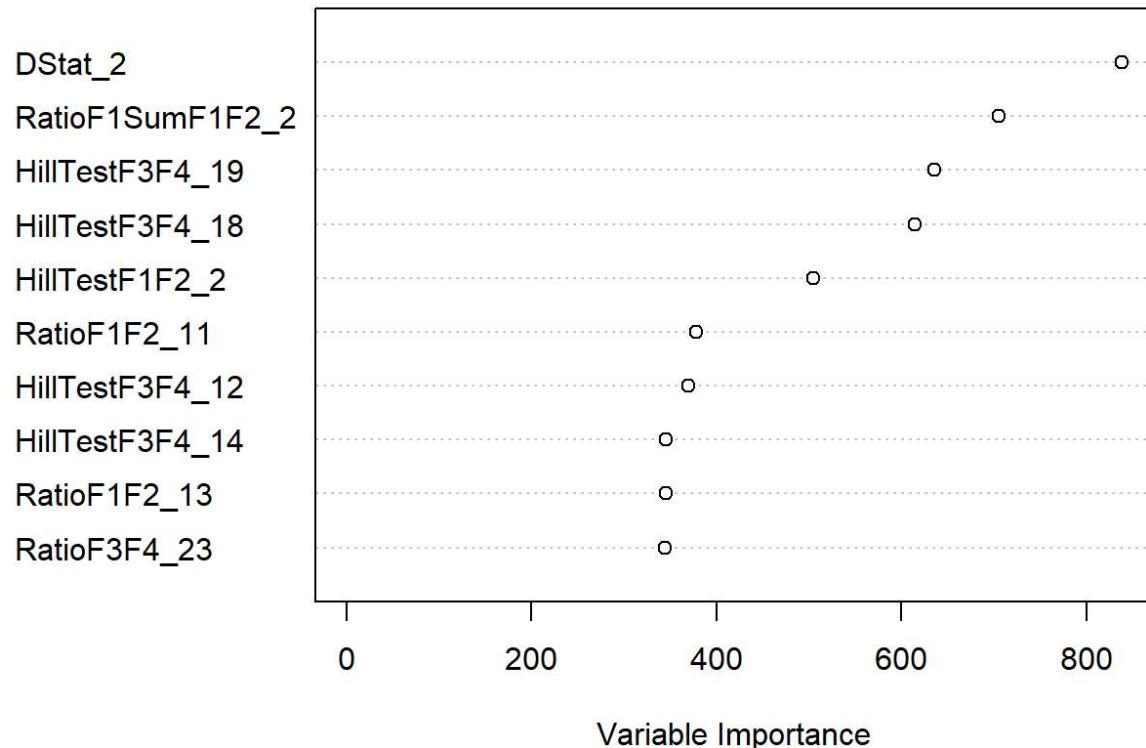
```
## selected model votes model1 votes model2 votes model3 post.proba  
## 1           15          368         192        440  0.6336167
```

```
variableImpPlot(model2)
```



```
variableImpPlot(model, n.var = 10, main = 'Importance des variables')
```

Importance des variables



Bibliographie

- [1] M. A. Beaumont, D. J. Balding, and B. (Eds.) McGuire, editors. *Handbook of Approximate Bayesian Computation*. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. Chapman and Hall/CRC, 2009.
- [2] Arnaud Estoup, Louis Raynal, Paul Verdu, and Jean-Michel Marin. Model choice using approximate bayesian computation and random forests : analyses based on model grouping to make inferences about the genetic history of pygmy human populations. *Journal de la Société Française de Statistique*, 159(3), 2018.
- [3] Robin Genuer and Jean-Michel Poggi. *Les forêts aléatoires avec R*. Pratique de la statistique. Presses Universitaires De Rennes, 03 2019.
- [4] Jean-Michel Marin, Pierre Pudlo, Christian P. Robert, and Robin J. Ryder. *Approximate Bayesian Computational Methods*, volume 26. 2011.
- [5] Osvaldo A. Martin, Ravin Kumar, and Junpeng Lao. Bayesian modeling and computation in python. 2021.
- [6] Pierre Pudlo, Jean-Michel Marin, Arnaud Estoup, Jean-Marie Cornuet, Mathieu Gautier, and Christian P. Robert. Reliable abc model choice via random forests. *Bioinformatics*, 32(6) :859–866, March 2016.
- [7] Charles-Elie Rabier, Vincent Berry, Marnus Stoltz, João D. Santos, Wensheng Wang, Jean-Christophe Glaszmann, Fabio Pardi, and Celine Scornavacca. On the inference of complex phylogenetic networks by markov chain monte-carlo. *PLOS Computational Biology*, 17(9) :e1008380, September 2021.
- [8] Louis Raynal, Jean-Michel Marin, Pierre Pudlo, Mathieu Ribatet, Christian P. Robert, and Arnaud Estoup. Abc random forests for bayesian parameter inference. *Genetics and Population Analysis*, 35(1) :173–190, January 2019.