Institut de Statistique, Biostatistique et Actuariat

Comparaison des méthodes de sélection des variables appliquées à la tarification des produits d'assurance non-vie

Mémoire présenté en vue de
l'obtention du master
en sciences actuarielles par :

TABOPDA WAFO Vaneck Clyde

Membres du jury :

Dr. VAN OIRBEEK Robin, *Promoteur*
Dr. BURY Thomas, *Copromoteur*
Prof. DENUIT Michel, *Lecteur*

Louvain-La-Neuve
Decembre 2021

Avant propos

Le présent mémoire s'inscrit dans le cadre de mes études en vu de l'obtention du diplôme de master en sciences actuarielles à l'Université Catholique de Louvain (UCLouvain) en Belgique. Cette prestigieuse école forme depuis des années des étudiants dans plusieurs disciplines dont l'actuariat qui a eu le mérite d'être classé plusieurs fois en tête de liste des meilleurs master du domaine assurance, risque et sciences actuarielles du classement "Eduniversal Best Masters Ranking".
C'est donc dans un environnement studieux et encadré par des professeurs d'une grande renommée que nous avons suivi notre formation en vue d'obtenir le diplôme en science actuarielle et d'intégrer l'institut des actuaires Belges.
Ce mémoire, en collaboration avec Allianz Benelux a pour but de comparer quelques méthodes de sélection des variables. En effet, quand on souhaite tarifier un nouveau produit d'assurance, il est souvent important de choisir les variables les plus pertinentes de la fréquence ou de la sévérité des sinistres afin de construire un modèle plus robuste et fiable. Cette sélection peut se faire de plusieurs manière et au cours de notre travail, nous explorerons quelques méthodes de sélection de variables qui sont basées sur le Boosting. Un accent particulier sera mis sur le concept de "variable importance".
Due à la crise sanitaire du Covid 19 ayant débuté pendant ce travail de recherche et à cause des conséquences de cette crise sur le fonctionnement des compagnies d'assurance, Les données utilisées dans le cadre de ce travail ne proviennent pas d'Allianz Benelux.

Remerciements

Mes remerciements s'adressent aux personnes m'ayant aidées et soutenues dans la réalisation de ce mémoire, ainsi que durant mon parcours universitaire.

Je souhaite tout d'abord exprimer ma profonde reconnaissance à l'égard de mes directeurs de mémoire Docteur Robin Van Oirbeek et Docteur Thomas Bury pour m'avoir choisi afin d'effectuer ce mémoire projet en collaboration avec la compagnie d'assurance Allianz Belgium, pour leur encadrement, leurs précieux conseils et leurs disponibilités tout au long du processus de rédaction de ce mémoire.

Mes remerciements vont également à l'endroit du Professeur Michel Denuit qui a accepté d'être le lecteur de ce travail.

Je remercie également l'équipe pédagogique de l'institut de statistiques, biostatistique et actuariat pour la qualité des enseignements reçus et leur disponibilité durant ma formation.

Je tiens à exprimer également toute ma reconnaissance à l'égard de ma famille en particulier mon grand frère Loic pour les sacrifices réalisés afin de me permettre de poursuivre ma formation dans de bonnes conditions.

Je remercie enfin tous mes amis de promotion pour les travaux de groupe et le partage d'information.

Table des matières

Table des figures

Chapitre 1

Introduction générale

La technologie informatique dont on dispose aujourd'hui nous offre une possibilité de stockage accrue. En partie pour cette raison, les bases de données, que l'on considère de nos jours, comptent un grand nombre de variables dites "explicatives". Cependant, la totalité de l'information collectée n'est pas forcément pertinente au vue du problème considéré. En assurance par exemple, lorsqu'on effectue de la tarification, en plus de sélectionner les variables qui permettent de réduire l'erreur du modèle considéré, il est également important de sélectionner les variables permettant d'expliquer le mieux la variable à prédire (fréquence ou sévérité des sinistres).
C'est la raison pour laquelle un objectif, tant pratique que théorique, consiste à déterminer, au sein de toutes les variables explicatives, celles qui sont réellement discriminantes pour l'application ou le problème étudié.
La sélection de variables est un processus qui permet de « sélectionner » un sous-ensemble de variables considérées par le processus comme pertinentes, permettant ainsi de supprimer le bruit généré par certaines variables et de réduire le coût de calcul de la phase d' apprentissage.
Un très grand nombre de méthodes de sélection de variables ont ainsi été définies avec pour objectif essentiel de garantir une bonne performance de prédiction, sans se soucier le plus souvent de certains paramètres tel que la présence des variables avec de très grande cardinalité, la présence des variables avec un grand pourcentage de valeur manquantes, ainsi que la dépendance entre les variables, pouvant affecter à la fois les performances de prédiction mais aussi la stabilité des méthodes de sélection de variables.

L'objectif principal de ce mémoire est d'effectuer une comparaison des méthodes de sélection de variables , méthodes de sélection appartenant la famille dite "all relevant feature selection" développées dans le module Python ARFS à savoir Leshy, BoostAGroota, GrootCV et ayant la particularité de se baser principalement sur les algorithmes du Boosting. Après avoir définit la concept de pertinence, nous allons également au cours de se travail explorer différentes métriques d'importance permettant de juger si une variable est pertinente ou pas pour un problème de régression.
Pour ce faire, nos méthodes se basant principalement sur le Boosting, nous allons dans le deuxième chapitre détailler le principe ainsi que certains algorithmes récent et populaire du Boosting qui sont utilisés comme régresseur pour les méthodes de sélection de variables de ARFS. Dans le chapitre 3, nous étudierons les deux principaux problèmes de la sélec-

tion des variables à savoir le problème minimal-optimal et le problème All Relevant qui est au coeur de ce mémoire, ensuite dans le chapitre 4 nous présenterons les différentes métriques d'importance utilisé pour nos méthodes de sélections et donnant ainsi la grandeur d'importance pour chacune des variables afin de voir celles qui sont pertinentes et celles qui ne le sont pas. Dans le chapitre 5 nous présenterons quelques méthodes de sélection de variables de ARFS précédé par la présentation de quelque méthodes sur lesquels les méthodes de ARFS se basent, et enfin dans le dernier chapitre, afin de comparer ses méthodes sur différents points, nous appliquerons ces méthodes sur un jeu de données artificiel où on a le contrôle sur le processus de génération des données et sur un jeu de données réel permettant de voir ainsi comment nos méthodes réagissent principalement en présence des prédicteurs colinéaires dans les variables explicatives.

Chapitre 2

Boosting

Le machine learning est une technique de programmation informatique qui utilise les probabilités et statistiques pour permettre aux ordinateurs d'apprendre d'eux-même, l'objectif étant d'apprendre aux ordinateurs d'agir sans être explicitement programmés en utilisant des programmes de développement qui s'ajustent chaque fois qu'ils sont exposés à différents types de données en entrée.
Dans le contexte du machine learning, les méthodes d'agrégations de modèles ont pour objectif de générer, puis d'agréger un ensemble de modèles dit de base, afin de construire un "méta modèle". L'idée sous-jacente de ces méthodes est de combiner des prédicteurs afin d'en améliorer les performances.
Le Boosting est une méthode d'agrégation de modèles. Il consiste à identifier des sous-problèmes, leur trouver une solution, puis à combiner ses solutions pour résoudre le problème général.
L'idée est qu'il est rare qu'un décideur ait sous la main un "expert" omniscient et incontesté lui permettant de faire le meilleur choix. Il n'a souvent d'autre ressource que de consulter un "comité d'expert" plus ou moins compétent, puis de combiner leur avis pour prendre sa décision. Ainsi, en réunissant un "comité d'experts", chacun peut se tromper, mais en combinant les avis, on a plus de chance d'avoir la bonne prédiction.
Le Boosting s'applique à des méthodes générales capables de produire des décisions très précises (au sens de la fonction de perte) à partir d'un ensemble de règles de décisions "faibles".
Avant de détailler le principe et les algorithmes du boosting, il convient de définir et présenter certaines notions et outils propres au machine learning qui seront par la suite utilisées sans grande ambiguïté.

2.1 Quelques Notions de base

Overfitting et underfitting

Un problème du machine learning est que certains modèles modélisent trop bien ou trop peu les données d'entraînement. L'overfitting se produit lorsqu'un modèle marche bien sur les données d'entraînement, mais ne généralise pas bien sur un jeu de données autre que les données d'entraînement, ce qui réduit la performance d'un modèle à un point où

le modèle devient inutile.
L'underfitting fait référence à un modèle qui ne peut ni modéliser les données d'entraînement, ni généraliser de nouvelles données.
L'overfitting est un problème car l'évaluation des algorithmes du machine learning sur les données d'entraînement est différente de celle qui nous intéresse le plus à savoir la performance de l'algorithme.
La figure suivante montre les règles d'overfitting et d'underfitting basée sur l'erreur prédictive et la complexité du modèle.

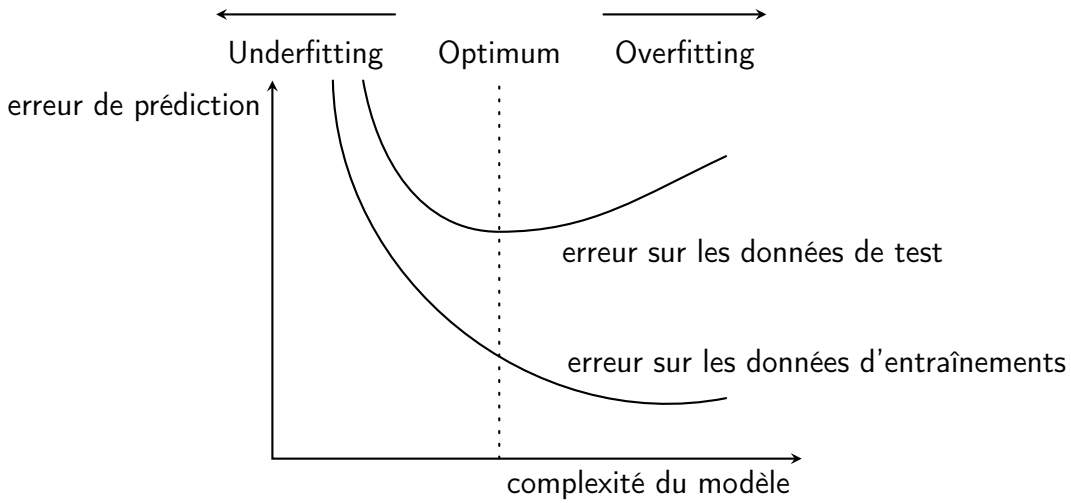


Figure 2.1.1 – Overfitting, underfitting et complexité d'un modèle

Compromis biais-variance

Le biais est une erreur de prédiction qui est introduite dans le modèle en raison de la simplification excessive des algorithmes du machine learning. Elle est la différence entre les valeurs prédites et les valeurs réelles ([1]).
Lorsque le modèle du machine learning fonctionne bien avec les données d'entraînement, mais ne fonctionne pas bien avec les données de test,une variance se produit.
Ainsi en machine learning, le compromis biais-variance est la propriété d'un modèle selon lequel la variance du paramètre estimé entre les échantillons peut être réduite en augmentant le biais dans les paramètres estimés. La décomposition biais-variance est un moyen d'analyser l'erreur de généralisation attendue d'un algorithme d'apprentissage par rapport à un problème particulier en tant que somme de trois termes : le biais, la variance et une quantité appelée erreur irréductible , résultant du bruit dans le problème lui-même ([1]).
Mathématiquement, soit Y la variable à prédire et X un vecteur de variables explicatives. On suppose qu'il existe une relation entre les deux tel que

$$Y = f(X) + e$$

où e est le terme d'erreur et est normalement distribué de moyenne nulle. On souhaite faire un modèle $\hat{f}(X)$ de f(X) en utilisant une régression linéaire ou tout autre technique de modélisation. Ainsi l'erreur au carré attendu en un point x est

$$Err(x) = E\Big[(Y - \hat{f}(x))^2\Big]$$

cette erreur peut être décomposé comme (voir [1]) :

$$\begin{aligned} Err(x) &= \Big(E[\hat{f}(x)] - f(x)\Big)^2 + E\Big[(\hat{f}(x) - E[\big(\hat{f}(x)\big)^2\Big] + \sigma_e^2 \\ &= \text{Biais}^2 + \text{Variance} + \text{erreur irreductible} \end{aligned}$$

où σ_e est l'erreur qui ne peut être réduit en créant de meilleur modèle.

Dans le contexte du machine learning l'overfitting se produit lorsque les modèles ont un faible biais et une variance élevée, tandis que l'underfitting se produit lorsque les modèles ont généralement un biais élevé et une faible variance (voir chapitre 2 [2]).
Afin de limiter le plus l'overfitting, une des techniques consiste à utiliser un ensemble de validation.

validation croisée

La validation croisée dite "cross validation" est une méthode d'estimation de fiabilité d'un modèle fondé sur une technique d'échantillonnage. Elle est une stratégie importante pour obtenir un modèle utile car elle permet d'estimer l'erreur que peut avoir un modèle sur un nouveau jeu de données. Dans la pratique on distingue trois ensembles de données :

- **Le training set** ou ensemble d'entraînement qui est utilisé pour entraîner le modèle et constitue donc une part importante du jeu de données.
- **Le validation set** ou ensemble de validation qui est utilisé pour le cross validation afin d'estimer l'erreur que pourra avoir notre modèle.
- **Le test set** ou ensemble de test qui est utilisé pour évaluer les performances du modèle finale. Ainsi, on compare en générale plusieurs modèle à travers leurs performances sur le test set.

Le **k-fold cross validation** est une technique consistant à diviser le jeu de données en k sous-ensembles où on utilise k-1 comme données d'entraînement et 1 comme données de validation, on répète le processus k-fois de telle sorte que chaque sous-ensemble soit utilisé comme données de validation exactement 1 fois, ensuite on fait la moyenne des k estimations afin d'obtenir l'estimation finale.

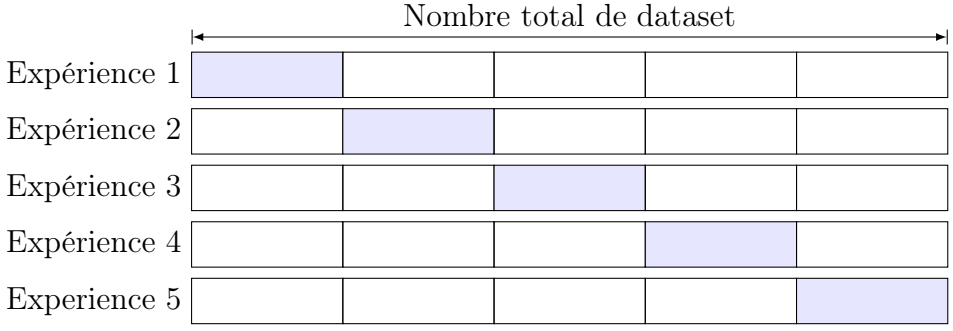


Figure 2.1.2 – Illustration d'un k-fold cross validation avec k=5

La figure 2.1.2 nous donne une illustration pour $k = 5$.

L'avantage d'une telle approche est que toutes les données peuvent être utilisées pour l'entraînement et toute anomalie présente dans l'un des sous-ensembles est susceptible de s'équilibrer dans le processus.

Fonction de perte

La fonction de perte est un outil important en machine learning . En effet, c'est cette fonction qui permet d'évaluer les performances d'un modèle. On reformule souvent un algorithme du machine learning sous forme d'un problème d'optimisation qui tente de trouver un ou plusieurs paramètres permettant de minimiser la fonction de perte.
L'une des fonctions de perte les plus utilisé est le **"mean squared error"** ou MSE qui est définie comme la moyenne des différences au carré entre les valeurs prédites $y^{'}$ et les valeurs réelles y.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i^{'} - y_i)^2$$

On peut faire le lien entre la MSE et le compromis biais-variance. Cela est donnée par (voir chapitre 2 [2]) :

$$\begin{aligned} MSE &= E\Big[(\hat{f}(x) - f(x))^2\Big] \\ &= E\Big[(\hat{f}(x) - E(\hat{f(x)}))^2\Big] + E\Big[(\hat{f}(x) - f(x))^2\Big] \\ &= variance(\hat{f}(X)) + biais^2 \end{aligned}$$

Une autre fonction de perte courante est le **"mean absolute error"** ou MAE qui est défini comme la moyenne des différences absolues entre les valeurs prédites $y^{'}$ et les valeurs réelles y. Elle a comme solution optimale la médiane et est pour cela plus robuste aux grandes valeurs. Elle est donnée par :

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i^{'} - y_i|$$

La MAE est donc utilisée pour mesurer à quel point les prédictions sont proches des résultats et est une moyenne de toutes les erreurs absolues. Elle est une mesure courante de l'erreur d'estimation dans l'analyse des séries chronologiques. La MSE mesure la moyenne des carrés des erreurs, c'est-à-dire la différence entre l'estimateur et l'estimation et est une fonction équivalente à la valeur attendue de la perte d'erreur au carré ou de la perte quadratique

2.2 Principe du Boosting

Les algorithmes du boosting construisent des "petits modèles" (solution partielle) et les combinent intelligemment pour obtenir un "grand modèle" (solution générale). Le travail

consiste alors à déterminer ces solutions partielles et à trouver une manière de les combiner. Pour cela, le boosting travaille de façon séquentielle. Il va commencer par construire un premier modèle qui va évaluer et à partir de là, chaque individu sera pondéré en fonction de la performance de prédiction (cours de LACTU 2110).
La construction se fait de façon récurrente : chaque modèle est une version adaptative du précédant en donnant un peu plus de poids lors de l'estimation suivantes aux observations mal prédites permettant ainsi de réduire à la foi la variance et le biais. L'effet de réduction de variance se comprend par le fait qu'en entraînant des classificateurs sur des échantillons différents, on génère des classificateurs différents. En agrégeant ces différents classificateurs, cela a pour effet de réduire la variance du classificateur agrégé.
L'idée est donc de combiner plusieurs règles faibles (c'est-à-dire un algorithme d'apprentissage qui est légèrement meilleur qu'une estimation aléatoire) afin d'approximer

$$f(x) = E(y/x)$$

où y est le est la variable expliquée et x le vecteur de variable explicative.
Nous allons à présent expliquer quelques algorithmes du boosting en commençant par le GBDT.

2.3 Gradient Boosting Decision Tree (GBDT)

Le Boosting regroupe de nombreux algorithmes et optimise leur performance.
un "weak learner" ou apprenant faible est un modèle statistique basique, ou plus généralement d'une méthode d'apprentissage permettant de fournir une prédiction de Y en fonction de X. L'algorithme du Gradient Boosting est un ensemble de "weak learner" créé les uns après les autres, formant un "strong learner" où chaque "weak learner" est entraîné pour corriger les erreurs de son prédécesseur (Cours de LACTU 2110). Si Y est la variable à expliquer et $X = (X_1, X_2, \ldots, X_p)$ le vecteur de variables explicatives, l'approche consiste à utiliser une descente de gradient pour minimiser le risque

$$R_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(Y_i, f(X_i))$$

où $l(y, f(x))$ mesure l'erreur (fonction de perte) entre la prédiction $f(x)$ et l'observation y.
Ainsi, étant donnée un modèle actuelle, on ajuste un weak learner aux résidus du modèle c'est-à-dire en utilisant les résidus plutôt que le résultat Y comme réponse, on ajoute ensuite ce nouveau weak learner dans la fonction ajusté afin de mettre à jour les résidus.
De façon générale, l'algorithme fonctionne comme suit (cours de LACTU 2110) :

- Le premier weak learner est basique, il s'agit juste de la moyenne des observations. Il est très peu efficace, mais servira de base au reste de l'algorithme.
- Ensuite on calcule l'écart entre cette moyenne et la réalité que nous appelons premier résidu. Le résidu est défini comme étant l'écart entre la prédiction de l'algorithme et la réalité et est donné par :

$$r_m = -\left[\frac{\partial l(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

. Le gradient boosting essaye de prédire à chaque étape non pas les données elles-mêmes, mais plutôt les résidus. Le second "weak learner" est entraîné pour prédire le premier résidu. Les prédictions pour ce "weak learner" sont ensuite multiplié par un paramètre λ dites paramètre "shrinkage" ou "Learning rate" utilisé pour contraindre le processus d'apprentissage et réduire le sur-ajustement ce qui permet donc d'augmenter la précision. Typiquement, $\lambda \in [0, 1]$.

- à partir de là, la créations des "weak learners" suit toujours le même schéma.

Algorithme 1 gradient boosting

---

initialiser $f_0(x) = argmin_\beta \sum_{i=1}^{n} l(y_i, \beta)$
pour m allant de 1 à M **faire** :
$r_m = -\left[\frac{\partial l(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$
entraîner un weak learner $\phi_m(x)$ sur les résidus r_m
estimer $\beta_m = argmin_\gamma \sum_{i=1}^{n} l(y_i, f_{m-1}(x_i) + \gamma\phi_m(x_i))$
$f_m(x) = f_{m-1}(x) + \lambda\beta_m\phi_m(x)$
finpour
retourner $f(x) = f_M(x)$

---

ici M représente le nombre d'itération et sa valeur peut être contrôlée via la validation croisée afin d'éviter l'overfitting et de sélectionner la valeur qui est optimale (cours de LACTU 2110).
Lorsque le "weak learner" est un arbre de décision, on parle de Gradient boosting décision tree (GBDT) ou gradient boosting machine. L'opération la plus coûteuse de GBDT est l'entraînement des arbres et la tâche la plus longue est de trouver les points de divisions optimaux, c'est à dire les valeurs des fonctionnalités dépendant des données qui sont divisées au niveau d'un nœud d'arbre. Les divisions optimales doivent être sélectionnées dans un pool de division candidate sur la base du gain d'information. GBDT utilise pour cela l'algorithme pré-trié c'est-à-dire énumère tous les points de partage possibles sur les valeurs pré-triées. Elle est simple mais inefficace en termes de puissance de calcul et de mémoire.
En somme, GBDT combine certaines des propriétés intéressantes des arbres de décisions comme leur flexibilité concernant les données d'entrées avec le concept de descente de gradient pour obtenir une méthode robuste tout en atteignant des performances élevées dans les situations pratiques où les données ne sont pas aussi bien conditionnées comme on l'aurait souhaité. La robustesse et les performances de cette méthode ont entraînées une augmentation exponentielle de sa popularité au cours des dernières années, ce qui s'est traduit par de nombreuses implémentations de ses algorithmes.

2.4 XGBoost : extrème gradient Boosting

XGBoost est une implémentation particulière du GBDT très utilisé en data science notamment pour la gestion des problèmes de régression, de classification, permettant d'obtenir des résultats de pointe sur les défis du machine learning ([3]). Il incorpore un modèle de régularisation afin d'éviter l'overfitting et incorpore également un algorithme "tree learning" pour le traitement des données énormes. L'évolution de XGBoost est due à plusieurs

systèmes importants et à des optimisations algorithmiques. XGBoost est une implémentation du GBDT conçue afin d'améliorer sa vitesse et sa performance ([3]).

Particularités

Une particularité de XGBoost réside dans le type de "weak learner" utilisé. Ici, les arbres qui ne sont pas assez bons sont élagués, c'est-à-dire qu'on leurs coupe des branches jusqu'à ce qu'ils soient suffisamment performants, sinon ils sont complètement supprimés. Cette méthode s'appelle "pruning" ou élagage. Ainsi, XGBoost s'assure de ne conserver que de bons "weak learner" ([4]).
De plus, l'algorithme de base est "parallélisable" c'est-à-dire qu'il permet d'exploiter la puissance des ordinateurs multicœurs permettant ainsi d'entraîner les modèles sur un très grand ensemble de données. XGBoost prend également en charge les paramètres de régularisations afin de pénaliser les modèles à mesure qu'ils deviennent de plus en plus complexe ([3]).

Terme de régularisation

Une nouvelle fonction objectif ℓ est considérée en complétant la fonction perte convexe différentiable l par un terme de régularisation :

$$\ell(f) = \sum_{i=1}^{n} l(y_i^{'}, y_i)) + \sum_{m=1}^{M} \Omega(\delta_m)$$

avec

$$\Omega(\delta) = \alpha|\delta| + \frac{1}{2}\beta||\omega||^2,$$

où $|\delta|$ est le nombre de feuilles de l'arbre de régression δ et ω le vecteur des valeurs attribuées à chacune de ses feuilles ([3]). Cette pénalisation ou régularisation limite l'ajustement de l'arbre ajouté à chaque étape et contribue à éviter un overfitting, notamment lorsque des observations affectées d'erreurs importantes sont présentes, augmenter le nombre d'itérations peut provoquer une dégradation de la performance globale plutôt qu'une amélioration. Le terme Ω s'interprète comme une combinaison de régularisation ridge de coefficient β et de pénalisation Lasso de coefficient α.

Données manquantes

La gestion de données manquantes ou de zéros instrumentaux anormaux est prise en compte de façon originale dans l'implémentation de XGBoost. Celui-ci propose à chaque division une direction par défaut si une donnée est manquante. Pour palier cette absence, le gradient est calculé sur les seules valeurs présentes ([3]).

Enfin d'autres spécificités de l'implémentation améliorent les performances de l'algorithme : stockage mémoire en bloc compressé pour paralléliser les opérations très nombreuses de tris, tampons mémoire pour réduire les accès disque, blocs de données compressés distribués (sharding) sur plusieurs disques.

XGBoost est implémenté en 'open source' et propose un panel d'hyperparamètre très important permettant d'avoir un contrôle total sur l'implémentation du gradient boosting. Avec toute ses évolutions, XGBoost a été pendant quelques années l'algorithme gagnant des compétitions "Kaggle" car rapide, précis et efficace, permettant une souplesse de manœuvre inédite sur le gradient boosting ([4]).

2.5 LigthGBM

Le GBDT est un algorithme du machine learning largement utilisé en raison de son efficacité, de sa précision et de son interprétabilité. Cependant, avec l'émergence du big data, GBDT fait face à de nombreux défis, notamment dans le compromis entre précision et efficacité. En effet, l'efficacité et l'évolutivité ne sont toujours pas satisfait lorsque la dimension de l'entité est élevée et en présence d'un grand nombre de variables car la complexité de calcul du GBDT est proportionnelle à la fois aux nombres d'instances de données et du nombre de variable du fait que pour chaque variable, le GBDT doit analyser toutes les instances.
Afin de relever ce défi, une idée simple consiste à réduire la taille des données et le nombre de variables. Deux nouvelles techniques permettent d'atteindre cet objectif :

- **Gradient based one Side Sampling** (GOSS) pour un échantillonnage des données d'entrée.
- **Exclusive Feature Bundling** (EFB) pour une réduction du nombre de variables.

Le nouvel algorithme GBDT avec GOSS et EFB est appelé ligthGBM et des expériences montrent que LigthGBM accélère le processus d'entraînement du GBDT de manière significative ([5]).
Notons également que contrairement à plusieurs algorithmes basés sur des arbres, ligthGBM construit ses arbres à la feuille (le meilleur en premier), tandis que les algorithmes classiques construisent les arbres par niveau (profondeur).

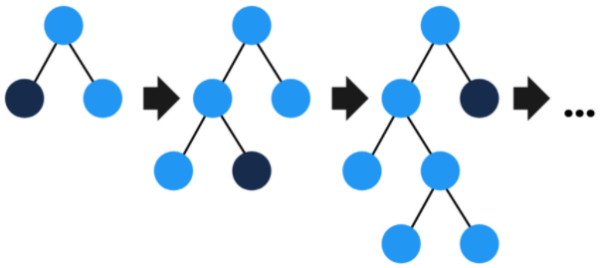


Figure 2.5.1 – Mécanisme de construction des arbres par feuille

2.5.1 Gradient based one side sampling (GOSS)

Le gradient de chaque instance de donnée dans GBDT nous fournit des informations utiles pour l'échantillonnage de données. Autrement dit, si une instance est associée à un petit gradient, l'erreur d'entraînement pour cette instance est petite et elle est déjà bien entraînée ([5]). Une idée simple consiste à supprimer ces instances avec un petit

gradient. Cependant, la distribution des données sera modifiée ce qui nuira à la précision du modèle. La nouvelle méthode GOSS permet d'éviter cela.

GOSS est une méthode d'échantillonnage pour le GBDT permettant d'atteindre un bon équilibre entre la réduction des données d'entrée et le maintien de la précision. Avec GOSS, on exclut une proportion significative des données d'entrée et on utilise le reste pour estimer le gain d'information. Ainsi, avec GOSS, on conserve toutes les instances avec un grand gradient (par exemple plus grand qu'un seuil prédéfini) et on effectue un échantillonnage aléatoire sur les instances avec un petit gradient. Afin de comparer l'influence sur la distribution des données, lors du calcul du gain d'information GOSS introduit un multiplicateur constant pour les instances avec un petit gradient ([5]).

Spécifiquement, GOSS trie d'abord les instances en fonction de la valeur absolue de leur gradient et sélectionne les $a * 100\%$ plus grandes instances. Ensuite, il effectue un échantillonnage aléatoire sur $b * 100\%$ du reste des instances, puis il amplifie les données échantillonnées avec un petit gradient d'une constante $\frac{1-a}{b}$ lors du calcul du gain d'information. En faisant cela, nous nous concentrons davantage sur les instances sous-entraînées sans modifier considérablement la distribution des données d'origine.
(voir l'algorithme pour GOSS à l'annexe A.1.1)

2.5.2 Exclusive Feature Bundling (EFB)

EFB est une nouvelle méthode pour réduire le nombre de variables. En effet, dans les applications réelles, bien qu'il existe un grand nombre de variables, l'espace des fonctionnalités est assez rare, ce qui offre la possibilité de concevoir une approche presque sans perte pour réduire le nombre de variables efficaces ([5]). Spécifiquement, dans un espace de fonctionnalités efficace, de nombreuses variables sont presque mutuellement exclusives, c'est-à-dire elles prennent rarement des valeurs différentes de 0 simultanément. EFB nous propose de regrouper en toute sécurité toute ces variables exclusives en une seule variable afin de réduire l'espace.

Example

Dans l'exemple suivant, "feature 1" et "feature 2" sont mutuellement exclusive. Afin, d'obtenir de compartiments ("buckets") qui ne se chevauchent pas, on ajoute la taille du "bundle" de "feature 1" à "feature 2". Cela garantit que les points de données non nuls des variables groupés ("feature 1" et "feature 2") résident dans différent compartiments. Dans "feature-bundle", les compartiments 1 à 4 contiennent les instances non nulles de "feature 1" et les compartiments 5,6 contiennent les instances non nulles de "feature 2" :

feature 1	feature 2	feature-bundle
0	2	6
0	1	5
0	2	6
1	0	1
2	0	2
3	0	3
4	0	4

Table 2.5.1 – Exemple illustratif de la fusion de deux variables mutuellements exclusives

Deux problèmes se poses :

Problème 1 : identifier les variables qui doivent être groupées ensembles

Une explication intuitive pour créer les groupes de variables est la suivante ([5]) :

1. Construire le graphe avec des arêtes pondérés c'est à dire qu'on mesure les conflits entre les variables où un conflit est la mesure de la fraction de variables exclusives ayant des valeurs non nulles qui se chevauche.
2. Trier ces variables en fonction de leurs nombres d'instance non nulle dans l'ordre décroissant.
3. Parcourir la liste ordonnée des variables et attribuer la variable à un "bundle"(groupe) existant (si conflit<seuil) ou créer un nouveau bundle (si conflit>seuil).

L'algorithme pour le groupement des variables est décrit à l'annexe A.1.2.

Problème 2 : savoir comment fusionner les variables

EFB fusionne les variables afin de réduire la complexité lors de l'entraînement. La clé est de s'assurer que les valeurs des prédicteurs d'origine peuvent être identifiées à partir du groupe de variables, c'est-à-dire rendre la fusion réversible. Pour cela, on va garder les variables exclusives dans différentes classes ([5]).
Une explication intuitive pour la fusion des variables est la suivante :

1. Calcul de la compensation à ajouter à chaque variable du groupe.
2. Itérer sur chaque instance et chaque variable.
3. Initialiser un nouveau compartiment à 0 pour les instances où toutes les variables sont nulles.
4. Calculer le nouveau compartiment pour chaque instance non nulle d'une variable en ajoutant une compensation (offset) respectif au compartiment d'origine de cette unité.

L'algorithme pour le processus de fusion des variables est décrite dans l'annexe A.1.2.

Nous venons ainsi de présenter deux implémentations du GBDT à savoir XGBoost et ligthGBM qui sont des algorithmes intégrés très populaire du machine learning en raison de leurs précisions et d'une bonne réduction du temps d'exécution. Ces algorithmes ainsi présentés font partis des plus utilisés mais il en existe plusieurs autres implémentations du GBDT dont CatBoost qui est un algorithme également populaire car permettant la gestion des variables catégorielles, c'est-à-dire des variables ayant des nombres discrets de valeur qui ne sont pas nécessairement comparables. Grâce à CatBoost, ces variables sont prises en charge directement lors de l'entraînement ce qui réduit le temps d'exécution car gérant directement la partie pré-entraînement. Ces implémentations du GBDT seront par la suite utilisées comme regrésseurs pour la sélection des variables pertinentes lors des applications pour ce mémoire.

Chapitre 3

Problèmes de sélection des variables

La sélection des variables est une étape indispensable de l'analyse des données lorsqu'il s'agit d'ensemble de données décrits avec des milliers de variables. Elle est appliquée pour sélectionner un ensemble beaucoup plus petit de variables pertinentes. La prémisse centrale lors de l'utilisation d'une technique de sélection des variables est que les données contiennent certaines variables qui sont redondantes et non pertinentes et peuvent donc être supprimées sans subir beaucoup de perte d'information. Cette réduction permet donc de faciliter la connaissance des classificateurs précis, d'accélérer le processus d'entraînement et de découvrir les variables les plus intéressantes qui peuvent permettre une meilleure compréhension du problème lui-même.
Il est question ici d'analyser deux problèmes distincts de sélection de variables :

- **Le problème "minimal-optimal"** dont l'objectif est d'apprendre un estimateur optimal en utilisant un nombre minimum de variables.
- **Le problème "all relevant"** dont l'objectif est d'identifier toutes les variables qui sont liées à la variable à prédire et qui est l'objectif de ce mémoire.

Nous verrons qu'all relevant est beaucoup plus difficile que minimal-optimal car cela nécessite une recherche exhaustive de sous ensemble, contrairement à minimal-optimal qui est résoluble en restreignant le problème à la classe de distribution de données strictement positive, et nous verrons que pour ces deux problèmes, il existe des algorithmes qui sont cohérents et optimaux (voir [6]).

3.1 Quelques définitions et concepts importants

Il est question dans cette partie de présenter quelques concepts importants pour nos développements futurs. Toutes les définitions, affirmations et démonstrations qui suivront dans cette partie proviennent de l'article [6]. On supposera un modèle de classification où les échantillons d'entraînement $(x^{(i)}, y^{(i)})_{i=1,..,n}$ sont des échantillons indépendants de variables aléatoires (X, Y) de densité $f(x, y)$ où $X = (X_1, .., X_n) \in \mathbb{R}^n$ est un vecteur et $Y \in \{-1, +1\}$ est une étiquette. Notons que les X_i désignent les variables aléatoires tandis que les x_i les observations. On considérera donc X comme un ensemble de variables et $R_i = X \backslash \{X_i\}$ pour l'ensemble de toutes les variables autres que X_i. Le domaine des

variables est noté par le symbole calligraphique $\mathcal{X}$ et on notera les densités $f(x)$ en continu et $p(x)$ en discret.

Classes de distribution strictement positive

Pour la sélection des variables, on tente de trouver des solutions sous-optimales, tout en considérant toutes les distributions de données possibles $f(x, y)$. En revanche, on limite la sélection des variables à certaines classes de distribution de données afin d'avoir des solutions optimales.

La classe de distribution de données strictement positive comprend les fonction $f(x, y)$ qui satisfont :

1. $f(x) > 0$ pour tous x dans sa mésure de Lebesgue
2. $P(p(y|X) = \frac{1}{2}) = 0$

Nous n'avons pas besoin de $f(x, y) > 0$ qui sera plus restrictif.
Le critère 1) stipule qu'un ensemble dans $\mathcal{X}$ a une probabilité nulle si et seulement si elle à une mesure de Lebesgue non nulle tandis que 2) stipule que la limite de décision optimale à une mesure nulle, et on verra que ces deux conditions sont principalement nécessaires pour garantir l'unicité des classificateurs de Bayes (voir la définition ci-dessous).
Il est raisonnable de supposer que cette classe couvre la grande majorité des problèmes pratiques de reconnaissance de formes, car la plupart des données proviennent des mesures physiques d'un certain type et sont inévitablement corrompues par le bruit et en général, la restriction strictement positive est considérée comme raisonnable chaque fois qu'il y'a une incertitude sur les données.

Classificateur,risque et optimalité

Un classificateur est une fonction $g(x) : \mathcal{X} \longrightarrow \mathcal{Y}$ prédisant une catégorie y pour chaque observation x.
La qualité d'un classificateur g se mesure par son risque $R(g)$ définie par :

$$R(g) = P(g(X) \neq Y) = \sum_{y \in Y} p(y) \int_{\mathcal{X}} 1_{\{g(X) \neq y\}} f(x|y) d_x$$

où $1_{\{.\}}$ désigne la fonction indicatrice.
Pour une distribution donnée, un classificateur de Bayes (ou optimal) est celui qui minimise $R(g)$ et on peut montrer que le classificateur g^* définie par :

$$g^*(x) = \begin{cases} +1 & si\ P(Y = 1|X = x) \\ -1 & sinon \end{cases}$$

est optimal (voir [7], et la demonstration à l'annexe B.1.1).
Désormais nous parlons de classificateur optimal g^* pour une donnée f.

Pertinence d'une variable

Une grande partie de la théorie est sur diverse définitions de la "pertinence" de variables. On la définit à travers le concept d'indépendance conditionnelle.

Une variable aléatoire X_i est conditionnellement indépendante d'une variable Y (conditionné par) étant donnée les variables $S \subseteq X$ si et seulement si

$$P(p(Y|X_i, S) = p(Y|S)) = 1$$

et on note Y$\perp$ X_i|S. L'indépendance conditionnelle est une mesure de non-pertinence.

Une variable X_i est **fortement pertinente pour Y** si et seulement si Y $\not\perp X_i|R_i$.
Formellement, une variable est fortement pertinente pour Y si elle contient les informations sur Y qui ne peuvent être obtenu à partir d'aucune autre variable.

Une variable X_i est **faiblement pertinente pour Y** si elle n'est pas fortement pertinente mais satisfait Y $\not\perp X_i|S$ pour tout ensemble $S \subseteq R_i$.
formellement, une variable faiblement pertinente contient également les informations sur Y, mais ces informations peuvent également être obtenu à partir d'autres variables.

Une variable X_i est **pertinente pour Y** si et seulement si elle est fortement ou faiblement pertinente pour Y.

Une variable X_i est **non pertinente pour Y** si elle n'est pas pertinente pour Y (ni fortement,ni faiblement pertinente pour Y).

Pertinence pour un classificateur

Une variable X_i est pertinente pour le classificateur g si et seulement si pour toute variable X'_i on a :

$$P(g(X_i, R_i) \neq g(X'_i, R_i)) > 0$$

où X_i et X'_i sont indépendants et identiquement distribué.
Cette définition stipule que pour être pertinente pour g, la variable X_i doit influencer la valeur de g(x) avec une probabilité non nulle.
La variable X_i sera fortement pertinente pour le classificateur de Bayes si la suppression de X_i seule dans les données entraîne toujours une détérioration de sa précision de prédiction et elle sera faiblement pertinente si elle n'est pas fortement pertinente, mais il existe un sous ensemble $S \subseteq R_i$ de sorte que la performance du classificateur sur S soit pire que celle sur S $\cup\{X_i\}$

3.2 Le problème minimal-optimal

Il consiste à trouver un ensemble de variables minimales et optimales, l'objectif étant d'apprendre un classificateur optimal en utilisant un nombre minimal de variables. Afin de résoudre le problème minimal-optimal, il faut recourir à des méthodes sous-optimales. Nous restreindrons le problème à la classe des distributions de données strictement positive car au sein de cette classe, le problème est polynomial dans le nombre de variables ([6]). De plus, dû à la mesure du bruit, la plupart des distributions de données rencontrées dans les applications pratiques sont strictement positives de sorte que notre résultat est largement applicable.

Dans la pratique, la distribution des données est inconnue et un classificateur doit être induit à partir des données d'apprentissage $D^l = \{(x^i, y^i)\}_{i=1}^l$ par un inducteur, définie comme une fonction $I : (\mathcal{X} \times \mathcal{Y})^l \to G$ où G est un espace de fonction ([6]).
Un inducteur est consistant si le classificateur induit converge en probabilité vers g^* (classificateur optimal). c'est à dire

$$I(D^l) \xmapsto{p} g^*$$

La consistance est un critère nécessaire et raisonnable pour un inducteur, et a été vérifié pour une grande variété d'algorithme([6]). Nous pouvons donc aborder le problème de sélection de variable en étudiant le classificateur de Bayes.
On définit **l'ensemble des variables minimal-optimal** S^* comme l'ensemble des variables pertinentes pour le classificateur de bayes g^*.
Clairement, S^* dépend uniquement de la distribution des données et est l'ensemble de variables minimales qui permet une classification optimale. Vu que g^* est unique pour les distributions strictement positives, il vient directement que S^* est unique. En effet, il existe un lien important entre l'ensemble Minimal-Optimal et le concept de pertinence forte.
Pour chaque distribution strictement positives f(x,y) l'ensemble minimal-optimal S^* contient uniquement **les variables fortement pertinentes** (voir théorème 8 de [6]).
En effet pour les distributions strictement positives f, g^* est unique. Considérons donc X_i une variable pertinente pour g^*. Par définition, on a $P(g^*(X_i, R_i) \neq g^*(X_i', R_i)) > 0$ et de la forme des classificateurs de Bayes et de son unicité pour les distribution positives, on a :

$$g^*(X_i, R_i) \neq g^*(X_i', R_i) \Longrightarrow p(y|x_i, r_i) \neq p(y|x_i', r_i) \qquad (1)$$

Ainsi,

$$P(p(Y|X_i, R_i) \neq p(Y|X_i', R_i)) \geqslant P(g^*(X_i, R_i) \neq g^*(X_i', R_i)) > 0$$

or

$$P(p(Y|X_i, R_i) \neq p(Y|X_i', R_i)) = 1 \Leftrightarrow P(p(Y|X_i, R_i) \neq p(Y|, R_i)) = 1$$

pour X_i et X_i' indépendants et identiquement distribués.
Ainsi

$$P(p(Y|X_i, R_i) \neq p(Y|X_i', R_i)) > 0 \Leftrightarrow P(p(Y|X_i, R_i) \neq p(Y|, R_i)) < 1$$

Ce qui veut dire que Y $\not\perp X_i|R_i$ et donc X_i est fortement pertinente.

Notons que l'unicité de g^* est requise car sans cela, l'implication (1) ne devait pas tenir. Ceci est important car il affirme que l'on peut ignorer en toute sécurité les variables faiblement pertinentes pour des distributions telle que $f(x) > 0$, et cela conduit à des algorithmes de recherches pour S^*.

3.2.1 Le problème all relevant

De façon générale, la sélection des variables est effectuée pour optimiser les performances des classificateurs (c'est-à-dire qu'on cherche à résoudre le problème minimal-optimal), mais cela n'apporte aucune compréhension approfondie de la variable à prédire. Le problème all relevant consiste à trouver les variables qui sont les plus pertinente pour la

variable cible et cela a suscité récemment beaucoup d'intérêt dans le domaine de la bio-informatique où il est question d'identifier toutes les variables (gènes) qui sont liées à la variable cible ([6]).

L'ensemble **des variables all relevant** noté S^A est défini comme l'ensemble des variables pertinentes pour la variable cible Y (voir définition 12 de [6]).
Il y'a donc deux problèmes centraux pour all relevant qui sont inexistant pour minimal-optimal

- Le premier est de détecter les variables faiblement pertinentes qui peuvent être obscurcis par d'autres variables.
- Le second est de discerner entre les variables faiblement mais vraiment pertinente,de celle qui ne sont pertinente qu'en apparence.

All relevant est beaucoup plus difficile à résoudre que minimal-optimal car il nécessite une recherche exhaustive de sous ensemble et la restriction à des distributions strictement positives n'est pas suffisante pour rendre ce problème traitable. Il faut donc trouver des contraintes supplémentaires.

Etant donné R,S,T,U des sous-ensembles disjoints de X et Y une variable, **une distribution PCWT** est une distribution strictement positive et satisfaisant les propriétés suivantes (voir [6]) :

- composition :
$$(S \perp T|R) \wedge (S \perp U|R) \Longrightarrow S \perp T \vee U|R$$
- faible transitivité :
$$(S \perp T|R) \wedge (S \perp T|R \vee Y) \Longrightarrow (S \perp Y|R) \vee (Y \perp T|R)$$

Où "$\Longrightarrow$" est le signe d'implication classique,"$\wedge$" l'intersection et "$\vee$" la réunion.

Afin de résoudre all relevant, il faut considérer la classe des distributions PCWT. Même si ces restrictions sont plus sévères que f(x)>0, ces classes sont assez réalistes pour de nombreux modèles (par exemples la distribution gaussienne conjointe est supposée être une distribution PCWT) et en considérant cette classe, il existe des algorithmes cohérents permettant de trouver l'ensemble S^A dont l'algorithme RIT qui a beaucoup d'implémentations.

Test d'indépendance récursif(RIT)

Une méthode simple et intuitive pour résoudre all relevant est de tester les fonctionnalités par paires pour les dépendances marginales : on teste d'abord chaque variable par rapport à la variable cible Y, puis on teste chaque variable par rapport à chacune des variables jugées dépendante de Y, et ainsi de suite jusqu'a aucune dépendance ne soit trouvé ([6]).
L'algorithme suivant décrit le processus :

Algorithme : test d'indépendance récursif (RIT)

Entrée : Y : Variable cible
Entrée :X : Ensemble des variables
pour X_i dans X **faire** :
    si $X_i \not\perp Y|\varnothing$ **alors** :
        $S = S \vee \{X_i\}$
    finsi
finpour
pour X_i dans S **faire**
    $S = S \vee RIT(X_i, X|S)$
finpour
retourner : S

Nous appelons cet algorithme l'algorithme RIT et il est prouvé que cet algorithme est consistant (converge vers S^A) pour les distributions PCWT (voir théorème 15 de [6]).

3.3 Méthodes pour la sélection des variables

Dans la pratique, on n'a pas souvent besoin de beaucoup de variables lors de la création d'algorithme du machine learning. Les principales raisons utilisées pour la sélection des variables sont :

- Permettre à l'algorithme du machine learning de s'entraîner plus rapidement.
- Réduire la complexité d'un modèle et le rendre plus facile à interpréter.
- Permettre de réduire le sur-ajustement.

Il est question de parler ici de diverse techniques et méthodologies utilisées pour définir notre espace des variables et aider nos modèles à mieux fonctionner.

les méthodes de filtrages

Les méthodes de filtrages sont généralement utilisées comme étape de pré-traitement. Ici, la sélection des variables est indépendante de tout algorithme du machine learning. Au lieu de cela, les variables sont sélectionnées sur la base de leur score dans divers tests statistiques pour leur corrélation avec la variable à prédire. La corrélation ici est un terme subjectif, les variables avec la corrélation la plus élevé sont les meilleurs.
Quelques tests utilisés sont(voir [8]) :

variables \ réponse	continue	catégorielle
continue	correlation de Pearson ou la correlation de Spearman	La régression logistique
catégorielle	Anova	Chi-carré

Table 3.3.1 – quelques méthodes de filtrages en fonction du type de la variable cible et de la variable prédictive.

- **corrélation de Pearson** : Elle est utilisé pour mesurer et quantifier la dépendance linéaire entre deux variables continues X et Y, et dont la valeur est donnée par :

$$\rho X, Y = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

avec $\rho X, Y \in [-1; 1]$.
la **corrélation de Spearman** est plus générale et est l'équivalent non paramétrique de la corrélation de Pearson.

- **La régression logistique ou modèle logit** : Il s'agit d'un modèle de régression binomiale permettant d'associer à un vecteur de variables aléatoire $(X_1, X_2, .., X_k)$ une variable aléatoire binomiale. Elle constitue un cas particulier de modèle linéaire généralisé et fournit un test statistique pour savoir si les moyennes de plusieurs groupes sont égales ou non.
- **Anova** : Il s'agit ici de l'analyse de la variance. Elle est similaire à LDA (analyse discriminante linéaire) à l'exception qu'il est exploité en utilisant une ou plusieurs variables indépendantes catégorielles et une variable dépendante continu. Il fournit également un test statistique pour savoir si les moyennes de plusieurs groupes sont égales ou non.
- **Chi-carré** : Il s'agit d'un test statistique appliqué aux groupes de variables catégorielles pour évaluer la probabilité de corrélation ou d'association entre ces variables en utilisant leur distribution de fréquence.

Retenons que les méthodes de filtrages ne suppriment pas la multicolinéarité. Elles ne sont pas conçues pour détecter les relations complexes avec la variable à prédire et ne sont généralement pas sensible pour tous les problèmes pertinents.

Les méthodes d'emballage ou "wrapper"

Dans les méthodes “wrapper”, on essaye d'utiliser un sous-ensemble des variables et de former un modèle en les utilisant. Ces méthodes s'appuient sur les informations à propos de la pertinence des variables obtenues d'une méthode de classification, et peuvent donc utiliser des informations plus approfondies sur les données que les filtres. Le problème se réduit à un ensemble de recherche, et donc ces méthodes sont très coûteuses en calcule. Certaines méthodes d'emballage permettent de résoudre le problème minimal-optimal. Comme exemple de méthodes, on a (voir [9]) :

- **La sélection en avant (forward selection)** : Il s'agit d'une méthode itérative dans laquelle nous commençons par n'avoir aucune variable dans le modèle. A chaque

itération, on ajoute la variable qui améliore le mieux notre modèle jusqu'à ce que l'ajout d'une nouvelle variable n'améliore pas les performances du modèle (les variables qui n'appartient pas au modèle finale ont une valeur p plus basse qu'un seuil prédéfini).

- **L'élimination en arrière (backward elimination)** : Dans cette méthode, nous commençons par utiliser toutes les variables. Puis, nous supprimons la variable la moins significative à chaque itération, ce qui améliore les performances du modèle. On répète cela jusqu'à ce qu'aucune amélioration ne soit observée lors de la suppression des variables.
- **L'élimination récursive des variables (stepwise selection)** : Il s'agit d'un algorithme d'optimisation qui vise à trouver le sous-ensemble des variables le plus performant. Il crée des modèles à plusieurs reprise et met de côté la meilleure ou la moins bonne des variables à chaque itération. Il construit le modèle suivant avec des variables de gauches jusqu'à ce que toutes les variables soient sélectionnées. Il classe ensuite les variables en fonction de leur ordre d'élimination.

Ces méthodes ci-dessus ne sont cependant pas des méthodes pour le problème all relevant. Nous verrons qu'il existe plusieurs implémentations des méthodes d'emballage pour all relevant (Boruta,BorutaShap,..) et Nous présenterons en chapitre 5 quelques implémentations des méthodes d'emballage pour all relevant utilisant les algorithmes du Boosting comme classificateurs.

les méthodes intégrées

Ces méthodes combinent les avantages des deux précédentes. Elles sont implémentées par des algorithmes qui ont leurs propres méthodes de sélection de variables. Ici, la sélection des variables s'effectue pendant la procédure d'apprentissage du classificateur, c'est à dire, ils optimisent explicitement l'ensemble des attributs utilisé pour obtenir la meilleure précision. Ces méthodes résolvent donc le problème minimal-optimal.

Chapitre 4

Mesures d'importance des variables

Entraîner un modèle qui prédit avec précision est excellent, mais souvent en plus de la prédiction, on souhaite être en mesure d'interpréter notre modèle. Si l'on souhaite par exemple prédire le prix des logements, savoir quelles variables sont les plus prédictives du prix nous indique pour quelles fonctionnalités les gens sont prêt à payer. L'importance des variables est donc l'outil d'interprétation le plus utile.
Il existe une variété de méthodes permettant une meilleure compréhension de la relation entre les variables prédictives et la variable target, mais la plupart des méthodes ne fournissent aucune idée de la mesure dans laquelle une variable contribue à la puissance prédictive du modèle défini ici comme l'importance de la variable. Cette perspective devient intéressante lorsqu'on rappelle que les modèles du machine learning visent la puissance prédictive plutôt que l'inférence (voir chapitre 8 [10]). Il est donc important également d'établir des méthodes qui se concentrent sur la dimension de la performance des variables.

Le mécanisme le plus courant pour calculer l'importance des variables et celui utilisé par les algorithmes du Random Forest est le mécanisme de diminution moyenne des impureté ou Gini impurity qui, pour une variable est calculé en mesurant son efficacité à réduire l'incertitude ou la variance lors de la création d'arbre. Elle peut être vu comme la diminution totale de l'impureté des nœuds en moyenne sur tous les arbres qui composent la forêt (voir [11]).
L'avantage est que le calcul de Gini Importance pour une variable ne prend pas beaucoup de temps supplémentaire, cependant il n'est pas nécessairement ce qui est le plus utile de mesurer car elle ne donne pas toujours une image précise de l'importance et ce mécanisme commun de calcul est biaisé car il a tendance à gonfler l'importance des variables continues par exemple ([11]).
On va donc présenter d'autres approches pour la mesure des variables qui sont plus stables, plus cohérentes, et permettant une bonne interprétation de la variable à prédire. Quelques exemples seront réalisées afin de visualiser comment les techniques fonctionnent en utilisant le jeu de données Boston du module scikit Learn qui contient 506 lignes de données et 13 colonnes et dont la tâche est de voir les variables les plus importantes pour prédire les prix des logements de Boston.

4.1 Permutation Importance

Le concept de "Permutation Importance" a été introduit par Breimann([12]) et appliqué sur un modèle de Random Forest. On réalise qu'il s'agit d'une technique courante, raisonnablement efficace et très fiable car elle mesure directement l'importance des variables en observant l'effet sur la précision du modèle lorsqu'on effectue un mélange aléatoire de chaque variable prédictive. Cette technique est largement applicable car elle ne repose pas sur les paramètres de modèle interne tel que les coefficient de régression linéaire par exemple.
L'idée est la suivante : lors de la permutation des valeurs d'une variable X_j, son pouvoir explicatif s'atténue car il rompt l'association avec la variable à prédire y. Par conséquent, si le modèle reposait sur la variable X_j, l'erreur de prédiction $e = L(y, f(X))$ du modèle f va augmenter lors de la prédiction sur le jeu de données avec la variable permutée par rapport à l'erreur sur le jeu de données original. L'importance de la variable X_j est alors évalué par l'augmentation de l'erreur de prédiction qui peut être déterminé en prenant la différence $e_{perm} - e_{orig}$ ou le ratio $\frac{e_{perm}}{e_{originale}}$. Une variable est considérée comme moins importante si l'augmentation de l'erreur de prédiction est comparativement faible et importante si cette erreur est importante. On peut déjà voir que lors du calcul de l'erreur de prédiction basé sur les entités permutées, il n'est pas nécessaire d'entraîner à nouveau le modèle.
L'algorithme suivant décrit le mécanisme pour Permutation Importance.

Algorithme pour Permutation Importance

Entrée : f : modèle entrainé
Entrée : X : Matrice des variables prédictives
Entrée : y : variable cible
Entrée : $L(y, f(x))$: mésure d'erreur
$e_{originale} = L(y, f(x))$
pour chaque X_j dans X **faire** :
    générer la matrice X_{perm} en permutant la variable X_j dans X
    estimer l'erreur e_{perm}=$L(y, X_{perm})$
    calculer l'importance de la variable $PFI_j = \frac{e_{perm}}{e_{originale}}$ ou $PFI_j = e_{perm} - e_{originale}$
fin pour
trier les variables par ordre décroissant de PFI

L'exemple ci-dessous est une illustration avec la métrique Permutation Importance pour un problème de régression sur le jeu données de Boston où l'on souhaite prédire le prix des logements de la ville de Boston avec un jeu de données à 13 variables et on obtient ainsi le classement de nos variables avec leur ordres d'importance.

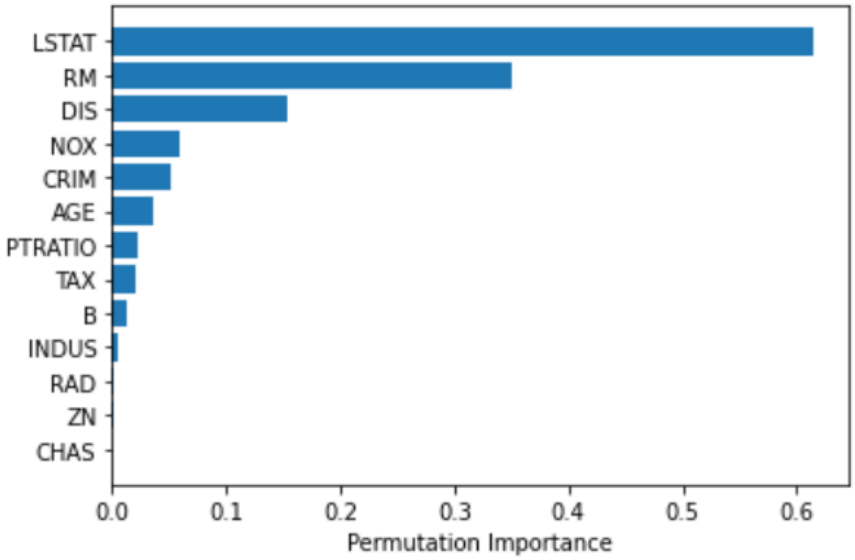


Figure 4.1.1 – Exemple d'utilisation de la métrique Permutation Importance pour déterminer l'importance des variables dans le jeu de donnée Boston utilisant XGBoost pour la prédiction

Le mécanisme de permutation importance est beaucoup plus coûteux en calcul que le mécanisme de diminution moyenne des impuretés, mais les résultats sont plus fiables et tout modèles peut utiliser cette stratégie pour calculer l'importance des variables ([11]).

Avantages

- Bonne interprétation : Elle est l'augmentation de l'erreur du modèle lorsqu'on permute les valeurs d'une variable et les résultats sont fiables
- Cette stratégie ne nécessite pas de ré-entraîner le modèle après avoir permuté chaque colonne, il suffit juste de re-exécuter les échantillons de test perturbé via la modèle déjà entraîné.
- En permutant une variable, on détruit également les effets d'interactions avec les autres variables. Ainsi, cette technique prend en compte à la fois l'effet de la variable principale et les effets d'interactions sur les performances du modèle.
- Cette technique n'est pas rigide par rapport à la métrique d'erreur.

Limites

- La technique de permutation importance dépend du mélange des instances de la variable, ce qui peut ajouter un caractère aléatoire à la mesure car lorsque la permutation est répétée, les résultats peuvent varier considérablement.
- Il est difficile de savoir si on doit utiliser les données d'entraînement ou de test pour calculer l'importance des variables via cette technique.
- Si les variables sont corrélées, l'importance de la variable peut être biaisée par des instances irréalistes. En effet, lorsque deux variables sont corrélées (comme la taille

et le poids d'une personne par exemple) et qu'on mélange une de ces variables, on crée une nouvelle instance peu probable, voir impossible (personne de 2 mètres pesant 30kg par exemple), pourtant ces nouvelles instances sont utilisées pour mesurer l'importance. Il faut donc toujours vérifier la corrélation entre variables avant d'utiliser la technique. Si plusieurs variables sont fortement corrélées, on ne doit garder qu'une seule de ces variables avant d'utiliser la métrique.

4.2 Valeurs Shapley et Shap importance

Une autre manière d'expliquer la prédiction est de supposé que chaque valeur d'une variable de l'instance est un "joueur" dans un jeu où la prédiction est le "paiement".

4.2.1 Les valeurs Shapley

Les valeurs Shapley sont des méthodes issues de la théorie des jeux de coalition qui nous indiquent comment repartir équitablement le "paiement" entre les variables et attribuent des paiements aux joueurs en fonction de leur contribution au paiement total. Ici, le jeu est la tâche de prédiction pour une seul instance de l'ensemble de donnée, le "gain" est la différence entre la prédiction réelle pour cette instance et la prédiction moyenne pour toute les instances, Les "joueurs" sont les valeurs des variables pour chaque instance qui collaborent pour recevoir le gain, c'est-à-dire la prédiction d'une certaine valeur (chapitre 9 [10]).
Supposons qu'on souhaite prédire les prix des logement avec un jeu de donnée de 3 variables prédictives A, B et C. La prédiction moyenne pour tous les logements est de 200000 €, et pour une instance de notre jeu de données, on prédit 220000 €. L'objectif est d'expliquer la différence entre la prédiction réelle (220000 €) et la prédiction moyenne (200000 €). Pour cela, on cherche à connaître dans quelle mesure chaque valeur des variables A, B et C a contribué à la prédiction par rapport à la prédiction moyenne. Une réponse naïve pourrait être par exemple que la valeur de A a apporté 10000 € dans la prédiction, celle de B a apporté 5000 € et celle de C a apporté 5000 €. La contribution s'élèvera donc à 20000 € qui est la différence entre la prédiction finale et la prédiction moyenne.

Pour une valeur d'une variable X_j, La valeurs Shapley est la contribution marginale moyenne de la valeur de X_j dans toutes les coalitions (associations) possibles. Pour chacune des coalitions, on cherche la valeur de la prédiction avec et sans la valeur de X_j, ensuite on prend la différence pour obtenir la contribution marginale. La valeur Shapley est donc la moyenne pondéré de toute les contributions marginales ([10]).
L'interprétation de la valeur Shapley pour une valeur de X_j est que la valeur de la j-eme variable a contribué ϕ_j à la prédiction de cette instance particulière par rapport à la prédiction moyenne de l'ensemble des données.
On s'intéresse à la manière dont la variable contribue à la prédiction d'une instance donnée.
Dans un modèle linéaire par exemple, il est facile de calculer l'effet individuel. En effet, la prédiction pour une instance dans un modèle linéaire est :

$$\hat{f}(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_p x_{i,p}$$

où i est l'instance pour laquelle on souhaite calculer la contribution, chaque $x_{i,j}$ est la valeur de la variable pour cette instance avec $j = 1 \dots p$ et β_j est le poids correspondant à la variable X_j. La contribution ϕ_j de la j-eme variable sur la prédiction $\hat{f}(\mathbf{x}_i)$ est

$$\phi_j(\hat{f}) = \beta_j x_j - E(\beta_j X_j) = \beta_j x_j - \beta_j E(X_j)$$

Avec cela on connaît à quel point chaque variable contribue à la prédiction. Si on additionne toutes les contributions des variables pour une instance, on obtient (voir chapitre 9 de [10]) :

$$\begin{aligned}\sum_{j=1}^{p} \phi_j(\hat{f}) &= \sum_{j=1}^{p} (\beta_j x_j - E(\beta_j X_j) \\ &= (\beta_0 + \sum_{j=1}^{n} \beta_j x_j) - (\beta_0 + \sum_{j=1}^{n} E(\beta_j X_j)) \\ &= \hat{f}(x) - E(\hat{f}(x))\end{aligned}$$

Il s'agit de la valeur prédite pour x moins la valeur moyenne prévu.
Cela ne marche bien que pour des modèles linéaires. Les valeurs Shapley donnent une solution pour calculer les contributions des variables pour les prédictions pour tout modèle du machine learning. Elle est définie par :

$$\phi_j(\hat{C}) = \sum_{S \subseteq \{x_1,\dots,x_p\} \setminus \{x_j\}} \frac{|S|(p - |S| - 1)!}{p!} (\hat{C}(S \cup \{x_j\} - \hat{C}(S))$$

où S est un sous-ensemble de variable utilisé dans le modèle, p est le nombre de variables, $\hat{C}$ est la fonction de paiement pour la coalition des joueurs. Pour obtenir ϕ_j, il faut donc calculer pour chaque coalition S dans laquelle le "joueur" x_j n'apparaît pas la différence de gain $\hat{C}(S \cup \{x_j\}) - \hat{C}(S)$. Cela permet de comparer le gain obtenu de la coalition avec et sans ce "joueur", afin de mesurer son impact lorsqu'il collabore avec cet ensemble de joueurs. Si cette différence est positive, cela signifie que le joueur x_j contribue positivement à cette coalition, et à l'inverse si la différence est négative, cela indique que le joueur pénalise le groupe, enfin si la différence est nulle cela signifie que le joueur n'apporte rien à ce groupe. On calcule ensuite la moyenne de ces écarts sur toutes les coalitions dans laquelle le joueur x_j intervient.

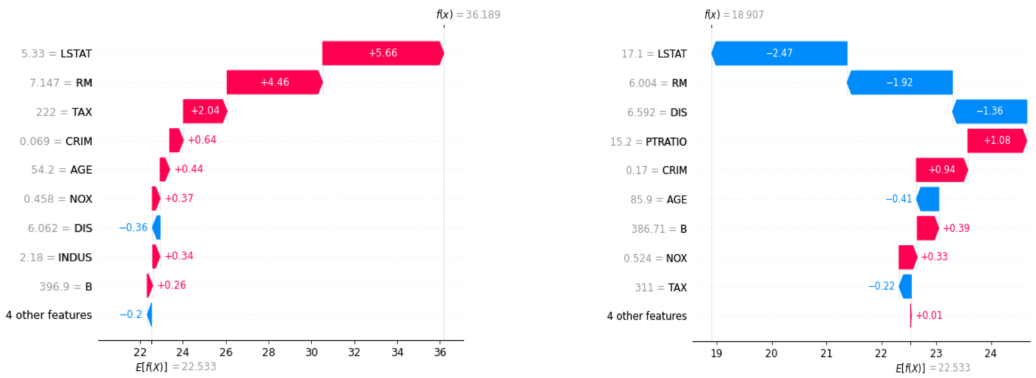


Figure 4.2.1 – valeurs shapley de deux instances du jeu de donnée boston

Cette illustration sur le jeu de données de Boston en figure 4.2.1 montre que les variables contribuant à une prédiction plus haute sont affichées en rouge, tandis que celles qui contribuent à une prédiction plus basse sont en bleu.
La valeur shapley est la seule méthode d'attribution qui satisfait les propriétés suivantes (chapitre 9 [10]) :

- **Efficience** :
$$\sum_{j=1}^{p} \phi_j = C(X) - C(\varnothing)$$
c'est-à-dire la somme des parts de chaque joueurs doit être égale au gain total (C($\varnothing$) est presque toujours nul).

- **Symétrie** :
$$C(S \cup \{x_j\}) = C(S \cup \{x_i\})$$
pour tous $S \subseteq \{x_1, ..., x_p\} \backslash \{x_i, x_j\}$ alors,
$$\phi_i(C) = \phi_j(C)$$
Cela veut dire que si deux joueurs contribuent de la même façon dans toutes les caractéristiques dans lesquelles ils apparaissent, leurs part doivent être égales.

- **Dummy** :
$$C(S \cup \{x_j\} = C(S)$$
pour tous $S \subseteq \{x_1, ..., x_p\}$ alors,
$$\phi_j = 0$$
Cela veut dire que si toute les coalitions dans lequel un joueur est présent ont le même gain avec et sans lui, alors la part de ce joueur est nulle.

- **Additivité** :
$$\phi_j(C_1 + C_2)) = \phi_j(C_1) + \phi_j(C_2)$$

Ainsi, les valeurs Shapley ont des avantages tels que l'efficience et le fait qu'il est époustouflant d'expliquer une prédiction comme un jeu joué par les valeurs des variables, Cependant les valeurs Shapley nécessitent beaucoup de temps de calcul car il y'a 2^p coalitions possible de valeurs d'entité et elles peuvent mal être interprétées.

4.2.2 Shap Importance

Les valeurs SHAP complètent les valeurs Shapley et offrent un haut niveau d'interprétabilité pour les modèles. SHAP (Shapley Additive Explanation) est une méthode pour expliquer les prédictions individuelles et est basée sur le jeu des valeurs Shapley théoriquement optimal. Comme précédemment, SHAP explique également la prédiction d'une instance x en calculant la contribution de chaque variable dans la prédiction. Une innovation apporté par SHAP est que l'explication des valeurs Shapley est représentée comme un modèle linéaire ([13]). SHAP spécifie l'explication comme suit :

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

où g est le modèle d'explication (linéaire), $z' \in \{0,1\}^M$ est le vecteur de coalition, M est la taille maximale de la coalition, et $\phi_j \in \mathcal{R}$ est la valeur Shapley de la variable X_j.
Dans le vecteur de coalition, une entrée de 1 signifie que la valeur de la variable correspondante est "présente" et 0 qu'elle est "absente".

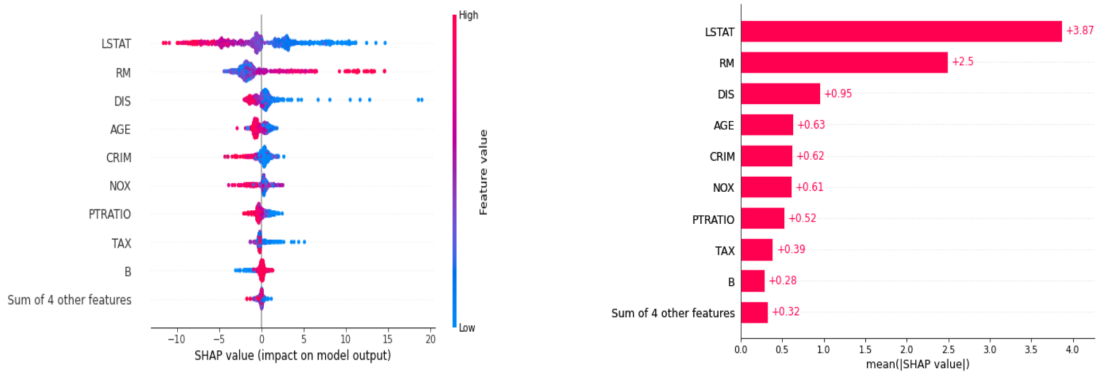


Figure 4.2.2 – Graphique récapitulatif SHAP du jeu de données de boston

La figure de gauche combine l'importance des variables avec les effets des variables. Chaque point de cette figure est une valeur Shapley pour une entité et une instance. La position sur l'axe Y est déterminée par la variable et sur l'axe X par la valeur Shapley. La couleur représente la valeur de la variable de la plus faible à la plus élevé. Nous avons donc une idée de la distribution des valeurs de Shapley par entité. Les variables sont classées en fonction de leur importance.
SHAP fournit également deux innovations par rapport aux valeurs Shapley (voir chapitre 9.6 de [10]) :

- **KernelShap** : elle est une approche alternative d'estimation basé sur le noyau pour les valeurs Shapley. Cependant elle est lente et ignore la dépendance des variables, ce qui la rend peu pratique lorsqu'on souhaite calculer les valeurs Shapley.
- **TreeShap** : elle est une approche d'estimation efficace pour les modèles arborescents tel que GBM, LigthGBM, XGBoost,..TreeShap est donc une variante de SHAP pour des modèles du machine Learning basés sur les les arbres.

Données utilisées pour calculer l'importance des variables

Une question qui se pose lors du calcul de l'importance des variables est de savoir si elles doivent être calculé sur les données d'entraînement ou sur les données de test. Pour la stratégie de "permutation importance", l'importance des variables reposent sur l'erreur du modèle, or les estimations d'erreur de modèle basées sur les données d'entraînement sont inutiles car lorsqu'on mesure l'erreur (ou les performances) du modèle sur les mêmes données sur lesquelles le modèle a été entraîné , la mesure est généralement trop optimiste car le modèle semble fonctionner beaucoup mieux que la réalité ce qui peut conduire au surajustement. On peut donc dire qu'à priori pour ces méthodes, il faut utiliser les données de test. Cependant, il existe également des arguments en faveur des données d'entraînement. En effet, l'importance des variables basée sur les données d'entraînement indiquent quelles variables sont importantes pour le modèle dans le sens où elles en dépendent pour

faire des prédictions. Breimann et Cutler décrivent permutation importance en passant un ensemble de validation pour le calcul du score, mais dans la pratique, on utilise le plus souvent l'ensemble de test afin de calculer l'importance des variables ([10]).

Chapitre 5

Algorithmes pour la sélection des variables "all relevant"

La sélection des variables est un processus très important qui implique une réduction de la dimension de l'espace d'entrée en sélectionnant le sous-ensemble des variables les plus pertinentes parmi toutes les variables.
Sur la base des différents problèmes de sélection des variables, il a été implémenté plusieurs algorithmes pour la recherche des caractéristiques "all relevant". Ces algorithmes pour la plupart évaluent la pertinence des variables dans le système d'information en comparant la mesure d'importance fournie par un classificateur (qui le plus souvent est basé sur un arbre de décision) pour les variables d'origines avec celle obtenu pour les variables aléatoires artificiellement ajoutés ([14]).
Plusieurs algorithmes pour all relevant ont été implémentés dont Boruta et BoostARoota qui se sont révélés très prometteurs car fournissant des performances supérieures par rapport aux classificateurs d'origine, mais présentant quelques lacunes notamment le temps d'exécution. Nous allons donc étudier dans le cadre de ce mémoire trois nouveaux algorithmes pour All relevant à savoir Leshy, BoostAGroota et GrootCV qui sont contenus dans le package ARFS et qui sont des algorithmes principalement basés sur ligthGBM et permettant encore plus d'améliorer les performances et la précision des deux méthodes précédentes.
Nous allons tous d'abord présenter deux méthodes alternatives pour all relevant à savoir Boruta et BoostAroota, ensuite nous présenterons les méthodes implémenté dans le package ARFS, qui sont des améliorations des deux méthodes précédentes.

5.1 Boruta et BoostAroota

5.1.1 Boruta

Boruta est un algorithme "Wrapper" alternatif pour la recherche des variables all relevant. L'algorithme du Random Forest peut généralement être exécuté sans réglage des paramètres et donne une estimation numérique de l'importance de chaque variable. Le Random Forest est une méthode d'ensemble dans laquelle la tâche de classification ou de régression est effectuée par vote de plusieurs apprenants faibles non biaisés (ici des arbres de décisions).
L'algorithme de Boruta évalue la pertinence des variables dans le système en comparant

la mesure d'importance fournie par le Random Forest pour les variables aléatoires artificiellement ajoutés (voir [14]). Ainsi, cet algorithme tente de capturer les variables importantes qui sont présentes dans notre jeu de données par rapport à la variable cible en comparant la pertinence des variables réelles avec celle des variables d'ombres. L'algorithme fonctionne comme suit ([15]) :

1. Créer des nouvelles copies de toutes les variables indépendantes du jeu de données.
2. Mélanger ces variables nouvellement ajouté afin de supprimer la corrélation avec la variable cible. Ces variables dupliquées et mélangées sont appelé "variables d'ombres".
3. Exécuter un classificateur de Random Forest sur les données étendues avec les variables d'ombres aléatoires puis classer ces variables en utilisant une mesure d'importance (la mesure par défaut est Gini impurity sur Python) afin d'évaluer l'importance des variables.
4. Calculer le Z-score qui est la moyenne de la perte de précision divisée par son écart type.
5. Trouver le Z-score maximum parmi toutes les variables d'ombre qu'on note MZSA.
6. Marquer les variables comme "sans importance" lorsqu'elles ont une importance nettement inférieure à MZSA et les supprimer définitivement du processus.
7. Marquer les variables comme "importante" lorsqu'elles ont une importance relativement plus élevée que MZSA.
8. Répéter la procédure jusqu'à ce que l'importance soit attribuée à tous les attributs ou jusqu'à ce que l'algorithme ait atteint une limite précédemment définie des exécutions du Random Forest.

Boruta est ainsi l'une des meilleures méthodes de sélection des variables pertinentes existante car elle fournit des résultats précis et stable ([15]). Cependant elle utilise par défaut le mécanisme de réduction moyenne des impuretés et nous savons que ce mécanisme peut être biaisé. Il est possible d'améliorer Boruta en ce qui concerne la complexité de calcul et la métrique qu'elle utilise pour classer des variables. ´

5.1.2 BoostAroota

Les processus de recherche tel que Boruta se sont révélés prometteurs car fournissant des performances supérieures avec la Random Forest, mais présentant des lacunes notamment un temps de calcul très lent pour des données de grande dimension, et du fait qu'il fonctionne mal sur d'autres algorithmes tel que le Boosting.
BoostARoota est un algorithme de sélection des variables pertinentes qui est basé principalement sur XGboost dont l'esprit est similaire à Boruta, mais utilisant XGBoost comme modèle de base plutôt que Random Forest ce qui lui permet de s'exécuter en une fraction de temps nécessaire à Boruta et elle offre des performances supérieur sur une variété d'ensemble de données ([16]). Le principe est presque similaire à celui de Boruta, cependant BoostARoota adopte une approche légèrement différente pour la suppression des variables. L'algorithme fonctionne comme suit [16]) :

1. Encoder l'ensemble des variables.
2. Effectuer une double copie de toutes les variables de l'ensemble d'origine.
3. Mélanger de manière aléatoire les nouvelles variables créées.
4. Exécuter un classificateur de XGBoost dix fois sur l'ensemble de données. On exécute dix fois afin de lisser le bruit aléatoire, ce qui donne des estimations plus robustes de l'importance et le nombre de répétitions est un paramètre qui peut changer.
5. On obtient ainsi les valeurs d'importance pour chaque variable. Il s'agit d'une métrique d'importance simple qui résume le nombre de fois où la variable particulière à été divisé dans l'algorithme XGBoost.
6. Calculer ensuite la "coupure" : c'est-à-dire la valeur moyenne de l'importance des variables pour toute les variables d'ombres et la diviser par quatre, afin de rendre la suppression des variables plus difficile.
7. Supprimer les entités dont l'importance moyenne sur les dix itérations est inférieure à la limite spécifié dans (6).
8. Recommencer la procédure à partir de (2) jusqu'à ce que le nombre de variables supprimé soit inférieur à 10% du total.
9. La méthode renvoie les variables restantes une fois terminé.

5.2 Le module Python ARFS

Trois nouvelles méthodes de sélection ont été développées à savoir Leshy (une évolution de Boruta), BoostAGroota (une évolution de BoostARoota) et GrootCV. Ces méthodes ont été implémentées dans le module "ARFS", compatible avec sklearn et fonctionnant aussi bien pour les problèmes de régression que pour les problèmes de classification ([17]). Ce module fournit également un module pour le pré-filtrage des données (colonnes avec beaucoup de valeurs manquantes, avec variance nulle, avec forte corrélation, avec aucune puissance prédictive ou faible puissance prédictive).
Nous allons maintenant présenter les méthodes de sélections des variables contenues dans ce module qui sont des alternatives pour le problème all relevant, et dans le chapitre suivant nous appliquerons ces méthodes sur différent type de données artificielles et réelles afin de pouvoir leur comparer, ce qui est le coeur même de ce mémoire.

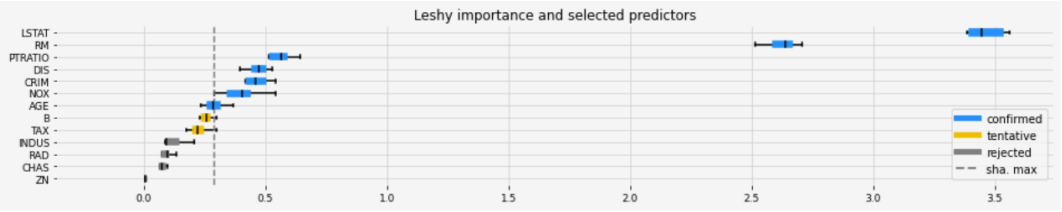
5.2.1 Leshy

Leshy est une nouvelle méthode de sélection des variables all relevant qui est une évolution de Boruta. Leshy apporte les modifications suivantes par rapport à la méthode originale de Boruta ([17]) :

- L'utilisation de ligthGBM comme modèle par défaut ce qui permet d'accélérer d'un ordre de grandeur le temps de fonctionnement.
- La détection et l'encodage des variables catégorielles. Leshy permet également d'utiliser CatBoost comme modèle.

- Elle autorise l'utilisation de "sample_weight" pour introduire certains paramètres qu'on connaît tel que l'exposition au risque (régression de Poisson) et le nombre de sinistre (régression de Gamma).
- La prise en charge de trois métriques d'importance différente à savoir Gini impurity, Shap importance et Permutation importance.

Leshy est donc une version améliorée de Boruta qui est plus précise, plus rapide, pouvant utiliser n'importe quel classificateur basé sur un arbre et incorporant trois métriques d'importances différentes.



<Figure size 432x288 with 0 Axes>

Figure 5.2.1 – représentation de l'importance des variables du jeu de données de Boston grâce à la méthode Léshy utilisant ligthGBM comme modèle et Shap importance comme métrique d'importance pour chaque variable.

La figure ci dessus nous donne une illustration de la méthode Leshy sur le jeu de données de Boston. On a le classement de toute les variables de la plus importante à la moins importante. Les variables marqué en bleu sont celles qui sont les plus pertinentes pour la prédiction du prix des logement de Boston, et celles en grise sont celles qui sont jugées non pertinente pour la prédiction.

5.2.2 BoostAGroota

BoostAGroota est également une nouvelle méthode de sélection des variables qui est quant à elle une évolution de BoostARoota. Cette nouvelle méthode apporte des modifications suivantes par rapport à BoostAroota ([17]) :

- BoostAroota utilise XGBoost comme modèle par défaut. BoostAGroota utilise par contre ligthGBM par défaut et permet également d'utiliser n'importe quel modèle basé sur des arbres de décisions.
- BoostAGroota utilise la métrique "Shap Importance" pour déterminer l'importance des variables tandis que BoostAroota utilise la métrique "Gini Impurity".
- BoostAGroota permet de gérer directement les prédicteurs catégoriques.
- Elle autorise l'utilisation de "sample_weight" pour la régression de Poisson ou toute autre application nécessitant une pondération.

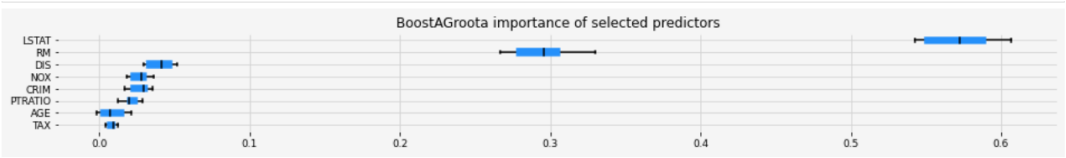


Figure 5.2.2 – Représentation de l'importance des variables les plus utiles grâce à la méthode BoostAGroota utilisant ligthGBM comme classificateur.

Cette figure ci-dessus nous donne une illustration de la méthode BoostAGroota sur le jeu de donnée de Boston. On a juste ici le classement de toute les variables les plus pertinentes pour la tâche de prédiction.

5.2.3 GrootCV

Il s'agit d'une nouvelle méthode pour la sélection des variables all relevant et qui est indépendante de toute autre méthode existante.
Elle à la particularité qu'elle n'utilise que le modèle du ligthGBM du fait de sa rapidité et de sa précision. Cette méthode utilise la cross validation pour déterminer l'importance des variables, et donc d'évaluer l'importance des variables sur différente partie de notre jeu de données permettant ainsi de "lisser le bruit" lorsqu'on estime cette importance ([17]). De plus la métrique d'importance utilisée est "Shap importance" pour la détermination de l'importance et elle n'est pas basée sur un pourcentage de colonne devant être supprimé. Ajouté à cela, GrootCV incorpore une méthode élégante pour le tracé des variables importantes.

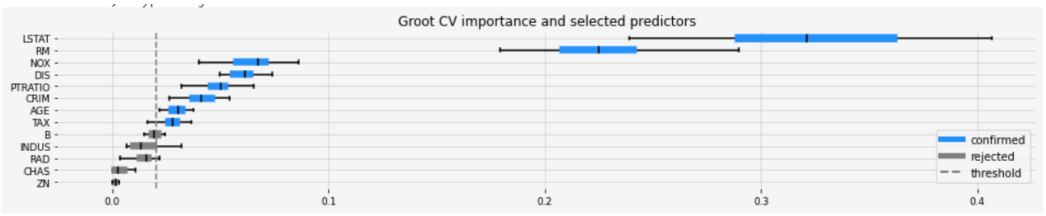


Figure 5.2.3 – Représentation de l'importance des variables du jeu de données de Boston grâce à la méthode GrootCV.

La figure ci-dessus nous donne une illustration de la méthode GrootCV sur le jeu de donnée de Boston. Comme pour les méthodes précédentes, on a le classement de toutes les variables de la plus importante à la moins importante. Les variables marquées en bleu sont celles qui sont les plus pertinentes pour la prédiction du prix des logement de Boston, et celles en gris sont celles qui sont jugées non pertinentes pour la prédiction ceci ayant utilisé la technique de la cross validation .

Chapitre 6

Application des méthodes de ARFS

Ce chapitre permettra de mettre en pratique les différentes méthodes de sélection de variable de ARFS à savoir Leshy, BoostAGroota et GrootCV qui sont tous implémentées sur Python. Afin de réaliser cela, nous appliquerons ces méthodes sur deux jeux de données :

- Sur un jeu de données artificiel sans lien avec l'assurance où on aura le contrôle sur tous les aspects à étudier puisqu'on connaît exactement le processus qui génère les données et cela nous offre donc plus de certitude sur les conclusions qu'on tirera.
- Sur un jeu de données réel à savoir le French MTPL qui sera présenté ci dessous.

6.1 Principe du travail

Le module ARFS fournit une classe de base pour effectuer la sélection des variables pour le préprocessing. Cette classe est la classe "FeatureSelector" qui implémente différentes méthodes pour identifier les variables à supprimer ([17]) :

1. **identify_missing (missing_threshold)** : cherche les variables avec une fraction de valeur manquante au dessus de missing_threshold.
2. **identify_single_unique** : cherche les colonnes sans aucune variance.
3. **identify_high_cardinality (max_card)** : cherche les variables catégorielles avec plus de max_card valeurs uniques.
4. **identify_collinear (correlation_threshold,encode,method)** : cherche les variables colinéaires ou multicolinéaires basées sur un coéfficient de corrélation. Pour chaque pair de variables avec une correlation plus grande que correlation_threshold, garde juste une parmi ses variables.
5. **identify_zero_importance** : identifie les variables qui n'ont aucune importance par rapport à un GBM utilisant la métrique SHAP pour la sélection des variables.
6. **identify_low_importance (cumulative_importance)** : cherche les variables de faible importance dont le degré d'importance ne contribue pas à avoir une importance cumulé totale de cumulative_importance pour toutes les variables et donnée par un GBM. Les variables identifiées sont ceux qui ne sont pas souhaité.

Ainsi, nous étudierons quelques effets de ces différents points (colinéarités, valeurs manquantes,...) sur nos jeux de données, fait de façon individuelle, c'est-à-dire, on va isoler chacun des effets cités ci-dessus et voir si cela à une influence sur la sélection des variables lorsqu'on le considère par rapport à quand on ne le considère pas et nous tenterons d'expliquer les observations qui surviendront.

6.2 Application sur un jeu de données artificiel sans lien avec l'assurance.

6.2.1 Génération de la base de données

Afin de générer cette base de données, on s'inspire de Boruta_py ([18]). On génère un jeu de données de 1000 instances et 13 variables prédictives construites tel que :

- La variable à prédire est la variable "target".
- Les variables var0,..,var4 sont construites comme pertinente pour la variable target avec une grande corrélation positive ou négative avec target.
- Les variables var5,..,var9 sont construites comme non pertinente pour la variable target.
- La variable var10 est une variable sans aucune variance avec uniquement la valeur 1 sur ces lignes.
- La variable 'dummy' est une variable catégorielle avec une grande cardinalité.
- La variable var12 est une variable contenant beaucoup de valeurs manquantes.

La figure ci dessous nous donne un aperçu de la variable target en fonction des variables prédictives pertinentes et non pertinente (bruit) qui se comporte comme on l'a prévu lors de la construction du dataset :

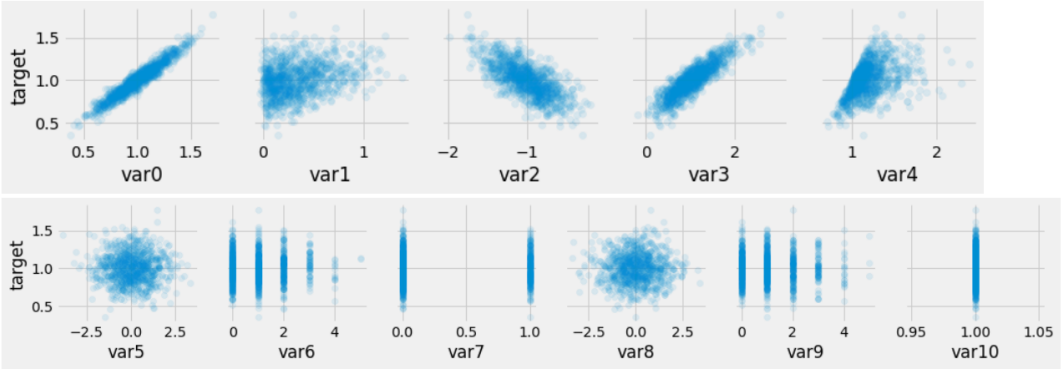


Figure 6.2.1 – Observation de la variable cible en fonction des variables prédictives pour le dataset artificiel

En fixant les paramètres (des fonctions listés ci-dessus) cumulative_importance= 0.95 , correlation_threshold=0.5, missing_threshold=0.2 et max_card=900 pour la première

étape du préprocéssing grâce au module "FeatureSelector" de ARFS, on obtient le récapitulatif suivant pour nos différentes variables :

	predictor	missing	single_unique	high_cardinality	collinear	zero_importance	low_importance
0	var0	1	1	1	1.0	1.0	1.0
1	var1	1	1	1	1.0	1.0	0.0
2	var2	1	1	1	0.0	1.0	1.0
3	var3	1	1	1	0.0	1.0	1.0
4	var4	1	1	1	0.0	1.0	1.0
5	var5	1	1	1	1.0	1.0	0.0
6	var6	1	1	1	1.0	1.0	0.0
7	var7	1	1	1	1.0	1.0	0.0
8	var8	1	1	1	1.0	1.0	0.0
9	var9	1	1	1	1.0	1.0	0.0
10	var10	1	0	1	NaN	NaN	NaN
11	dummy	1	1	0	NaN	NaN	NaN
12	var12	0	1	1	NaN	NaN	NaN

Figure 6.2.2 – Récapitulatif du résultat après l'étape du préprocessing grâce au module FeatureSelector de ARFS avec les paramètres fixés comme ci dessus.

Dans ce tableau récapitulatif, les valeurs de 0 signifient que la variable doit être supprimé pour l'effet mentionné en colonne après notre étape du préprocessing, et les valeurs de 1 signifient que les variables ne doivent pas être supprimé.
De même grâce à la fonction "plot_feature_importance" on a la représentation de l'ordre d'importance de nos variables issue du GBM :

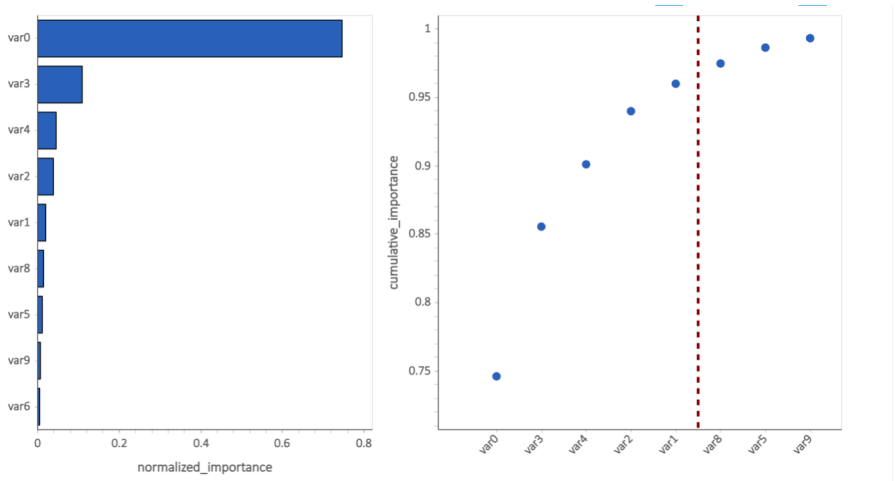


Figure 6.2.3 – Ordre d'importance des variables issue d'un algorithme du GBM.

La figure de gauche nous donne un aperçu de l'importance des variables issues du GBM par ordre d'importance, et la figure de droite est une représentation du cumule de l'importance de nos variables jusqu'à un seuil de 95%. Nous voyons qu'on a besoin de 05 variables

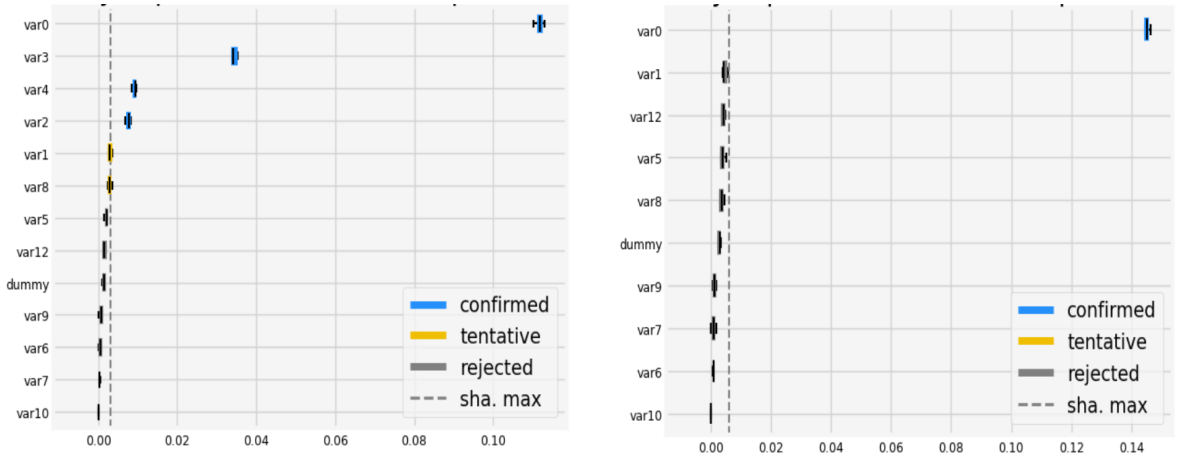
pour avoir un cumule d'importance d'au moins 95%. Après avoir fait cette étape du préprocessing, nous devons à présent appliquer nos méthodes sur le jeu de données et voir le comportement de ces méthodes lorsqu'on supprime chaque effet.

6.2.2 effet de la multicolinéarité

On regarde ici l'effet des variables multicolinéaires sur la sélection des variables. Après l'étape du préprocessing, la méthode 'identify_collinear' suggere de supprimer 03 variables (var2, var3, et var4) car ces variables ont une corrélation supérieur à 0.5 (qui a été fixé ci-dessus) avec la variable var0. Notons que la variable var0 est choisit de façon aléatoire parmi les 04 variables multicolinéaires, n'importe quel autre variable aurai été choisit par cette méthode. Nous appliquons maintenant les méthodes de sélection de variables de ARFS avec toutes les variables versus ces méthodes lorsqu'on ne considère pas les variables multicolinéaires en ne gardant qu'une seule d'elles afin de voir si l'importance dépend de la métrique d'importance utilisé (native, shap ou permutation importance).

Leshy

La figure ci dessous nous donne une application de la méthode Leshy utilisant XGBoost avec la métrique SHAP :

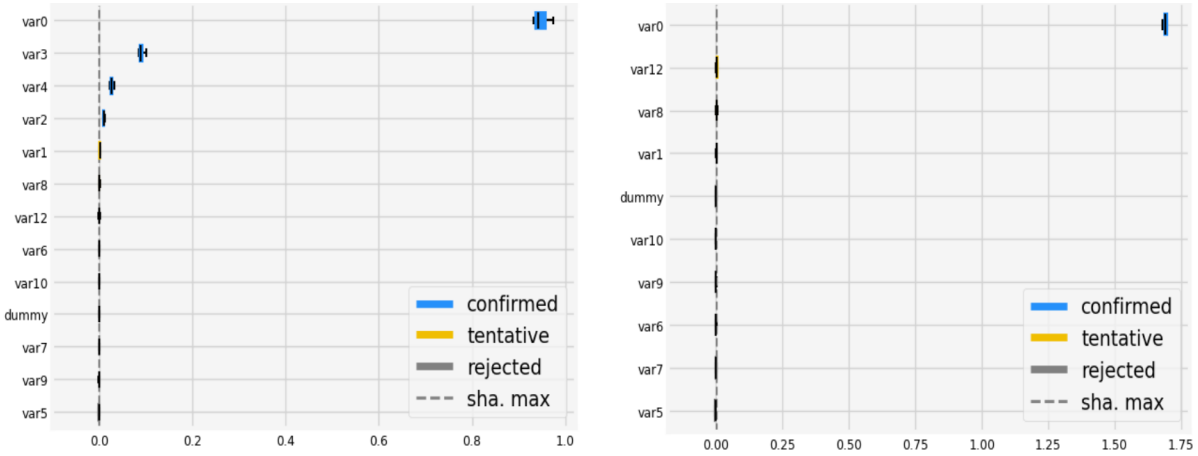


(a) avec toutes les variables (b) sans les variables colinéaires

Figure 6.2.4 – Leshy utilisant XGBoost comme régresseur et Shap comme métrique d'importance

On observe une différence lorsqu'il y'a tout les prédicteurs et lorsqu'on filtre les prédicteurs multicolinéaires en gardant un seul. L'importance du prédicteur var0 est moins élevé en présence de toutes les variables et cette importance augmente lorsque les prédicteurs multicolinéaire sont filtrés passant d'à peu près de 0.11 à 0.145 . La muticolinéarité est donc un problème car Shap étant une attribution linéaire d'importance des caractéristique, L'importance est divisé entre les prédicteurs multicolinéaires.

Le schéma ci dessous illustre leshy avec XGBoost, mais en utilisant plutôt la métrique permutation importance :

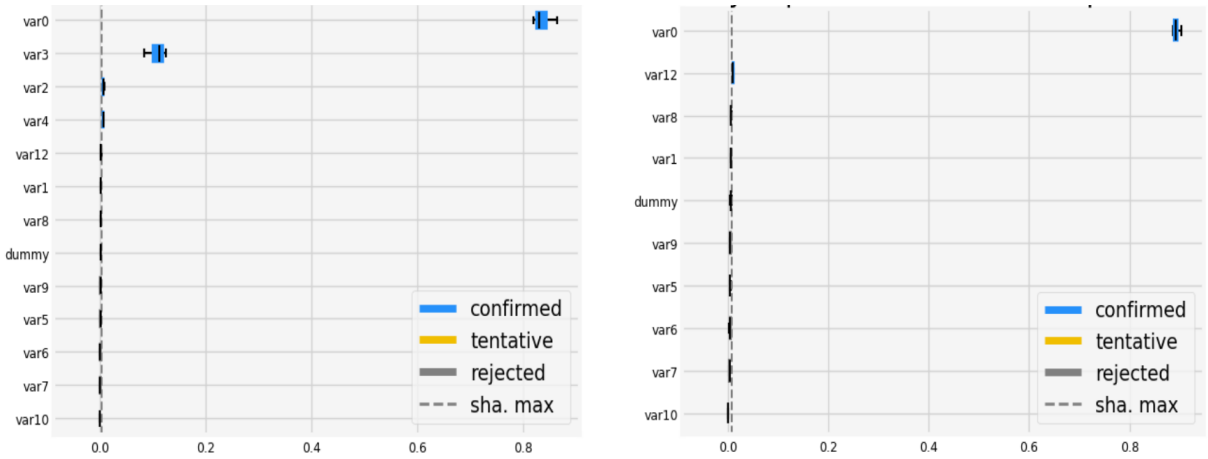


(a) avec toutes les variables (b) sans les variables colinéaires

Figure 6.2.5 – Leshy utilisant XGBoost comme régresseur et PIMP comme métrique d'importance

On fait les mêmes conclusions que ci dessus avec la métrique Shap, Il y'a une grande augmentation de l'importance du prédicteur var0 sans les prédicteurs multicolinéaires et qui reste quand même la seule variable la plus pertinente vu que toutes les variables multicolinéaires étaient jugées pertinentes pour la tâche. Ainsi avec Permutation importance, l'importance est également partagé entre les entités multicolinéaires lorsque toute ces variables sont dans le jeu de données.

En utilisant Gini impurity comme métrique,on a également les mêmes conclusions comme cela peut être observé sur la figure suivante :



(a) avec toutes les variables (b) sans les variables colinéaires

Figure 6.2.6 – Leshy utilisant XGBoost comme régresseur et Gini impurity comme métrique d'importance

En utilisant ligthGBM comme classificateur à la place de XGBoost (voir les figures à l'annexe C.1.1), nous avons les conclusions identiques que celle de XGBoost.
Nous savons à présent que la multicolinéarité partage l'importance entre les variables multicolinéaires. Modifions a présent les valeurs du paramètre 'correlation_threshold" afin de voir comment le niveau d'importance dépend du niveau de corrélation. Les figures suivantes illustrent Leshy avec XGBoost utilisant permutation importance comme métrique pour différent niveau de corrélation entre les variables :

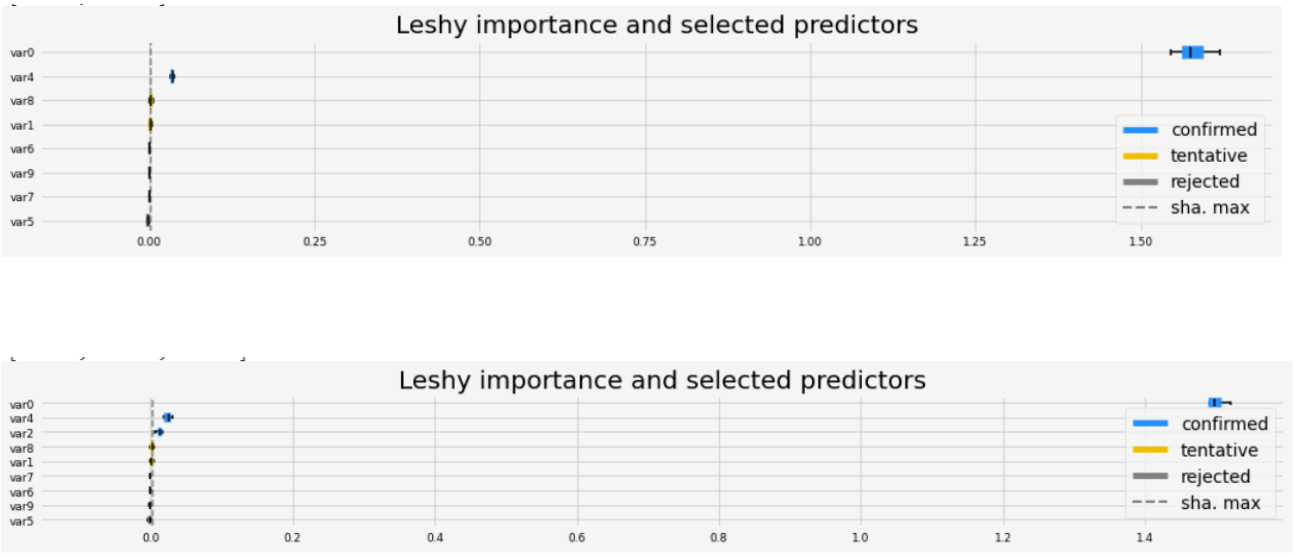


Figure 6.2.7 – Importance des variables avec différentes valeurs de corrélation utilisant permutation importance comme métrique

La figure du haut donne l'importance des variables pour correlation_threshold=0.6 et celle du bas donne l'importance pour correlation_threshold=0.75 . Pour une corrélation de 0.6, la variable var0 est colinéaire avec var2 et var3 tandis que pour une corrélation de 0.75, elle n'est colinéaire qu'avec la variable var3. Nous voyons à travers ces figurent que le partage de l'importance entre les prédicteurs multicolinéaire est fonction du niveau de la corrélation. plus la corrélation entre les variables est grande plus l'importance des variables diminue. Cela va de même également en utilisant les métriques SHAP et native, en présence de prédicteurs multicolinéaires, l'importance des variables est partagée entre ces prédicteurs et cela dépend du niveau de corrélation entre les variables (voir figure en annexe).

BoostAGroota

Contrairement à Leshy, BoostAGroota n'affiche que les variables qui ont été sélectionnées comme pertinentes après la sélection des variables, les variables jugées non pertinentes n'étant pas affichées.
Les figures 6.2.8 nous donnent une application de BoostAGroota utilisant XGBoost et la métrique SHAP, l'image du dessus est le résultat de l'application de BoostAGroota avec tous les prédicteurs, et en dessous est celle de boostAGroota avec les prédicteurs colinéaires filtrés. :

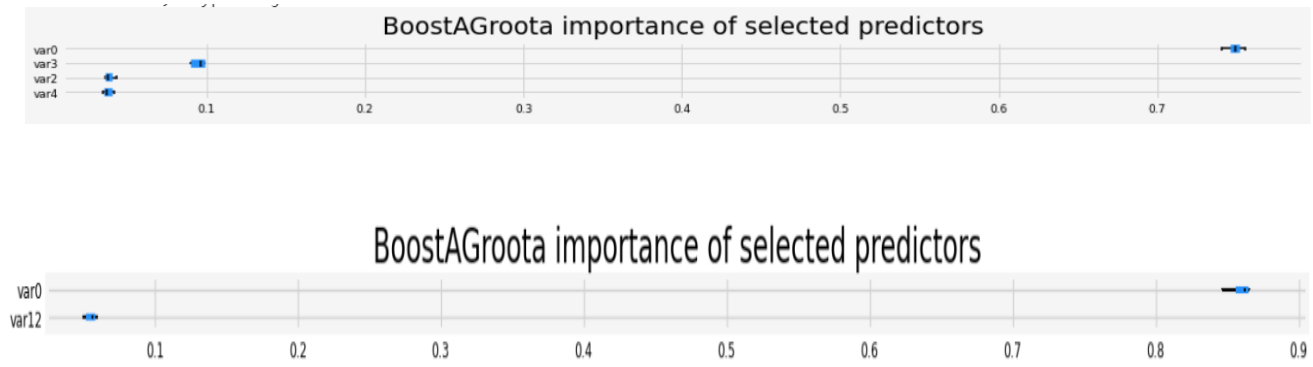


Figure 6.2.8 – BoostaGRoota utilisant XGBoost comme régresseur et SHAP comme métrique d'importance

Comme Leshy, BoostAGroota utilisant la métrique SHAP soufre du problème de multicolinéarité. L'importance de la variable var0 augmente lorsqu'il n y'a pas de prédicteur multicolinéaire avec var0. En présence des prédicteurs multicolinéaires, BoostAGroota partage l'importance (ici SHAP) entre les variables colinéaires. Notons aussi que la variable var12 est sélectionné lorsqu'on supprime l'effet des variables multicolinéaires alors qu'elle ne l'était pas lorsqu'on considérait toute les variables. Cela est due au fait qu'il doit exister une corrélation (inférieur à 0.5) entre la variable var12 et une autre variable du dataset qui était colinéaires à var0, donc en supprimant cette variable, elle a partagé son importance entre toutes les variables colinéaires avec elle.
En utilisant Permutation Importance comme métrique, on a les figures ci dessous :

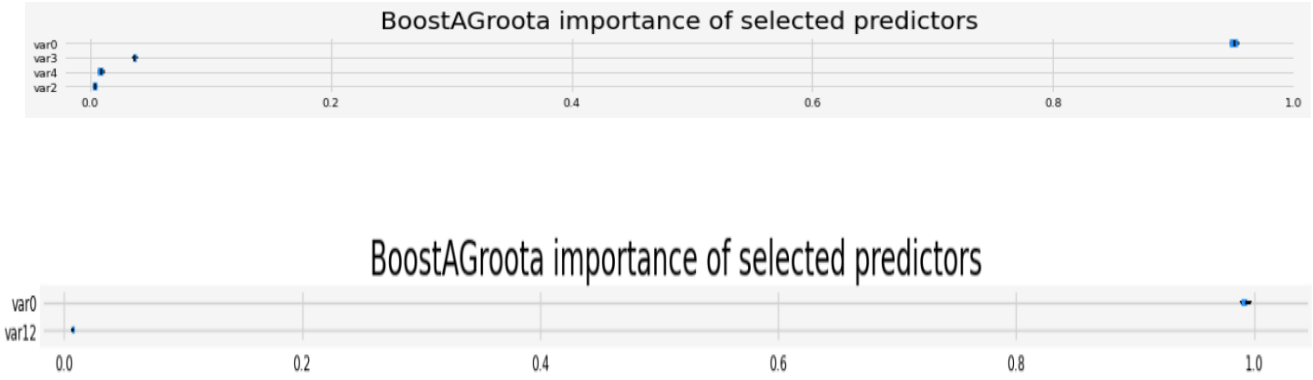


Figure 6.2.9 – BoostaGRoota utilisant XGBoost comme régresseur et Permutation importance comme métrique d'importance

Comme pour la métrique SHAP, on observe une augmentation de l'importance de la variable var0 lorsqu'on supprime la colinéarité due au fait que Permutation importance partage l'importance entre les variables multicolinéaires et la variable var12 est également sélectionnée alors qu'elle n'était pas classé avant.
En utilisant ligthGBM au lieu de XGBoost comme régresseur (voir l'annexe C.1.1), On aura les même observations qu'avec XGBoost.
Comme fait précédemment, augmentons les valeurs du paramètre 'correlation_threshold' pour voir si le niveau d'importance dépend du niveau de corrélation. Les figures suivantes

illustrent BoostAGroota avec XGBoost utilisant SHAP importance comme métrique pour différent niveau de corrélation entre les variables :

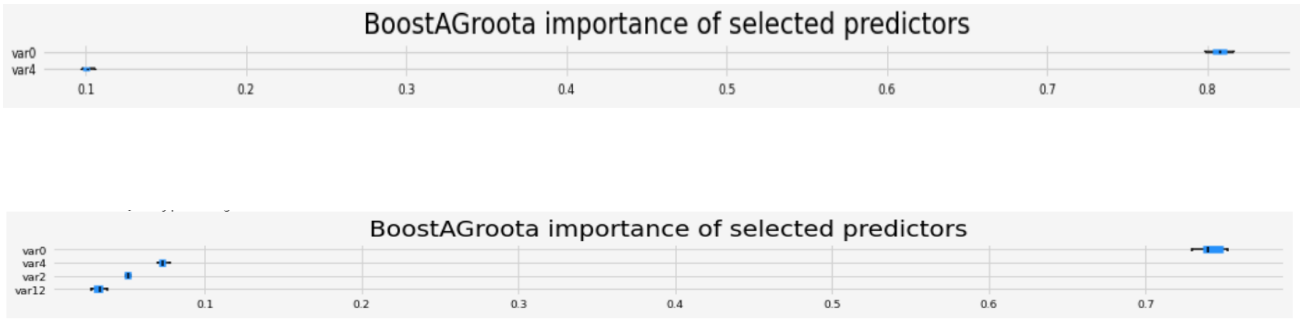


Figure 6.2.10 – Importance des variables avec différentes valeurs de corrélation notament 0.6 pour la figure du haut et 0.75 pour la figure du bas, utilisant permutation importance comme métrique.

La figure au dessus représente BoostAGroota sans les prédicteurs multicolinéaires lorsqu'on prend une corrélation de 0.6 (correlation_threshold=0.6), tandis que la corrélation pour la figure du dessous est de 0.75 .
De nouveau on observe que le niveau de corrélation modifie l'importance des variables. plus il y'a de variables colinéaires dans la prédiction, plus l'importance de la variable est réduite. Pour une corrélation de 0.6, var0 est multicolinéaire avec deux variables (var2 et var3) et donc son importance est plus élévé lorsqu'on supprime cette colinéarité. Cette importance est réduite pour une corrélation de 0.75 où ici var0 est colinéaire avec une seule variable. Ainsi donc,la colinéarité affecte BoostAGroota où l'importance d'une variable est fonction du niveau de corrélation choisi.

GrootCV

Comme présenté ci dessus, la méthode grootCV utilise uniquement ligthGBM comme classificateur de base ainsi que la métrique shap et elle utilise la cross validation pour déterminer l'importance des variables.
La figure 6.2.11 illustre grootCV avec tous les prédicteurs :

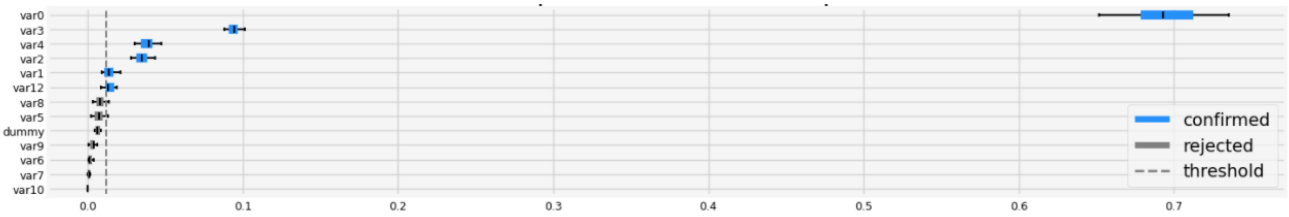


Figure 6.2.11 – GrootCV avec tous les prédicteurs

de même, pour une 'correlation_threshold=0.5', on a la figure 6.2.12 pour grootCV sans les prédicteurs multicolinéaires :

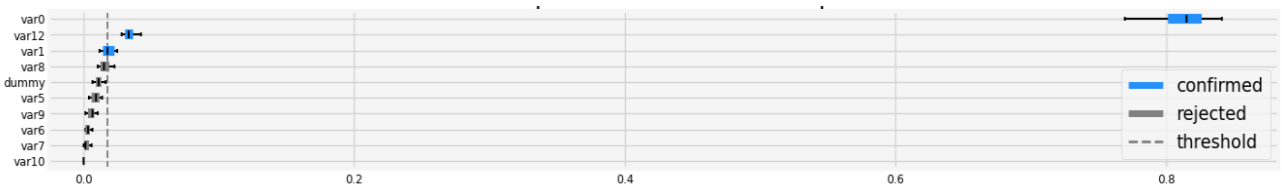


Figure 6.2.12 – GrootCV sans les prédicteurs colinéaires

Nous pouvons voir que sans les variables multicolinéaires, La variable var0 à une plus grande importance que lorsqu'on considère toute les variables. Ainsi grootCV est sensible également au problème de multicolinéarité comme les méthodes précédentes. Cette colinéarité affecte aussi le 'ranking'. En effet, lors de la sélection des caractéristiques avec toutes les variables, la variable var1 et plus pertinente que la variable var12 mais Lorsqu'on considère l'effet de la colinéarité, cet ordre est inversé du au fait que l'importance de var12 à augmenté.
L'attribut **cv_df** de GrootCV nous donne les différents niveau d'importance de chaque variable sur les différents folds de la cross validation. Nous pouvons donc calculer et étudier le comportement de la moyenne et de la variance des features importances sélectionnés et voir comment elles varient en fonction de l'intensité de la corrélation en comparant ceux ci lorsqu'on considère la multicolinéarité et lorsqu'on ne considère pas.
pour la variable var0, on obtient :

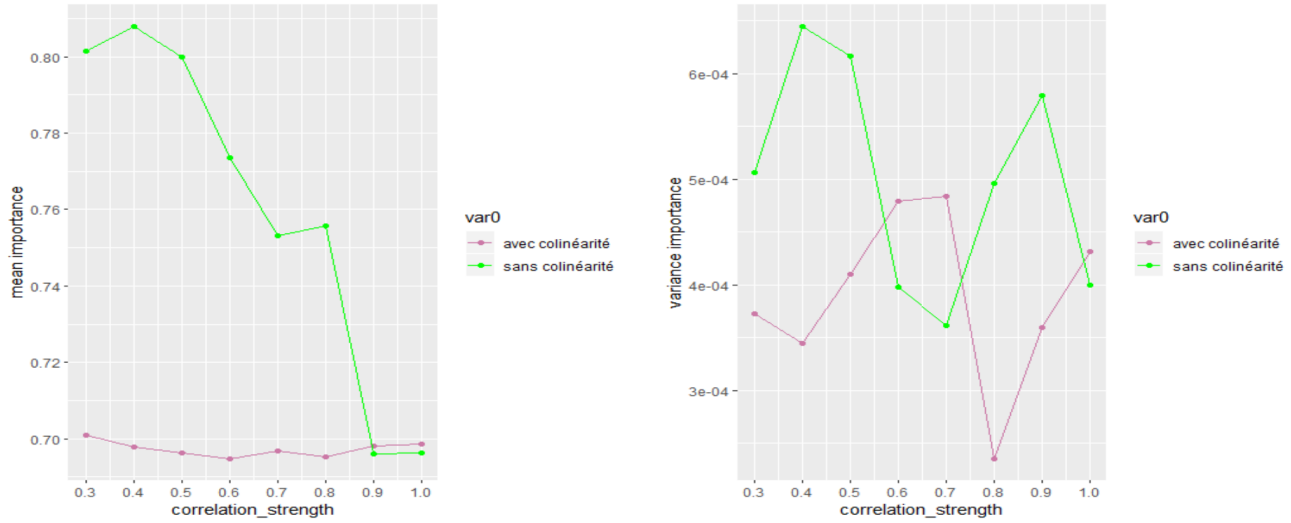


Figure 6.2.13 – étude de la moyenne (figure de gauche) et variance (figure de droite) d'importance de var0 en fonction du niveau de corrélation

On peut donc voir sur la première figure que comme pour les méthodes Leshy et BoostaGroota, GrootCV est très sensible à la corrélation. Plus la corrélation est grande, plus l'importance de la variable décroît ceci lorsqu'on ne considère pas les variables colinéaires à var0 avec le niveau de corrélation spécifié. En regardant l'évolution de la moyenne et la variance des importance en fonction du niveau de corrélation, on s'aperçoit que la colinéarité dégrade la sélection des variables car on observe clairement un grand écart lorsqu'on considère la colinéarité de lorsqu'on ne considère pas .
on a la même observation avec la variable var12 comme le montre les figures suivantes

pour la moyenne et variance des importances :

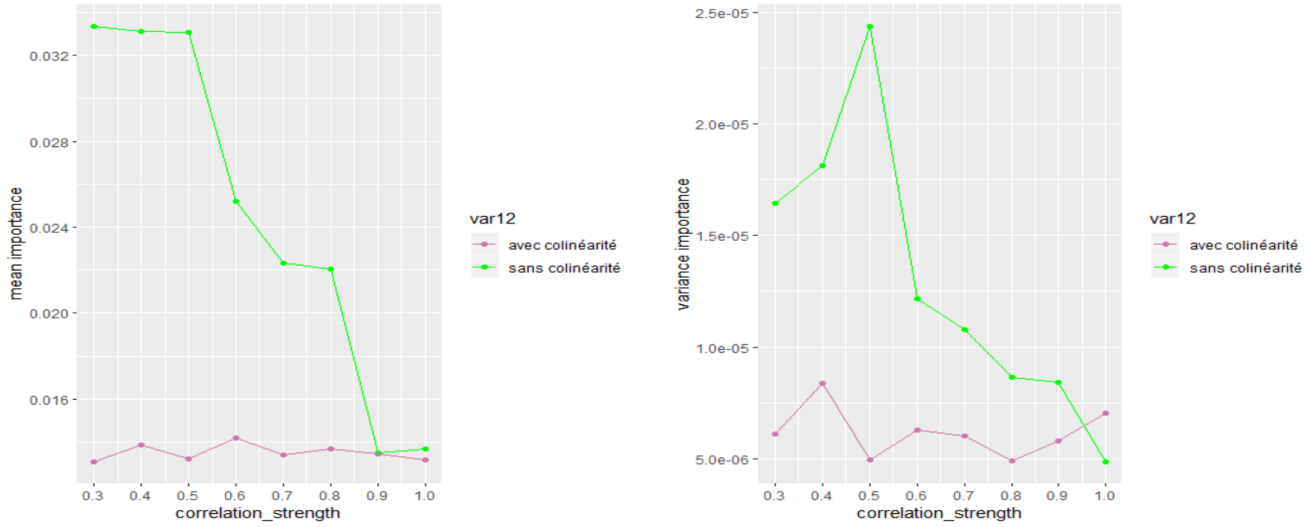


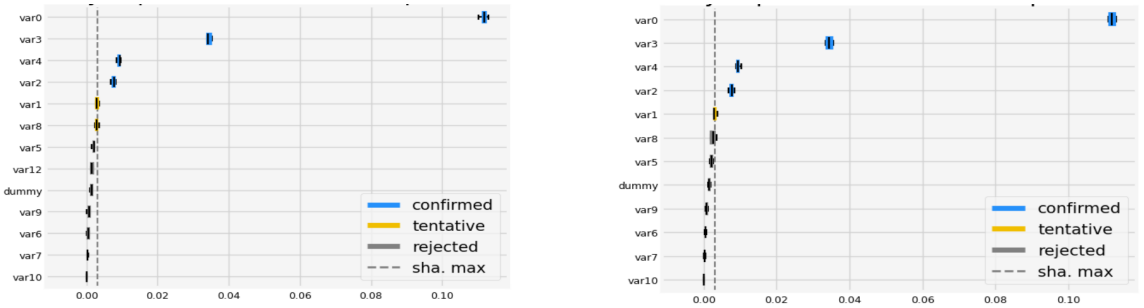
Figure 6.2.14 – étude de la moyenne (figure de gauche) et variance (figure de droite) d'importance de var12 en fonction du niveau de corrélation

On peut observer un écart assez significatif pour la moyenne ainsi que pour la variance des importances avec également une décroissance de l'importance moyenne lorsque le niveau de corrélation augmente, ce qui nous conforme dans le fait que la colinéarité dégrade la qualité de sélection des variables avec GrootCV.

6.2.3 effet des variables avec plusieurs valeurs manquantes

Lors de la génération du dataset artificiel, La variable var12 à été généré comme contenant beaucoup de valeurs manquantes. Nous vérifions à présent si la présence dans le jeu de données de telle variable peut avoir une influence lors de la sélection des variables pertinentes. Pour nos méthodes on va donc regarder la feature sélection avec les variables à plusieurs valeurs manquantes versus ceux sans ces variables.

Leshy

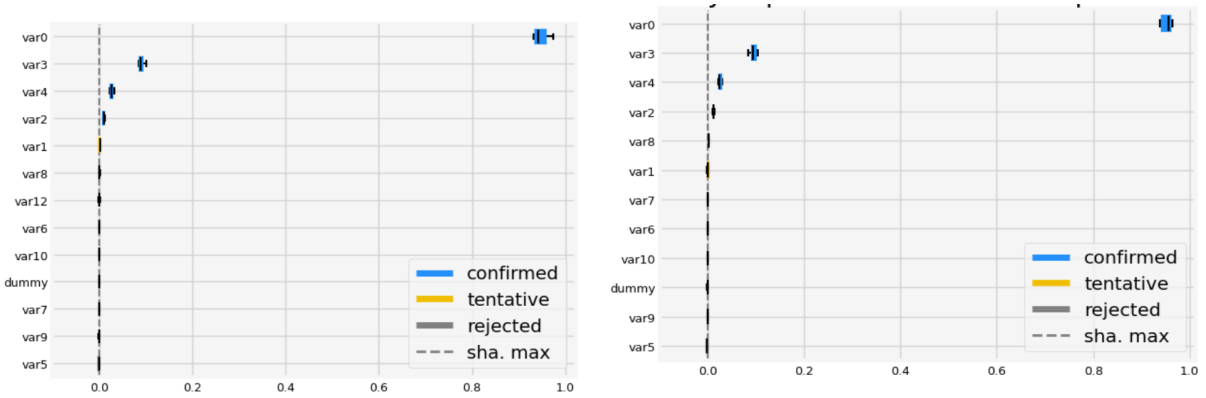


(a) avec toutes les variables (b) sans les variables avec valeurs manquantes

Figure 6.2.15 – Leshy avec xgboost et Shap comme métrique d'importance

La figure ci-dessus montre Leshy avec XGBoost et SHAP comme métrique d'importance.

Nous pouvons voir que cela n'a à priori aucune influence sur la sélection des variables. Il n'y a pas de modification du niveau d'importance ni du ranking (classement) de nos variables qu'on considère la variable avec plusieurs valeurs manquantes ou non.
En utilisant Permutation importance comme métrique, on a la même conclusion comme le montre la figure ci dessous :



(a) avec toutes les variables

(b) sans les variables avec des valeurs manquantes

Figure 6.2.16 – Leshy utilisant XGBoost comme régresseur et PIMP comme métrique d'importance

En utilisant Gini impurity comme métrique d'importance et en utilisant ligthGBM plutot que XGBoost comme régrésseur on aura les mêmes conclusions (voir l'annexe C.1.2) : Leshy ne souffre pas du problème de la présence de plusieurs valeurs manquantes lors de la sélection des variables pertinentes.

BoostAGroota

La figure 6.2.17 ci dessous est une application de BoostAGroota utilisant XGBoost comme regrésseur et SHAP comme métrique d'importance :

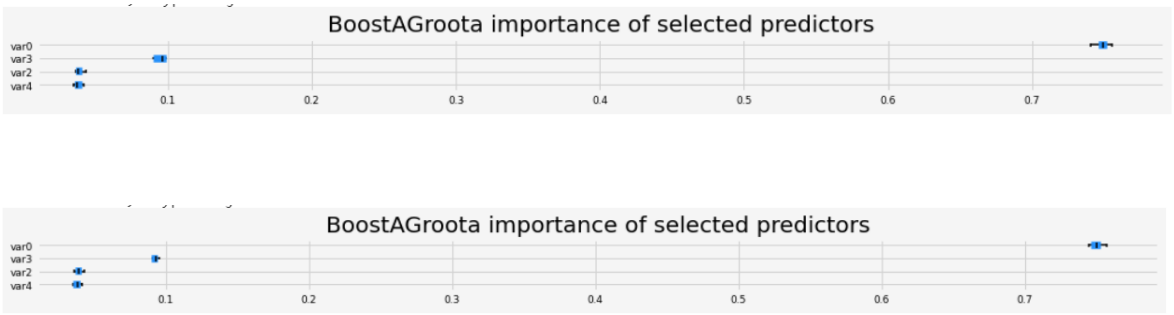


Figure 6.2.17 – BoostaGRoota utilisant XGBoost comme régresseur et SHAP comme métrique d'importance.la figure du haut est boostaGRoota avec toute les variables et celle du bas est boostaGRoota sans la variable var12.

La figure du dessus est celle avec toute les variables, et celle en dessous est celle sans les variables avec plusieurs valeurs manquantes.
Comme Leshy, avec l'utilisation de SHAP comme métrique d'importance, BoostAGroota ne semble pas subir l'influence des variables avec plusieurs valeurs manquantes. On observe pas de modification de l'importance d'une variable ni la modification de son ranking, ni la modification des variables pertinentes pour notre variable à prédire.

En utilisant Permutation importance comme métrique, on à l'illustration ci-dessous :

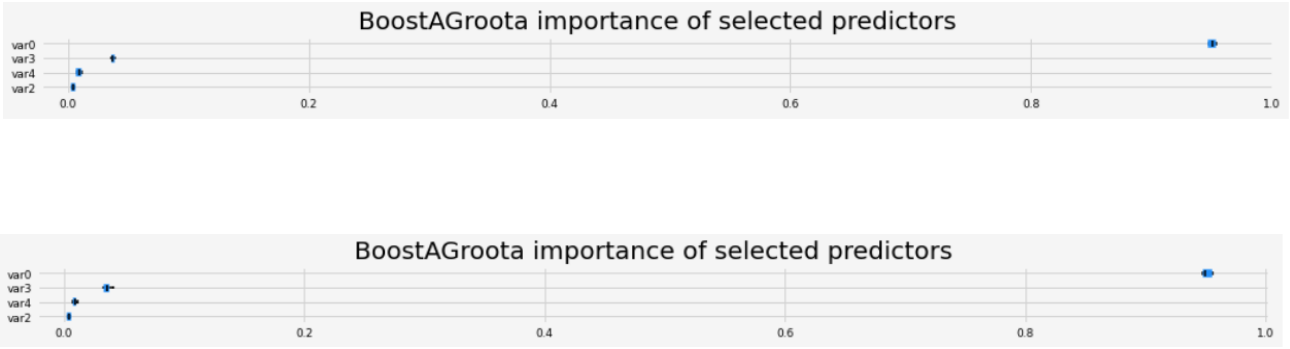


Figure 6.2.18 – BoostaGRoota utilisant XGBoost comme régresseur et Permutation importance comme métrique d'importance

On a la même conclusion que la métrique SHAP, en utilisant permutation importance,il n y'a pas de modification dans la sélection des variables pertinentes avec ou sans les variables avec une grande quantité de valeur manquante. De façon générale donc, la méthode boostAGroota ne semble pas souffrir du problème des valeurs manquantes.

GrootCV

En utilisant GrootCV sans les variables avec plusieurs valeurs manquante,on obtient la figure suivante :

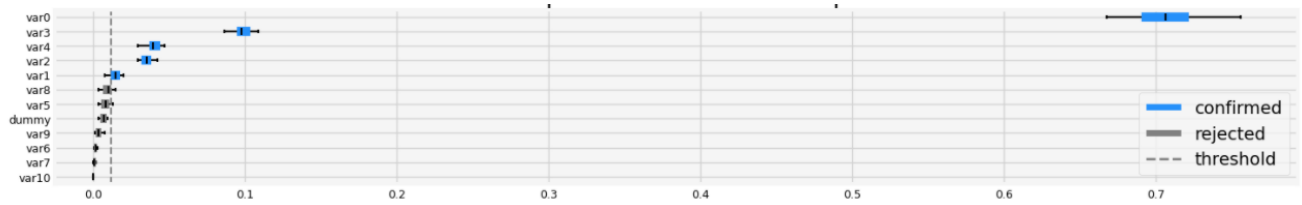


Figure 6.2.19 – GrootCV sans les variables avec plusieurs valeurs manquantes

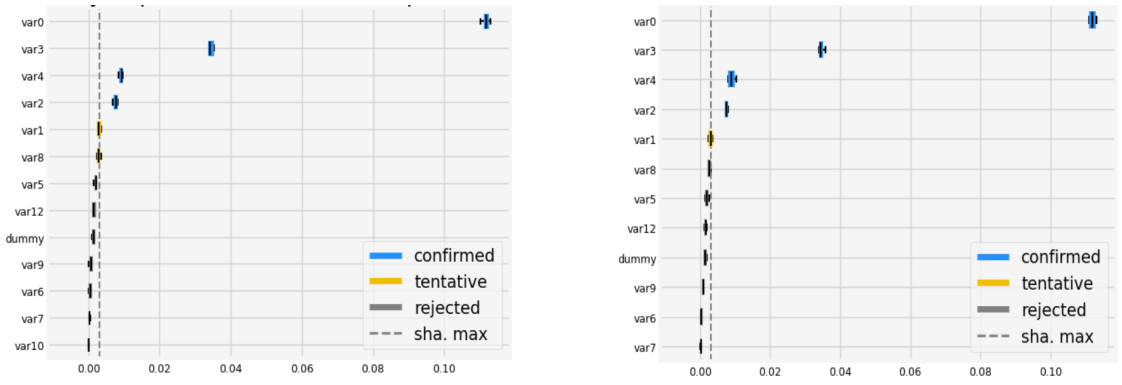
si on compare avec le cas où on considère toutes les variables (voir figure 6.2.11), il n y'a pas eu qualitativement de grande modification, Ainsi GrootCV ne souffre pas trop du problème des valeurs manquantes.

6.2.4 Effet des variables sans aucune variance

On va regarder ici l'influence de la variable **var10** qui a été simulé comme une variable avec une valeur unique sur ces lignes. Nous allons voir si cette variable à une influence sur la sélection des variables pertinentes en regardant comme précédemment nos méthodes sans et avec les variables sans aucune variance.

Leshy

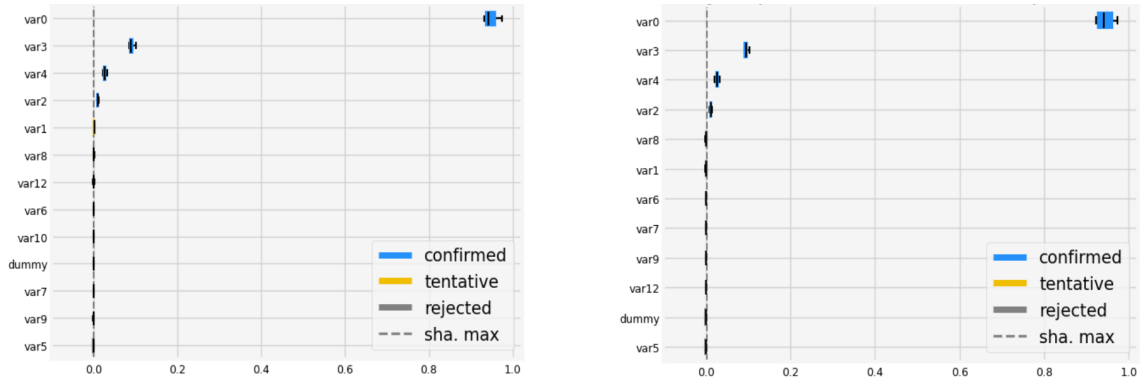
La figure 6.2.20 illustre Leshy avec xgboost et SHAP comme métrique d'importance, lorsqu'on supprime la variable avec une valeur unique sur ces lignes dans le jeu de données.



(a) avec toutes les variables (b) sans la variable sans aucune variance

Figure 6.2.20 – Leshy avec XGBoost et Shap comme métrique d'importance

Comme nous pouvons le remarquer, cette variable n'a pas un grand impact lors de la sélection des variables pertinentes. On observe pas une grande modification de l'importance de chaque variable, ou de leur classement dans la sélection des variables pertinentes lorsqu'on considère ou pas la variables var10. En utilisant Permutation importance plutôt que SHAP, on a la même remarque comme le montre l'illustration ci-dessous :



(a) avec toutes les variables (b) sans les variables à valeur unique

Figure 6.2.21 – Leshy avec XGBoost et PIMP comme métrique d'importance

En utilisant gini impurity comme métrique et en utilisant lightGBM au lieu de XGBoost comme régresseur, on aura le même conclusion : Leshy semble ne pas être affecté par les variables sans aucune variance, ces variables étant sélectionné comme non pertinente lors de la sélection des caractéristiques pertinentes.

BoostAGroota

Ci-dessous,une autre illustration de BoostAGroota utilisant XGBoost comme régresseur et SHAP comme métrique. La figure du dessus est celle avec toutes les variables et celle en dessous est celle sans la variable avec des valeurs uniques :

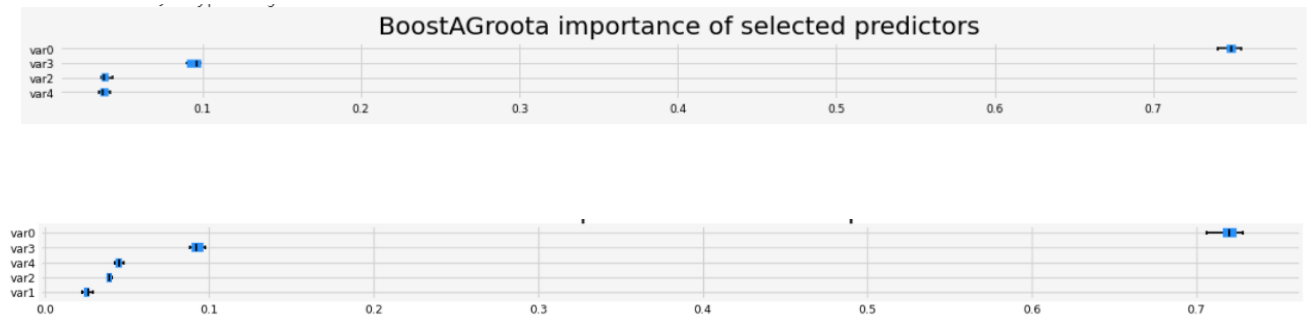


Figure 6.2.22 – BoostAGroota avec XGBoost et SHAP comme métrique d'importance.

Comme pour Leshy, on observe pas une grande modification de l'importance des variables globales. On observe néanmoins une modification du nombre de variable qui sont sélectionné comme importante, dont 04 variables lorsqu'on considère toute les variables du dataset et 05 variables lorsqu'on ne considère pas la variable var10. Cette variable semble ainsi cacher l'importance de var1 qui à l'étape du préprocéssing contribuait à avoir une importance cumulé de 95% pour un GBM (voir figure 6.3.1).
En utilisant PIMP plutôt que SHAP on a :

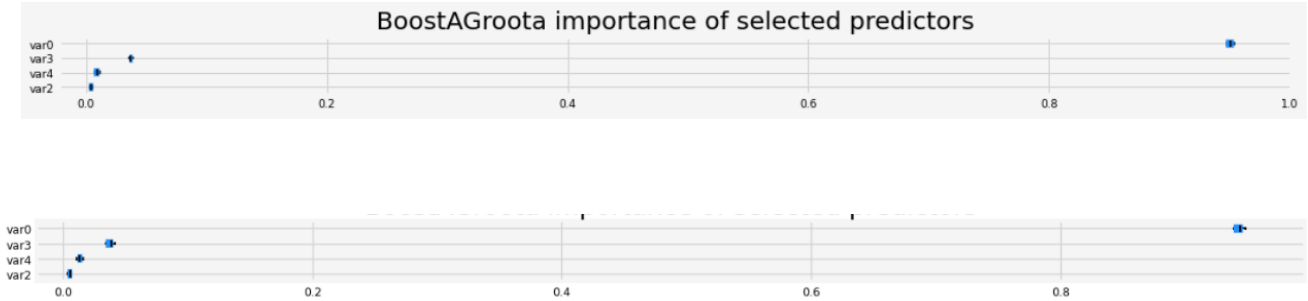


Figure 6.2.23 – BoostAGroota avec XGBoost et Permutation importance comme métrique

Avec permutation importance, il n y'a pas de grande modification que ce soit au niveau de la quantité d'importance de la variable ou encore du classement et du nombre de variable sélectionné. Permutation importance ne semble donc pas être affecté par les variables à valeur unique.

GrootCV

On a l'illustration suivante lorsqu'on utilise GrootCV sans la variable avec une valeur unique :

Figure 6.2.24 – GrootCV sans la variables avec une valeur unique

En comparaison avec le cas où il y'a toute les variables (voir figure 6.2.11),il n y'a pas eu de grande modification. Le niveau d'importance pour nos variable n'a pas changé, le classement et nombre de variable sélectionné n'ont pas été modifié également. Ainsi la méthode ne semble pas souffrir de l'influence des variables à valeur unique lors de la sélection des variables pertinentes.

6.3 Applications sur le jeu de données réel French TPLMD

6.3.1 Description de la base de donnée

Afin d'appliquer nos modèles sur un jeu de donné réel, nous nous servirons du jeu de donnée **French Motor Third Party Liability Claims**. Il s'agit d'un jeu de données où chaque échantillon correspond à une police d'assurance c'est à dire un contrat au sein d'une compagnie d'assurance et un particulier, et ce jeu de données est construit en groupant les tables FreMTPL2Freq (voir [19]) contenant le nombre de sinistre avec la table FreMTPL2sev (voir [20]) contenant le montant des sinistres pour le même numéro de police.
Ce jeu de données est constitué de 678 013 polices d'assurance et contient les caractéristiques de risque du preneur d'assurance et du véhicule assuré. Initialement, le jeu de données est constitué de 13 variables décrit comme ci dessous :

Variable	Label
ID number	Numero de la police (identifiant unique).
ClaimNb	Nombre de réclamations sur la police donnée.
Exposure	Exposition totale en unité d'année.
Area	Variable catégorielle représentant l'indicatif régional.
VehPower	Puissance du véhicule.
VehAge	Âge du véhicule en année.
DrivAge	Âge du conducteur (le plus courant) en année .
BonusMalus	Niveau du bonus malus compris entre 50 et 230 (avec niveau de référence de 100).
VehBrand	Variable catégorielle représentant la marque de voiture.
VehGas	Variable binaire : voiture Diesel ou Essence.
Density	Densité d'habitant au km^2 dans la ville de résidence du conducteur.
Region	Region en France (avant 2016).
ClaimAmount	Montant total de la réclamation pour la police donnée.

Table 6.3.1 – Les variables de la base de données

Pour des raisons pratique, Nous avons également créé les variables "Frequency" et "AvgClaimAmount" représentant respectivement pour chaque police la fréquence de sinistre (nombre total de sinistre enregistré par rapport à l'exposition) et le coût moyen des sinistres enregistrés (coût total des sinistres par rapport au nombre de sinistre).

6.3.2 Statistique descriptive de la base de données

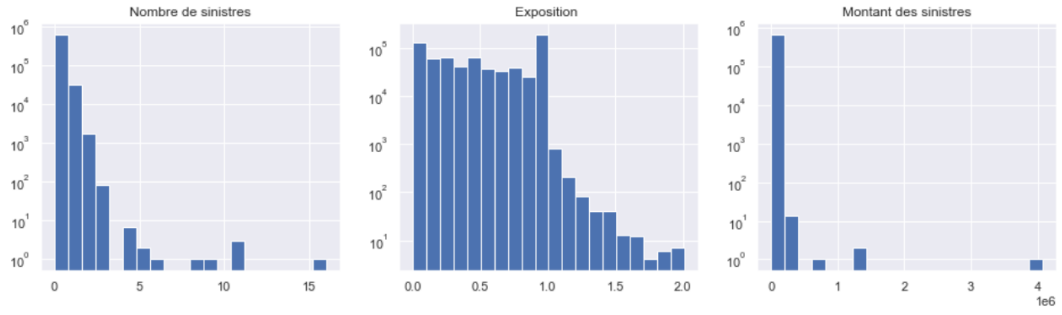


Figure 6.3.1 – Repartition du portefeuille en fonction du nombre de sinistre, de l'exposition et de la sévérité.

La figure ci dessus montre comment le nombre de sinistre, l'exposition au risque et la sévérité sont distribuées dans le portefeuille.
La majeure partie des assurés (93.3%) n'a déclaré aucun sinistre pendant leur période

d'exposition, un nombre substantiel (5.8%) en a déclaré 1 seul sinistre, et les autres assurés (0.35%) ont déclarés au moins 2 sinistres et au plus 15 sinistres. 46.9% des assurés ont été exposés pendant une année entière (365 jours), 52.7% ont été exposés pendant une durée inférieur à un an et ont donc soit racheté leur contrat, soit rejoint le portefeuille au cours d'une année d'enregistrement. La fréquence globale des sinistres, qui est calculé comme le ratio entre le nombre total des sinistres et l'exposition totale est égale à 10.07%. De même, la sévérité globale moyenne des sinistres du portefeuille, calculé comme le rapport entre le montant total des sinistres et le nombre total des sinistres est égale à 1 659.4431 euros.

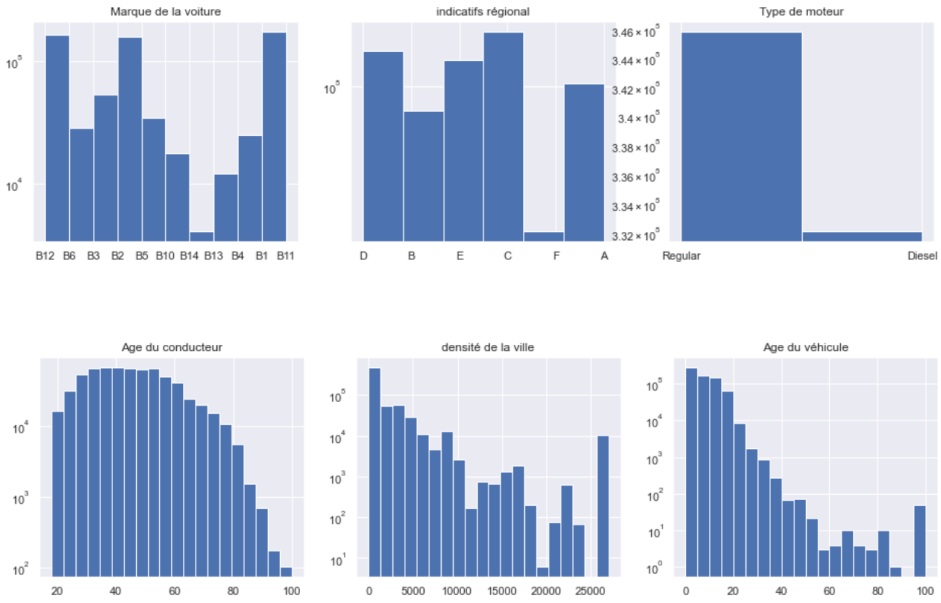


Figure 6.3.2 – répartition du portefeuille en fonction des facteurs de risque.

La figure 6.3.2 montre également la répartition du portefeuille au sein des différents facteurs de risque. Nous constatons que la plupart des voitures assurés ont un moteur de type Essence (52.4%) plutôt que Diesel. Une grande partie des assurés ont un âge compris entre 20 et 60 ans (81.8%), ce qui veut dire qu'il y'a peu d'anciens conducteurs dans le portefeuille et donc plus de risque de segmentation, et une grande proportion des assurés (58.4%) réside dans des villes dont la densité de la population est inférieur à 500 km^2.

Nous allons à présent étudier la sélection des variables grâce aux différentes méthodes de ARFS. Nous étudierons ici la feature sélection que pour la fréquence des sinistres. En effet, on peut extraire de façon générale plus d'information des fréquences que des montants des sinistres. Ainsi, les résultats obtenu à partir des fréquences des sinistres sont plus fiables que ceux obtenus à partir des montants des sinistres.

6.3.3 Sélection des variables pour la fréquence des sinistres

Grâce à la classe FeatureSelector de ARFS, en utilisant la variable "Exposure" comme "sample_weight", la variable "frequency" comme variable à prédire et en fixant les paramètres cumulative_importance= 0.95 , correlation_threshold=0.9, missing_threshold=0.1

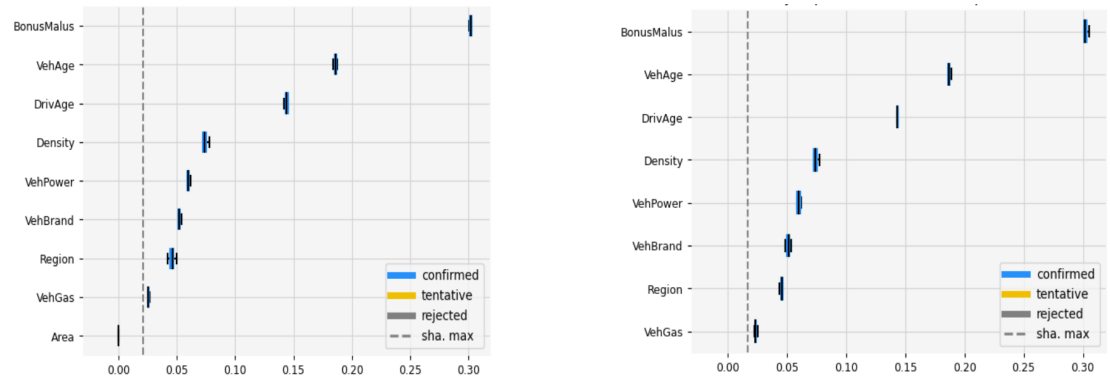
et max_card=2000 pour la premières étapes du préprocéssing grâce au module "FeatureSelector" de ARFS, on obtient le récapitulatif suivant pour nos différentes variables :

	predictor	collinear	missing	single_unique	high_cardinality	zero_importance	low_importance
0	VehPower	1	1	1	1	1	1
1	VehAge	1	1	1	1	1	1
2	Density	1	1	1	1	1	0
3	BonusMalus	1	1	1	1	1	1
4	Region	1	1	1	1	1	0
5	VehBrand	1	1	1	1	1	1
6	Area	0	1	1	1	1	0
7	VehGas	1	1	1	1	1	1
8	DrivAge	1	1	1	1	1	1

Figure 6.3.3 – Récapitulatif du résultat après l'étape du préprocessing

Nous pouvons remarquer que le jeu de données semble assez bien trié, notamment pas de variable avec un grand nombre de valeur manquante, pas de variable sans aucune variance, pas de variable avec une grande cardinalité. Cependant il existe une grande corrélation supérieur à 90% entre la variable Area et la variable Density (corrélation de 93.77%) . Nous regarderons donc ici uniquement l'effet de cette colinéarité sur les méthodes de sélection de variables de ARFS. Nous appliquerons donc ces méthodes avec toutes les variables du dataset versus ces méthodes lorsqu'on ne considère pas une de ces deux variables colinéaires (ici on ne considéra que la variable Density).

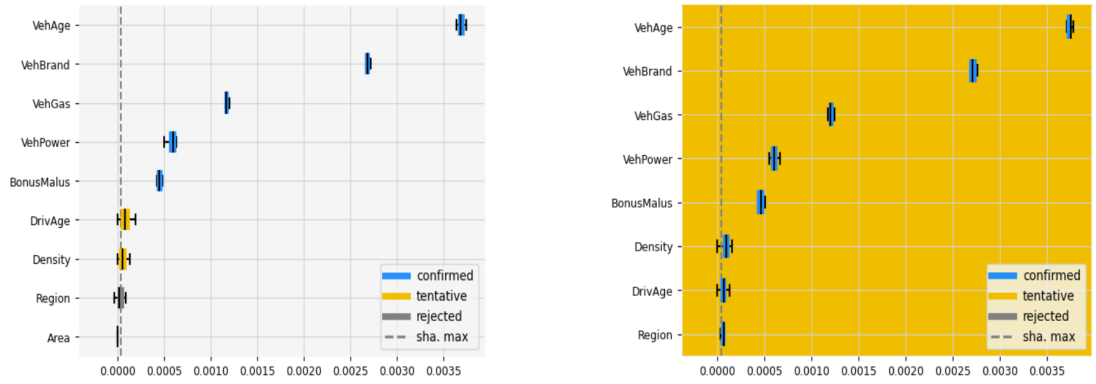
Leshy



(a) avec toutes les variables (b) sans les variables collinéaires

Figure 6.3.4 – Leshy avec XGBoost et Shap comme métrique d'importance

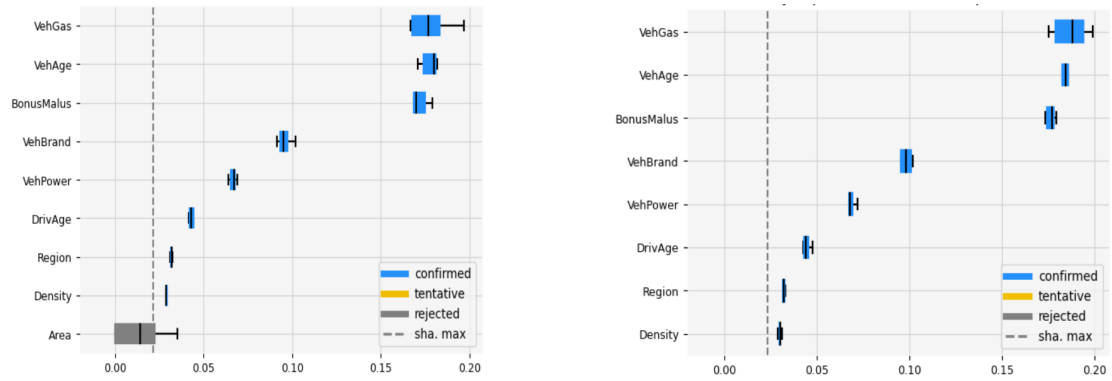
La figure ci dessus nous donne une application de la méthode Leshy utilisant XGBoost avec la métrique SHAP.
Comme Nous pouvons le voir, la variable Area ayant une importance SHAP presque nulle, elle n'influence pas trop la sélection des variables, ni le ranking, ni le niveau d'importance de la variable Density et de toute les autres variables. En utilisant Permutation Importance plutôt que Shap, nous obtenons :



(a) avec toutes les variables (b) sans les variables colinéaires

Figure 6.3.5 – Leshy avec XGBoost et Permutation Importance comme métrique

Ici on note une certaine influence de la variable Area, celle si semble cacher l'importance de la variable Density. En effet, lorsqu'on considère toute les variables, la variable Density n'est pas considérée comme pertinente pour la fréquence des sinistres, mais lorsqu'on supprime la variable Area, elle est choisit comme pertinente. De plus on note une modification du ranking de la variable Density lorsqu'on considère les variables colinéaires et 08 variables sont jugées pertinentes sans la variables Area tandis que 5 variables juste sont jugées pertinentes lorsqu'il y'a toutes les variables dans le dataset, Ce qui nous conforte comme vu avec le dataset artificiel, que Leshy avec permutation importance souffre bien du problème de la colinéarité car elle cache l'importance de certaine variables qui peuvent être pertinente.
En utilisant la métrique Gini impurity on a :



(a) avec toutes les variables (b) sans les variables colinéaires

Figure 6.3.6 – Leshy utilisant XGBoost comme régresseur et Gini impurity comme métrique d'importance.

On a la même conclusion qu'avec SHAP ci dessus on observe pas une grande modification du niveau d'importance et du ranking de nos variables.
Ainsi la variable Area ayant une importance presque nulle (bruit), cela semble ne pas trop affecté Leshy avec les métriques SHAP et Gini impurity. Cependant avec la métrique Permutation Importance, cela semble affecté notamment dans le ranking et le niveau d'importance. On peut conclure que Leshy est affecté par la colinéarité, mais cependant, si la

variable colinéaire à une importance nulle, cela ne semble pas trop influencer la sélection des caractéristiques avec les métriques Shap et Gini impurity, mais cela est bien affecté avec la métrique PIMP.

BoostAGroota

La figure ci dessous est une illustration de BoostAGroota avec XGBoost et Shap comme métrique :

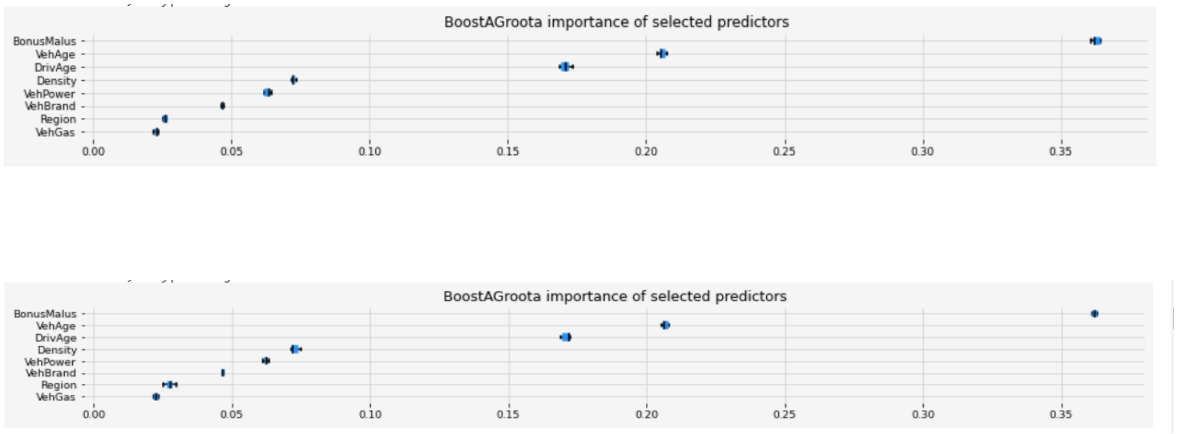


Figure 6.3.7 – BoostaGRoota utilisant XGBoost comme regrésseur et SHAP comme métrique d'importance

La figure au dessus est BoostAGroota avec toutes les variables et celle en dessous est BoostAGroota sans les variables colinéaires (ici Area). Comme pour Leshy ci-dessus, la variable Area étant un bruit et sans importance, Cela ne semble pas affecter la méthode avec la métrique Shap (pas de modification du ranking ou de l'ordre d'importance des variables. Avec la métrique Gini impurity, on a la même conclusion qu'avec Shap (voir l'annexe C.2).
en utilisant Permutation importance, on a la figure 6.3.8.

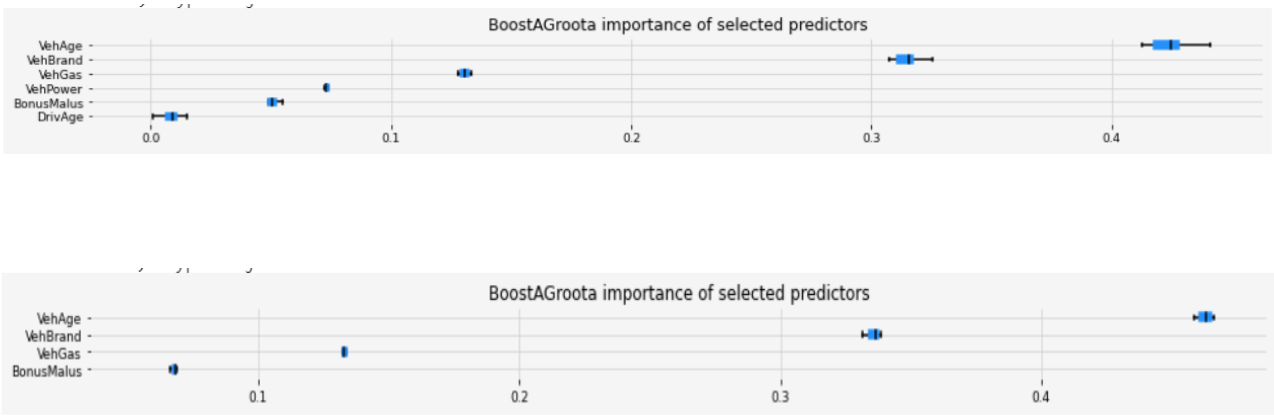


Figure 6.3.8 – BoostaGRoota avec XGBoost et Permutation importance comme métrique d'importance

La figure du dessus représente BoostAGroota avec toutes les variables, et celle en dessous représente BoostAgroota sans la variable Area. On observe clairement une différence notamment au niveau du nombre de variables sélectionnées, mais aussi au niveau de leur classement et du niveau d'importance car la mesure d'importance des variables est plus élevée sans le prédicteur colinéaire que lorsqu'on considère toute les prédicteurs.
Ainsi avec la métrique Permutation importance, BoostAGroota semble affecté la sélection des variables pour ce dataset notamment dans le ranking et le niveau d'importance, cette affectation n'est pas très visible avec les métriques Shap et Gini impurity due au fait que la variable Area est non pertinente.

GrootCV

avec tous les prédicteurs du dataset, nous avons :

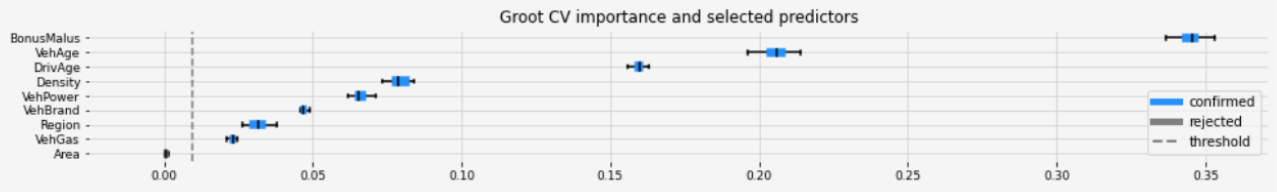


Figure 6.3.9 – GrootCV avec tous les prédicteurs

Sans le prédicteur colinéaire nous avons :

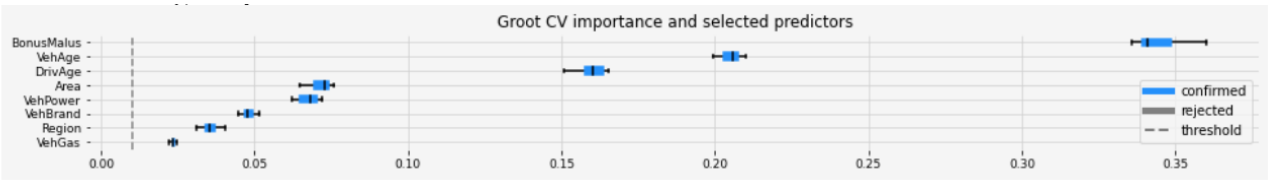


Figure 6.3.10 – GrootCV sans le prédicteur colinéaire

Nous remarquons qu'il y'a une grande différence avec et sans le prédicteur colinéaire. Ici, c'est la variable Density qui est supprimé. On peut voir que avec tous les prédicteurs, la variable Area a une importance SHAP nulle (bruit) et la variable Density est fortement pertinente (voir figure 6.3.9. Lorsqu'on supprime la variable Density du jeu de données, le rang et le niveau d'importance de la variable Area augmente et il devient une variable pertinente pour la fréquence des sinistres (voir figure 6.3.10). Cela nous conforte comme deja dit plus haut sur le fait que la colinéarité dégrade la qualité de sélection des variable.

Chapitre 7

Conclusion

Dans cette étude, il était question de comparer les méthodes de sélection des variables du module Python ARFS. Ces méthodes à savoir Leshy, BoostAGroota et GrootCV sont basées essentiellement sur les algorithmes du Boosting dont XGBoost, ligthGBM et pouvant utiliser trois mesures d'importance différente à savoir permutation importance, SHAP et Gini impurity, la méthode GrootCV n'utilisant que ligthGBM comme régresseur et SHAP comme mesure d'importance. Après avoir présenté ces algorithmes populaires du Boosting qui sont les principaux régresseurs pour les méthodes de ARFS et après avoir étudié les deux principaux problèmes de la sélection des variables, nous avons étudié ces méthodes de mesure de la dimension des performances d'une variable, ce qui nous a permis de présenter les méthodes de ARFS et d'appliquer ces méthodes sur deux jeux de données, dont un jeu de données artificiel sur lequel on a le contrôle sur tous les aspects à étudier et un jeu de données réel afin de voir l'impact de certains aspects lors de la sélection des variables avec ARFS.

Nous pouvons dire que nos trois méthodes ne sont pas sensibles au problème de présence dans le jeu de données des variables avec des valeurs manquantes (voir section 6.2.3) ou des variables sans variance (voir section 6.2.4). En effet, la présence de ces variables dans le jeu de données lors de la sélection des variables pertinentes n'affecte pas le niveau d'importance ni le ranking pour les variables du jeu de données. Ainsi pour la feature sélection avec les méthodes de ARFS, on peut se passer de ces effets lors de la première étape du préprocessing car cela n'aura à priori pas d'impact lors de la sélection des variables.
Nos trois méthodes souffrent principalement du problème de la multicolinéarité. La présence des prédicteurs multicolinéaires dans le jeu de données dégrade la qualité de sélection des variables. Que ce soit avec les métriques SHAP, permutation importance ou Gini impurity, nos méthodes partagent l'importance des variables multicolinéaires entre elles et ceci en fonction du niveau de corrélation.
La multicolinéarité réduit donc l'importance des variables pertinentes lorsqu'elles sont fortement corrélées avec une ou plusieurs autres variables dans le jeu de données car plus il y'a de prédicteurs colinéaires, plus l'importance est réduite et donc avant de faire donc la sélection des variables avec les méthodes de ARFS, il est important de détecter d'abord les corrélations entre les différentes variables prédictives, de les supprimer en ne gardant qu'une seule parmi elles avant de faire la sélection des variables.
La classe "FeatureSelector" de ARFS disposant de la méthode **identify_high_cardinality** (voir la section 6.1), il est également important de préciser qu'un travail complémentaire

consistera peut être à étudier l'effet des variables catégorielles ayant une grande cardinalité afin de voir si comme la multicolinéarité, ces variables peuvent avoir une grande influence lors de la sélection des variables pertinentes avec ARFS.

Bibliographie

[1] Seema Singh. « Understanding the bias-variance tradeoff ». In : *Towards Data Science* (2018) (pages 4, 5).

[2] Jerome H Friedman. *The elements of statistical learning : Data mining, inference, and prediction*. springer open, 2017 (pages 5, 6).

[3] Tianqi Chen et Carlos Guestrin. « Xgboost : A scalable tree boosting system ». In : *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, p. 785-794 (pages 8, 9).

[4] *Algorithmes de boosting-Adaboost,Gradient Boosting,XGBoost*. https://datascientest.com/algorithmes-de-boosting-adaboost-gradient-boosting-xgboost. Oct. 2020 (pages 9, 10).

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye et Tie-Yan Liu. « Lightgbm : A highly efficient gradient boosting decision tree ». In : *Advances in neural information processing systems* 30 (2017), p. 3146-3154 (pages 10-12).

[6] Roland Nilsson, José M Pena, Johan Björkegren et Jesper Tegnér. « Consistent feature selection for pattern recognition in polynomial time ». In : *The Journal of Machine Learning Research* 8 (2007), p. 589-612 (pages 13, 15-18, 61).

[7] Luc Devroye, László Györfi et Gábor Lugosi. *A probabilistic theory of pattern recognition*. T. 31. Springer Science & Business Media, 2013 (page 14).

[8] Saurav Kaushik. « Introduction to Feature Selection methods with an example (or how to select the right variables ?) » In : *Analytics Vidhya* (2016) (page 18).

[9] Verma Vikas. « A comprehensive guide to Feature Selection using Wrapper methods in Python ». In : *Analytics Vidhya* (2020) (page 19).

[10] C Molnar, S Gruber et P Kopper. *Limitations of interpretable machine learning methods*. 2020 (pages 21, 24-28).

[11] Terence Parr, Kerem Turgutlu, Christopher Csiszar et Jeremy Howard. « Beware default random forest importances ». In : *March* 26 (2018), p. 2018 (pages 21, 23).

[12] Leo Breiman. « Random forests ». In : *Machine learning* 45.1 (2001), p. 5-32 (page 22).

[13] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal et Su-In Lee. « Explainable AI for trees : From local explanations to global understanding ». In : *arXiv preprint arXiv :1905.04610* (2019) (page 26).

[14] Miron B Kursa et Witold R Rudnicki. « The all relevant feature selection using random forest ». In : *arXiv preprint arXiv :1106.5112* (2011) (pages 29, 30).

[15] *Is this the Best Feature Selection Algorithm "BorutaShap"?* https://medium.com/analytics-vidhya/is-this-the-best-feature-selection-algorithm-borutashap-8bc238aa1677. Juin 2020 (page 30).
[16] *BoostAroota*. https://github.com/chasedehan/BoostARoota. Jan. 2018 (page 30).
[17] *All relevant feature selection*. https://github.com/ThomasBury/arfs. Juil. 2021 (pages 31-34).
[18] https://github.com/scikit-learn-contrib/boruta_py/blob/eaad6a3b0991a664e73fead17e95boruta/test/unit_tests.py. Déc. 2018 (page 35).
[19] https://www.openml.org/d/41214. Nov. 2018 (page 48).
[20] https://www.openml.org/d/41215. Nov. 2018 (page 48).

Annexe A

Boosting

A.1 LigthGBM

A.1.1 Algorithme pour Goss

Algorithme pour GOSS

Entrée : I : donnée d'apprentissage
Entrée : a : taux d'echantillonage des données avec de grand gradient
Entrée : b : taux d'echantillonage des données avec de petit gradient
Entrée : l : fonction de perte
Entrée : L : apprenant faible
Entrée : N : itération
modele← {} , fact ← $\frac{1-a}{b}$
topN← $a * taille(I)$, randN←$b * taille(I)$
pour i allant de 1 à N **faire** :
pred ← models.predict(I)
g← l(I,pred)
$\omega \leftarrow [1, 1, ..]$
$trie \leftarrow GetsortedIndices(abs(g))$
$topset \leftarrow trie[1 : topN]$
$randset \leftarrow Randompick(trie[topN : taille(I)]$,randN)
usedset ← $topset + randset$
ω[randset] ← ω[randset]*fact
newmodel ← $L(I[usedset], -g[usedset], \omega$[usedset])
modele.append(newmodel)
finpour

A.1.2 Algorithme pour EFB

1. **Algorithmes pour grouper les variables :**

Algorithme formel pour grouper les variables

Entrée : F : variables
Entrée :K : seuil
Construire le graphe G
searchOrder $\leftarrow G.sortBydegree()$
$bundle \leftarrow \{\}$
$bundlesconflit \leftarrow \{\}$
pour$i$ dans searchOrder **faire** :
    neednew $\leftarrow vrai$
    pour$j$ allant de 1 à taille(bundles) **faire** :
        Cnt $\leftarrow conflict(bundles[j]$,F[$i$])
        si(Cnt+bundlesconflit[i]$\leqslant k$) **alors**
            bundle[j].ajouter($F[i]$)
            needNew $\leftarrow faux$
        finsi
    finpour
    si$neednew$ **alors**
        ajouter $F[i]$ comme un nouveau groupe de bundles
    finsi
finpour
retourner $bundles$

2. **Algorithme pour fusionner les variables** :

Algorithme pour la fusion des variables

Entrée : $numdata$:nombre de donnée
Entrée :F :un groupe de variable exclusive
binRanges $\leftarrow \{0\}$
$totalbin \leftarrow 0$
pour$f$ dans F **faire** :
    totalBin $\leftarrow totalBin + F$
    $binRanges.append(totalBin)$
finpour
$newBin \leftarrow nouveau$ Bin(numdata)
pour i allant de 1 à $numdata$ **faire** :
    $newbin[i] \leftarrow 0$
    pour$j$ allant de 1 à $taille[F]$ **faire** :
        si ($F[j].bin[i]\#0$) **alors** :
            newBin[i] $\leftarrow F[j]$.bin(i) + binRanges[j]
        finsi
    finpour
finpour
retourner : $newBin$, binRanges

Annexe B

Problèmes de sélection des variables

B.1 Classificateur, risque, et optimalité

B.1.1 Preuve que le classificateur de Bayes g^* est optimal

Soit $g(x) : \mathcal{X} \longrightarrow \{-1; 1\}$ un classificateur, montrons que $R(g) \leqslant R(g^*)$ c'est-à-dire $P(g^*(X) \neq Y) \neq P(g(X) \neq Y)$.
Pour $X = x$,posons $\eta(x) = P(Y = 1|X = x)$ on a :

$$\begin{aligned} p(g(X) \neq Y|X = x) &= 1 - p(g(X) = Y|X = x) \\ &= 1 - [p(Y = 1, g(X) = 1|X = x) + p(Y = -1, g(X) = -11|X = x)] \\ &= 1 - [1_{\{g(X)=1\}} * p(Y = 1|X = x) + 1_{\{g(X)=-1\}} * p(Y = -1|X = x)] \\ &= 1 - [1_{\{g(X)=1\}} * \eta(x) + 1_{\{g(X)=-1\}} * (1 - \eta(-x))] \end{aligned} \tag{B.1.1}$$

où 1_A désigne la fonction indicatrice de l'ensemble A.
Ainsi, en posant $A = p(g(X) \neq Y|X = x) - p(g^*(X) \neq Y|X = x)$, on a

$$\begin{aligned} A &= 1_{\{g^*(X)=1\}} * \eta(x) + 1_{\{g^*(X)=-1\}}(1 - \eta(x)) - [1_{\{g(X)=1\}} * \eta(x) + 1_{\{g(X)=-1\}} * (1 - \eta(x))] \\ &= \eta(x)[1_{\{g^*(X)=1\}} - 1_{\{g(X)=1\}}] + (1 - \eta(x)[1_{\{g^*(X)=-1\}} - 1_{\{g(X)=-1\}}] \\ &= \eta(x)[1_{\{g^*(X)=1\}} - 1_{\{g(X)=1\}}] + (1 - \eta(x))[1 - 1_{\{g^*(X)=1\}} - 1 + 1_{\{g(X)=1\}}] \\ &= (2\eta(x) - 1)[1_{\{g^*(X)=1\}} - 1_{\{g(X)=1\}}] \\ &\geqslant 0 \end{aligned} \tag{B.1.2}$$

En intégrant A par rapport à $f(x)d_x$, on obtient le résultat.
Ainsi, g^* est optimal et g^* est unique pour les distributions strictement positives, sauf pour les sous ensembles de mesure nulle de X (voir [6]).

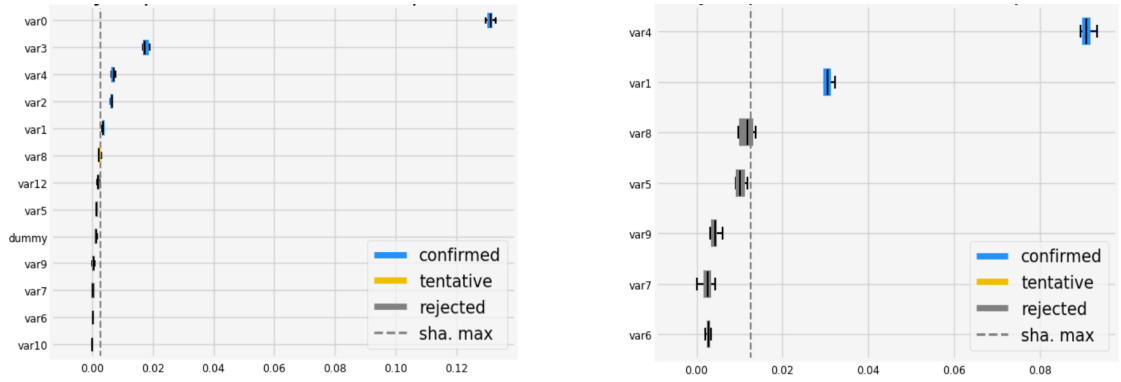
Annexe C

Application des méthodes de ARFS

C.1 Sur le dataset artificiel

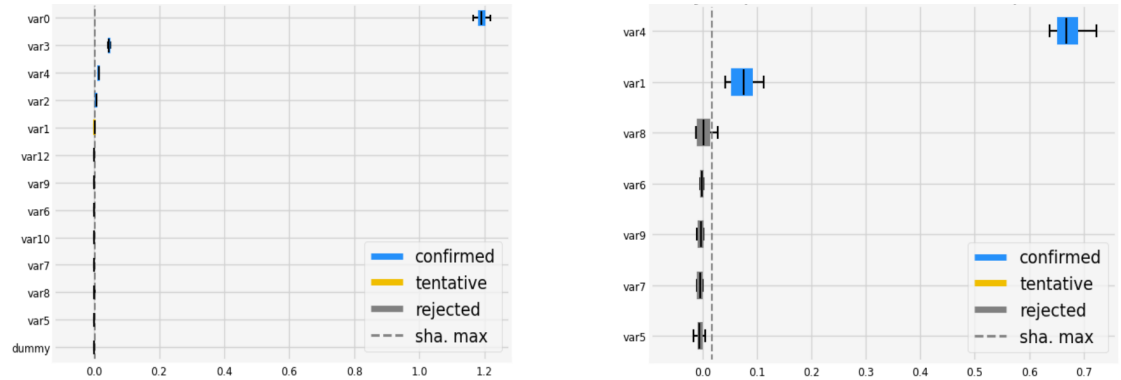
C.1.1 Effet de la multicolinéarité

Leshy avec ligthGBM comme regrésseur



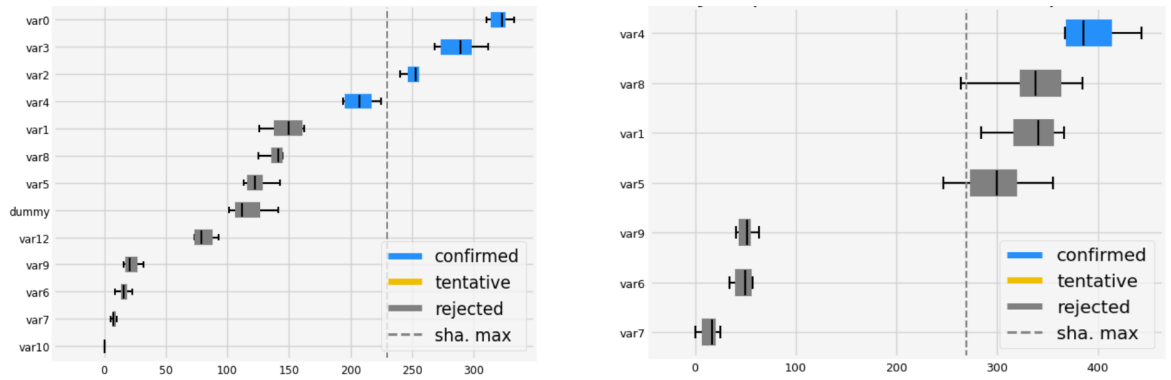
(a) avec toutes les variables (b) sans les variables colinéaires

Figure C.1.1 – Leshy avec ligthGBM et SHAP comme métrique d'importance



(a) avec toutes les variables (b) sans les variables colinéaires

Figure C.1.2 – Leshy avec ligthGBM et PIMP comme métrique d'importance



(a) avec toutes les variables (b) sans les variables colinéaires

Figure C.1.3 – Leshy utilisant ligthGBM comme régresseur et native comme métrique d'importance

BoostAGroota avec ligthGBM comme régresseur

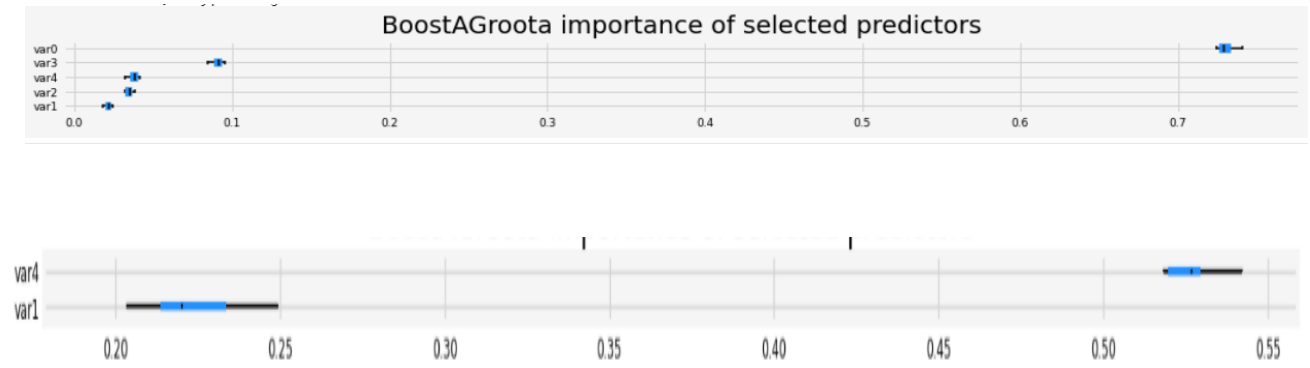


Figure C.1.4 – BoostaGRoota avec ligthGBM et SHAP comme métrique d'importance

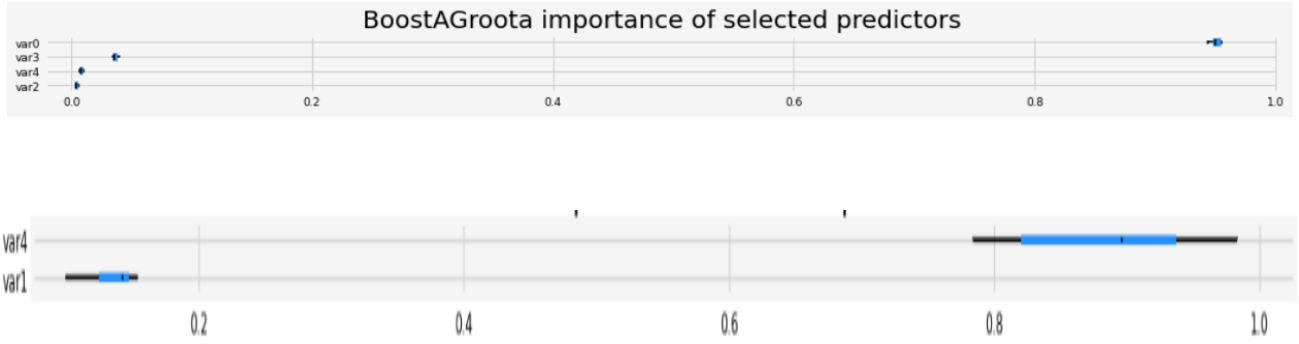


Figure C.1.5 – BoostaGRoota avec ligthGBM et PIMP comme métrique d'importance

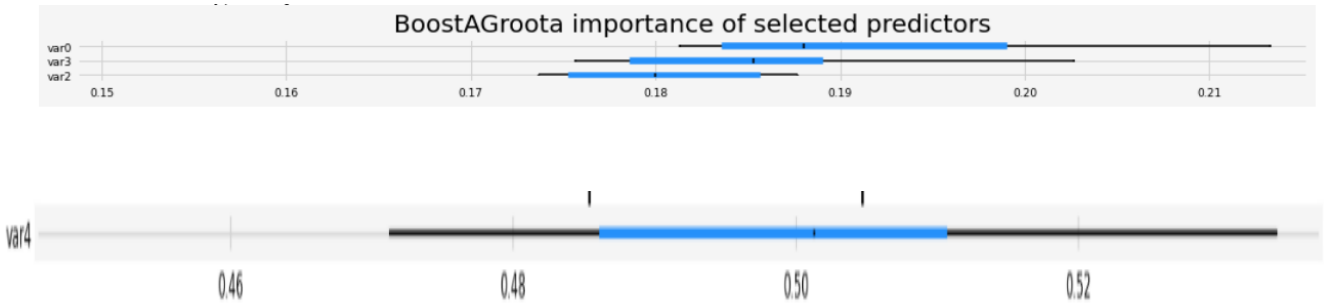
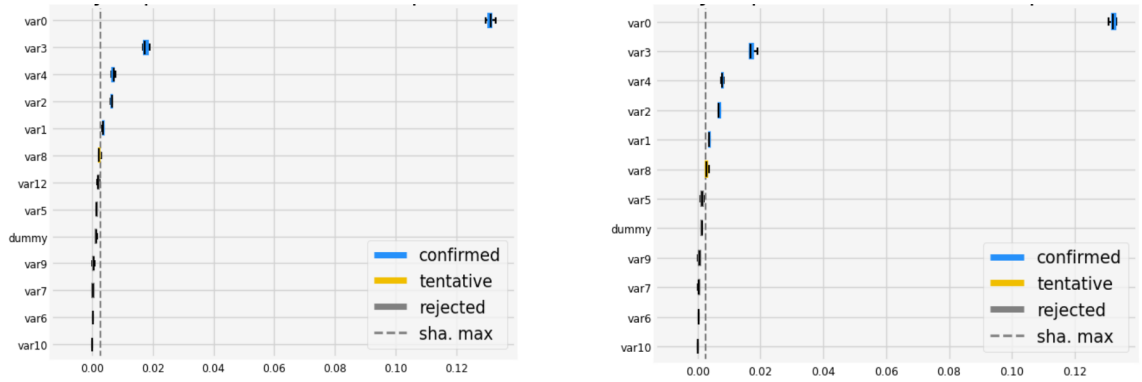


Figure C.1.6 – BoostaGRoota utilisant ligthGBM comme régresseur et native comme métrique d'importance

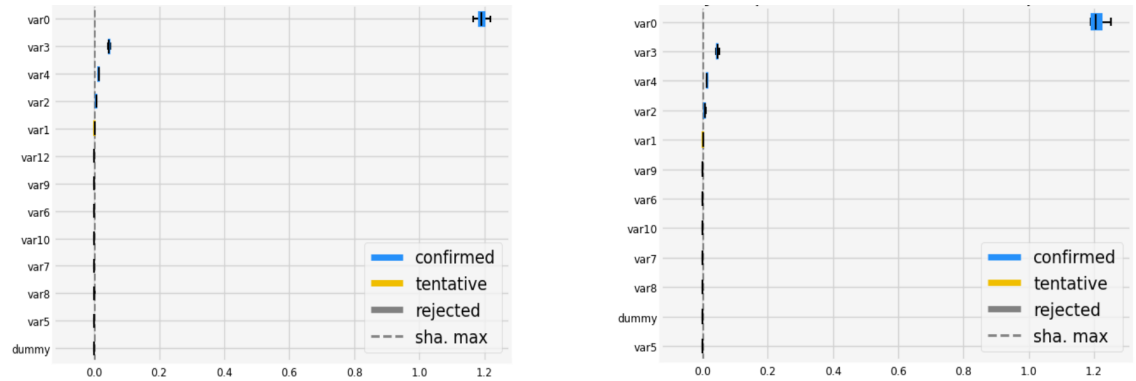
C.1.2 Effet des variables avec plusieurs valeurs manquantes

Leshy avec ligthGBM comme régresseur



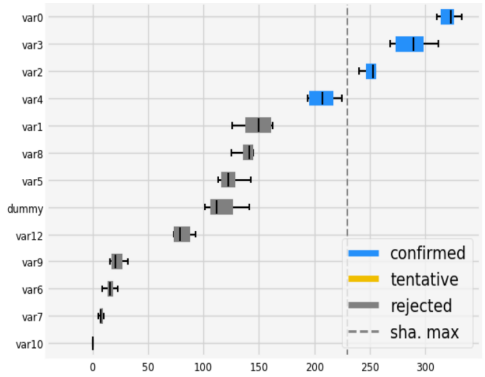
(a) avec toutes les variables (b) sans les variables avec valeurs manquantes

Figure C.1.7 – Leshy avec ligthGBM et Shap comme métrique d'importance

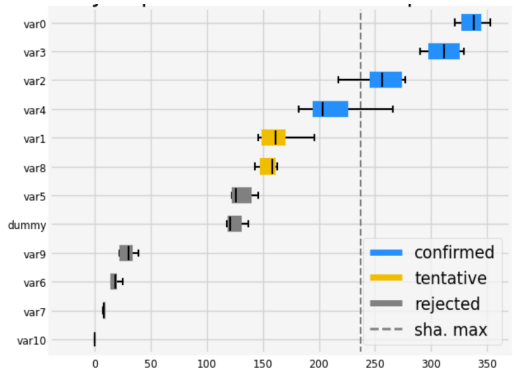


(a) avec toutes les variables (b) sans les variables avec valeurs manquantes

Figure C.1.8 – Leshy avec ligthGBM et PIMP comme métrique d'importance



(a) avec toutes les variables



(b) sans les variables avec valeurs manquantes

Figure C.1.9 – Leshy avec ligthGBM et Gini impurity comme métrique d'importance

C.2 Sur le dataset réel French TPLMD

BoostaGRoota avec XGBoost et Gini impurity comme métrique

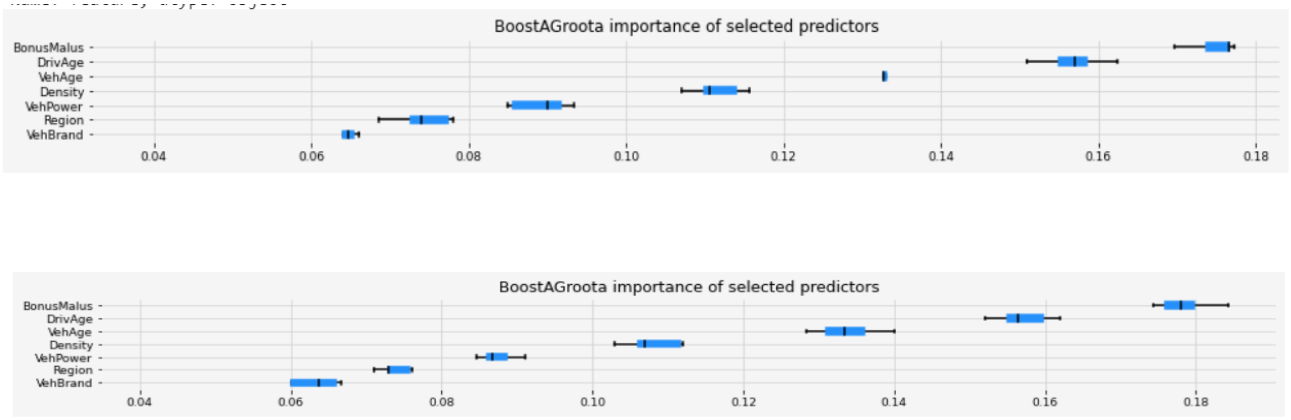


Figure C.2.1 – BoostaGRoota avec XGBoost et Gini impurity comme métrique d'importance