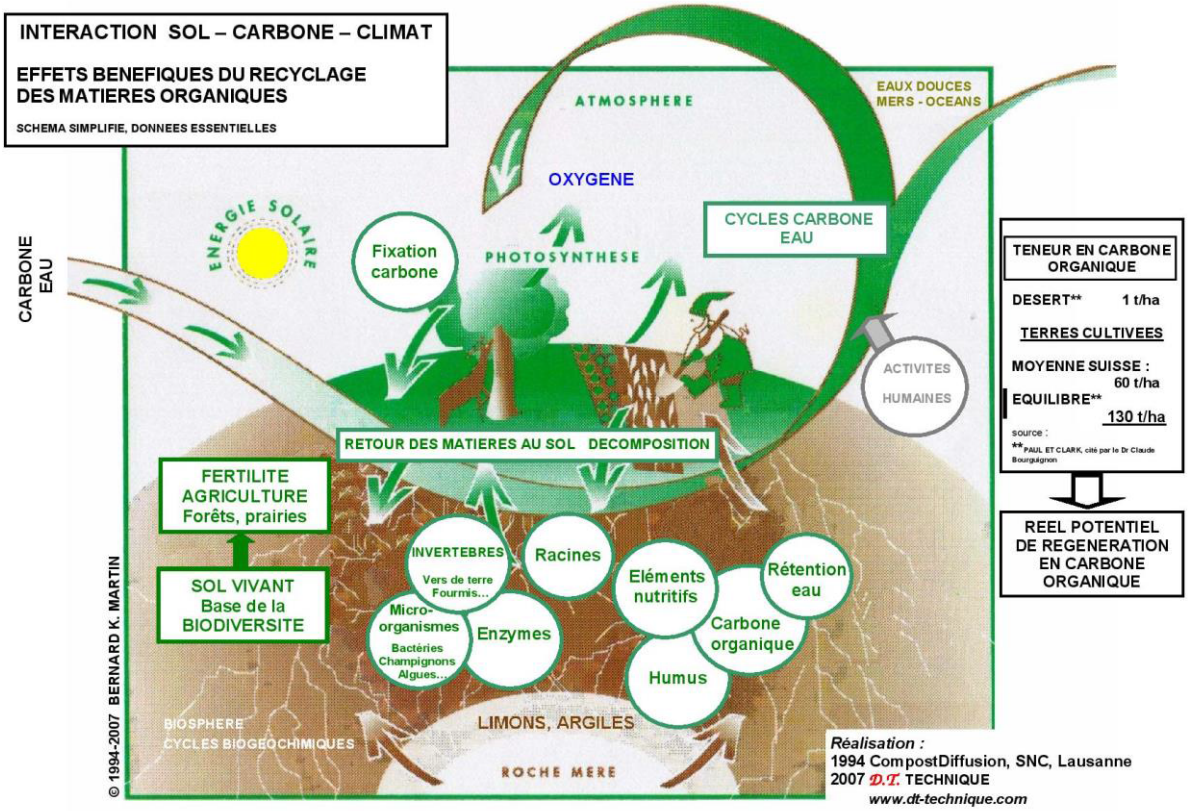
Estimation des incertitudes liées à la prédiction ponctuelle d'une variable pédologique à partir de la base de données géographiques sur les sols

Violaine Murciano

RAPPORT DE STAGE 2013
MASTER 2 STATISTIQUES ET PROBABILITES APPLIQUEES



Estimation des incertitudes liées à la prédiction ponctuelle d'une variable pédologique à partir de bases de données géographiques sur les sols

MURCIANO Violaine

Liste des Abréviations :

INRA : Institut National de la Recherche Agronomique
EPST : Etablissement Public à caractère Scientifique et Technologique
EPCS : Etablissement Public de Coopération Scientifique
CIRAD : Centre de coopération Internationale en Recherche Agronomique pour le Développement
CNRS : Centre National de la Recherche Scientifique
UMR : Unité Mixte de Recherche
US : Unité de Service
IRD : Institut de Recherche pour le Développement
ONF : Office National des Forêts
AGPF : Amélioration, Génétique et Physiologie Forestières
ZF : Zoologie Forestière
GBFor : Génétique et Biomasse Forestière
GIS Sol : Groupement d'Intérêt Scientifique Sol
IGN : Institut Géographique National
ADEME : Agence De l'Environnement et de la Maîtrise de l'Energie
IUSS : International Union of Soil Sciences (Union Internationale des Sciences du Sol)
IGCS : Inventaire, Gestion et Conservation des Sols
GBM: Generalized Boosted Regression Models
MART: Multiple Additive Regression Trees
RMSE: Root Mean Square Error
MSE: Mean Square Error
MPE: Mean Prediction Error
RPD: Ratio of Performance to Deviation

Remerciements

Je tiens à remercier l'ensemble de l'équipe de l'Unité de Service InfoSol de l'INRA d'Orléans pour son accueil et son aide qui a contribué au bon déroulement de mon stage.

Je remercie plus particulièrement Marion BARDY, directrice de l'US InfoSol pour m'avoir accueilli au sein de sa structure.

Je remercie vivement Dominique ARROUAYS, mon tuteur de stage, d'avoir pris le temps de répondre à mes questions, mes doutes. Pour m'avoir guidé progressivement tout au long de ma mission.

Je remercie également Jean-Baptiste PAROISSIEN et Manuel MARTIN pour leur disponibilité et leur aide tout au long du projet.

Je remercie Anne RICHER DE FORGES et Jean Philippe CHENU pour l'extraction des données.

Enfin je remercie Richard EMILION, professeur encadrant du stage, Didier CHAUVEAU ainsi que toute l'équipe pédagogique du master « Statistiques et Probabilités Appliquées » de la Faculté des Sciences d'Orléans pour leurs enseignements m'ayant permis d'effectuer ce stage.

Introduction

Etudiante en deuxième année de Master Statistique et Probabilités Appliquées à l'Université d'Orléans, j'ai eu l'opportunité d'effectuer mon stage de fin d'étude à l'Institut National de Recherche Agronomique d'Orléans, au sein de l'Unité de Service InfoSol.

Le sol est la couche superficielle de la croûte terrestre qui résulte de la transformation de la roche mère et des apports organiques. Le sol se différencie de la croûte terrestre par la présence de vie. Il est également un réservoir de carbone planétaire qui diminue cependant depuis quelques années.

Aujourd'hui, il y a de nombreuses questions qui intéressent concrètement la gestion des territoires ruraux et des ressources. Pour pouvoir y répondre, il faut disposer d'informations précises sur les propriétés des sols et leur localisation.

L'objectif de mon stage est d'effectuer l'estimation des incertitudes liées à la prédiction de variables pédologiques sur les sols français.

Dans ce rapport je vais tout d'abord présenter le contexte de la demande ainsi que les objectifs. Puis je présenterai le matériel et les méthodes utilisées durant ce stage. Ensuite, j'exposerai les résultats obtenus. Je terminerai enfin par une discussion et interprétation des résultats.

1. Contexte et objectifs

1.1. Contexte du stage

1.1.1. Présentation générale de l'INRA

Etablissement public à caractère scientifique et technologique (EPST), l'INRA est placé sous la double tutelle du ministère de l'Agriculture, de l'Agroalimentaire et de la Forêt et du ministère de l'Enseignement supérieur et de la Recherche. Premier institut de recherche agronomique en Europe, deuxième dans le monde, l'INRA mène des recherches finalisées pour une alimentation saine et de qualité, pour une agriculture compétitive et durable, et pour un environnement préservé et valorisé.

C'est en 1946, juste après la guerre, que l'INRA a été créé dans un contexte de reconstruction du pays et de modernisation de l'agriculture française. Aujourd'hui les objectifs ont évolué et ont désormais une dimension mondiale. En effet, les recherches de l'INRA sont motivées par l'évolution permanente des questionnements scientifiques ainsi que les défis planétaires posés à l'agronomie et l'agriculture tels que le changement climatique, l'épuisement des ressources fossiles et la nutrition humaine.

L'INRA compte 8488 agents titulaires dont plus de 1800 chercheurs, 2500 ingénieurs, 4000 techniciens et administratifs, mais également près de 2000 doctorants. Sont accueillis chaque année environ 2000 stagiaires et 1800 chercheurs et étudiants étrangers.

L'INRA fait partie d'Agreenium, qui est un Etablissement Public de Coopération Scientifique (EPCS) et qui comprend aussi le Centre de coopération internationale en recherche agronomique pour le développement (CIRAD) et les écoles nationales d'agronomie. Il entretient de nombreuses collaborations et échanges avec la communauté scientifique internationale avec des pays d'Europe, Asie, Amérique et Afrique.

Cet établissement public qu'est l'INRA est composé de 13 départements scientifiques et 18 centres régionaux. On y trouve 213 Unités de Recherche dont 112 Unités Mixtes de Recherche (UMR) qui associent donc l'INRA à d'autres organismes de recherche (comme par exemple le Centre National de la Recherche Scientifique (CNRS), le CIRAD, ou l'institut de Recherche pour le Développement (IRD)) et d'enseignement (notamment des Universités et Grandes Ecoles). L'INRA comprend aussi 49 Unités Expérimentales qui représentent une surface totale d'environ 10 000 ha ainsi que 94 000 animaux en élevage. En 2013, l'INRA possède un budget prévisionnel de 881,61 millions d'Euros. Cet institut détient donc un dispositif unique de recherche dans les sciences du vivant.

1.1.2. Centre INRA Val de Loire, site d'Orléans

Le centre INRA Val de Loire compte 960 agents dont 660 titulaires ce qui en fait le sixième centre de l'institut par son effectif. Il mène des recherches autours de quatre pôles répartis sur trois sites, Orléans, Tours et Bourges:

- Le pôle **Dynamique des sols et gestion de l'environnement** comprend deux unités du département Environnement et Agronomie. Les enjeux de ce pôle concernent la production agricole et la protection de l'environnement.
- Le pôle **Biologie intégrative des arbres et organismes associés** comprend 3 unités du département Ecologie forestière, Prairies et milieux Aquatiques et deux unités sous contrat avec l'Université d'Orléans et l'Office National des Forêts. Les recherches de ce pôle concernent l'étude des populations d'insectes vivant au sein de ces écosystèmes, le processus responsable de la formation du bois ainsi que la tolérance des arbres aux maladies et au stress hydrique.
- Le pôle **Biologie intégrative animale et gestion durable des productions animales** comprend 5 unités du département Physiologie Animale et Système d'Elevage et de Génétique animale. L'objectif de ce pôle consiste à améliorer la durabilité des systèmes d'élevage dans une dimension économique, sociale et environnementale.
- Le pôle **Santé animale et santé publique** comprend une unité et une plate-forme d'infectiologie expérimentale des départements de Santé Animale et de Microbiologie de la Chaîne Alimentaire.

Les deux premiers pôles sont sur le site d'Orléans.

C'est en 1972 que le site d'Ardon fut choisi pour y implanter un centre de l'INRA. Si au tout début ce centre ne comptait qu'un bungalow, aujourd'hui il est composé de 3 domaines de recherche et de 5 unités différentes. Ces domaines sont les suivants :

- **La sélection des arbres forestiers** dont les différents critères sont une croissance optimale, une certaine qualité du bois et une bonne résistance aux parasites. Dans ce domaine, diverses connaissances y sont développées telles que la génétique, la physiologie des arbres, les biotechnologies et perfectionnement des techniques de diffusion des variétés améliorées.
- **La biologie des insectes forestiers ravageurs** ainsi que leur épidémiologie et leur relation avec les arbres-hôtes.
- **La maîtrise des érosions et pollutions, l'évaluation des risques agro climatiques** en se basant sur la modélisation des comportements physiques et hydriques des sols, en relation avec leur histoire et leur cartographie.

Ces différentes thématiques donnent au centre d'Orléans une orientation en matière d'**environnement** et de **développement durable**. Il a par ailleurs déployé une politique

d'accueil et de partenariat avec différents organismes (Office National des Forêts (ONF), ArboCentre) qui favorise le lien entre la recherche, le développement et l'aide à la décision publique.

Le centre est composé de 5 unités :

- 3 unités de recherche : Amélioration, Génétique et Physiologie Forestières (AGPF), Zoologie Forestière (ZF) et Science du Sol ;
- 1 unité de service : InfoSol ;
- 1 unité expérimentale : Génétique et Biomasse Forestière (GBFor).

Le centre d'Orléans compte environ 200 agents titulaires dont 38% sont des Chercheurs et Ingénieurs, 49% des Techniciens et 13% des Administratifs. Une cinquantaine de non titulaires sont accueillis chaque année, beaucoup sont des étudiants Doctorants ou Masters mais aussi des post-doctorants et chercheurs étrangers. Le budget du centre en 2005, hors salaires, s'élevait à 5,9 millions d'Euros dont les origines sont diverses (Ministère, collectivités territoriales, Europe, autres organismes de recherche, partenaires,…).

1.1.3. L'Unité de Service InfoSol

Dirigée par Marion Bardy, l'Unité de Service InfoSol fait partie du Département Environnement et Agronomie. La mission principale de cette unité est de constituer et de gérer un système d'information à vocation nationale sur les sols, par rapport à leur distribution spatiale, leurs propriétés et l'évolution de leurs qualités.

L'US InfoSol travaille dans le cadre du groupement d'intérêt scientifique Sol (GIS Sol), en collaboration avec le ministère de l'agriculture et de la pêche, le ministère de l'environnement et du développement durable, l'Institut Géographique National (IGN), l'Institut de Recherche pour le Développement (IRD), et l'Agence de l'Environnement et de la Maîtrise de l'Energie (ADEME).
Le GIS SOL propose un ensemble de programmes nationaux pour faciliter et encourager une gestion patrimoniale et durable des sols.

Les principales activités de l'unité sont, d'une part, la réalisation et la coordination de l'acquisition des données nécessaires à la constitution d'un système d'information sur les sols de France ainsi que le contrôle de la qualité de ces données et d'autre part la création et l'alimentation de bases de données permettant l'archivage et l'exploitation des informations. Enfin la dernière activité principale est la diffusion et la valorisation des connaissances sur les sols et les outils d'exploitation thématique produits par les recherches de l'Institut.

1.2. Contexte de la demande

Les grands enjeux planétaires actuels concernent la sécurité alimentaire, le changement climatique, l'accaparement des terres, l'urbanisation et ainsi que la gestion de l'eau. Afin de répondre à ces enjeux, la connaissance et la protection des sols s'avèrent être indispensables. Il est donc primordial de mettre en place des outils permettant de prendre en compte les propriétés des sols à l'échelle mondiale.

C'est ainsi qu'en 2006 a été lancé le projet GlobalSoilMap. Il a été lancé à l'initiative du Groupe de Travail « Digital Soil Mapping » de l'Union Internationale de Science du Sol (IUSS).

L'objectif est donc d'obtenir une base de données librement accessible de quelques propriétés des sols d'intérêt majeur sur la totalité du monde sous la forme de valeurs moyennes ou modales assorties d'intervalles de confiance ou fourchette de valeurs les plus probables de façon à rendre compte en même temps de l'incertitude associée.

Au niveau français, ces valeurs sont renseignées par le programme Inventaire, Gestion et Conservation des Sols (IGCS). Ce programme a pour objectifs :

- D'identifier, définir et localiser les principaux types de sols d'une région ou d'un territoire, caractériser leurs propriétés présentant un intérêt pour l'agriculture et l'environnement.
- D'évaluer les aptitudes à différents usages et en préciser les risques pour aider aux décisions.
- De constituer les bases de données de qualité répondant aux besoins des utilisateurs.

Il s'agit donc de cartographier, c'est-à-dire de délimiter des ensembles homogènes, des sols à l'échelle de la France.

1.3. Démarche et outils

Afin de renseigner ces bases de données, la première étape des experts se situe sur le terrain. En effet, les experts vont en différents endroits de la France creuser le sol afin de le décrire et d'en prélever des échantillons ; c'est ce qu'on appelle des profils de sols.
C'est sur ces profils que sont faites des mesures et des analyses pour obtenir différentes valeurs comme par exemple le carbone, le pH, ou encore la profondeur, la classe du matériel parental…
On définit également des couches, appelées strates.

Pour chaque strate, le pédologue va soit effectuer plusieurs analyses statistiques qui vont lui permettre d'obtenir une valeur moyenne ou modale des différentes variables ainsi

qu'une valeur minimale et maximale. Sinon, il peut estimer ces mêmes valeurs mais par expertise et non par analyses.
Dans la plupart des cas, cependant, on a juste une valeur moyenne souvent obtenue par expertise. C'est pour ces cas-là que j'interviens durant mon stage.

Différents champs sont disponibles dans les bases de données, chacun correspondant à une caractéristique des sols. Ces caractéristiques sont par exemple la teneur en carbone, le pH, l'argile, le limon et le sable.
Durant mon stage, j'ai utilisé seulement deux de ces valeurs : carbone et pH. Pourquoi avoir choisi ces 2 là ? D'une part, car ces 2 variables ont des distributions différentes.

D'autre part, elles sont d'une importance majeure.

Concernant le carbone, il est présent dans le sol sous forme de matière organique. Ce carbone agit sur la qualité du sol, plus il y a de carbone dans le sol mieux celui-ci se comporte, moins il y a de problèmes. Il agit également sur le changement climatique, l'effet de serre; en effet, plus il y a de carbone dans le sol, moins il y en a dans l'air. Cela contribue à la lutte contre l'effet de serre. L'objectif est donc d'augmenter les stocks de carbone pour essayer d'atténuer les émissions et ainsi diminuer l'effet de serre.
Les sols mondiaux contiennent environ 1500 milliards de tonnes de carbone organique et 9 à 10 milliards de tonnes sont rejetées dans l'air annuellement par les activités humaines.

En ce qui concerne le pH, il joue un rôle pour l'agronomie, les forêts ainsi que la production agricole. Celui-ci est important pour la mobilité et la disponibilité des éléments. En milieu non calcaire et sous climat tempéré les sols ont tendance à s'acidifier naturellement. Si le pH est trop élevé ou trop faible, il peut y avoir des problèmes de toxicité ou de carence pour les plantes.

1.4. Objectifs

L'objectif de ce stage est donc d'estimer les valeurs minimales et maximales des différentes variables à partir des valeurs modales prédites ainsi que d'autres covariables disponibles dans l'espace.
La connaissance de l'incertitude est importante notamment pour les personnes qui font de la modélisation, des études de sensibilité afin de pouvoir se rendre compte dans quelle gamme on se situe et d'avoir une estimation des incertitudes associées à une prédiction.

2. Matériel et Méthodes

2.1. Données

L'étude porte sur l'ensemble des analyses des données IGCS. Ces données sont regroupées au sein de bases de données, composées de 55 colonnes et 13261 lignes pour le carbone et 37261 pour le pH. Ces colonnes correspondent à différentes variables qui sont décrites en Annexe 1.

16 covariables ont été sélectionnées pour analyser les valeurs minimales, maximales, étendues et étendues normalisées du carbone et du pH. Ces covariables ont été choisies par expertise. Dans cette base remaniée, toutes les lignes ayant des valeurs manquantes ont été enlevées afin de pouvoir comparer toutes les méthodes car toutes ne les acceptent pas. Il reste donc 9311 lignes pour le carbone et 22059 pour le pH.

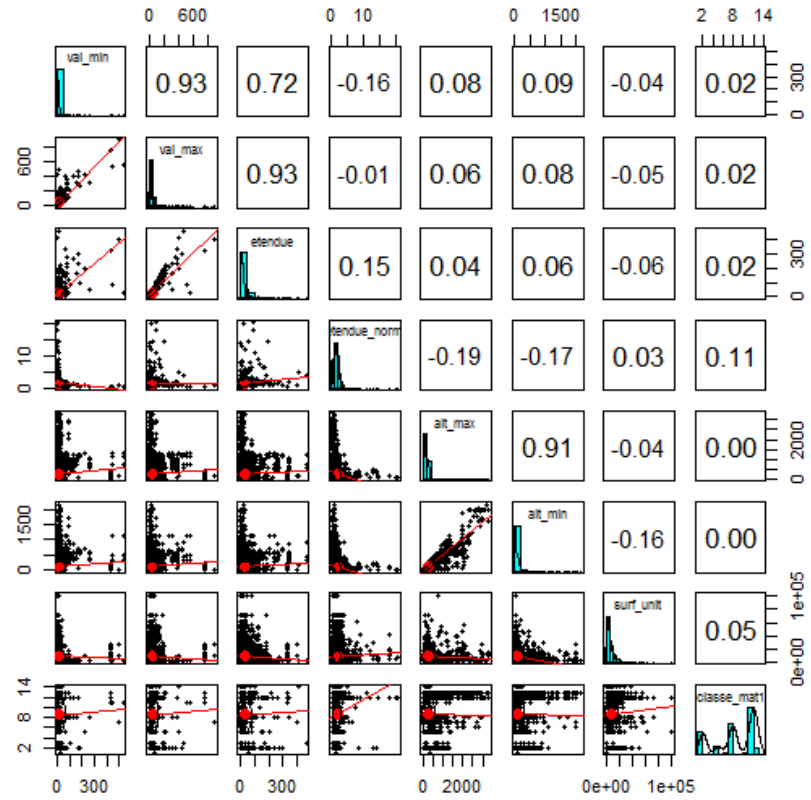
Les 20 variables sont les suivantes (16 covariables et 4 variables à expliquer) :

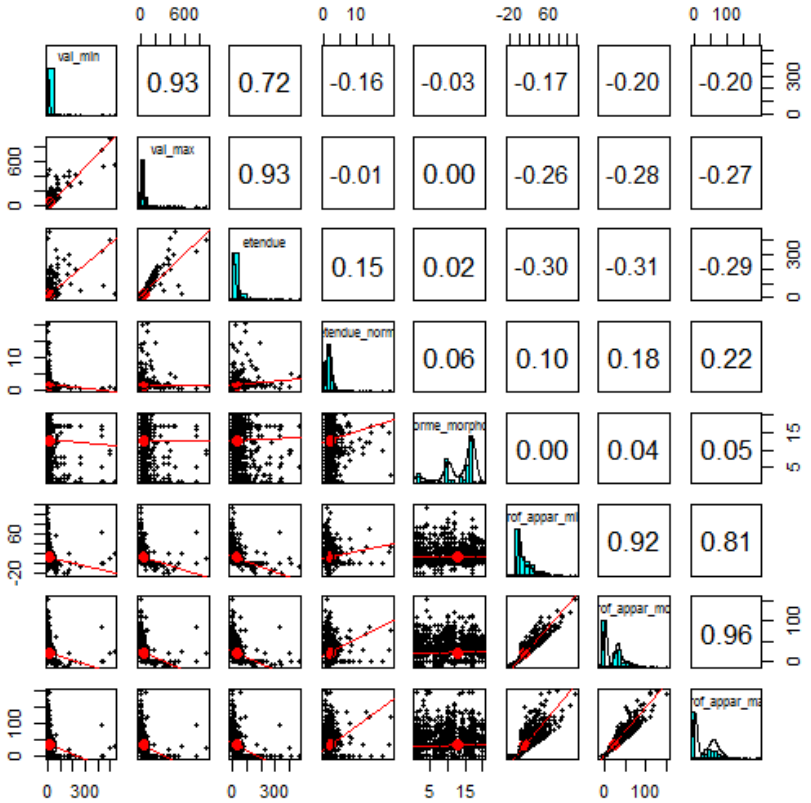
Tableau 1: Liste des variables utilisées dans l'étude

Variables	Type	Description rapide
alt_max	quantitative	Altitude maximale
alt_min	quantitative	Altitude minimale
surf_unit	quantitative	Superficie de l'UCS
classe_mat1	qualitative (14 niveaux pour carbone et 13 pour pH)	Catégorie de matériel parental
forme_morpho	qualitative (20 niveaux pour carbone et 25 pour pH)	Forme morphologique
prof_appar_min	quantitative	Profondeur minimale d'apparition
prof_appar_moy	quantitative	Profondeur moyenne d'apparition
prof_appar_max	quantitative	Profondeur maximale d'apparition
epais_min	quantitative	Epaisseur minimale
epais_moy	quantitative	Epaisseur moyenne
epais_max	quantitative	Epaisseur maximale
val_mod	quantitative	Valeur modale
alt_moy	quantitative	Altitude moyenne : (alt_max + alt_min)/2
alt_delta	quantitative	alt_max – alt_min
profondeur	quantitative	epais_moy + (prof_appar_moy / 2)
surf_uts	quantitative	Superficie de l'UTS (pourcent * surf_unit)/100
val_min	quantitative	Valeur minimale
val_max	quantitative	Valeur maximale
etendue	quantitative	Etendue : val_max – val_min
etendue_norm	quantitative	Etendue normalisée : (val_max – val_min) / val_mod

Les Figure 1 et Figure 2 présentent les diagrammes de dispersion (scatterplot) pour les différentes covariables du carbone et du pH. Ces scatterplots permettent de repérer d'éventuelles relations entre les covariables et principalement entre les quatre variables à expliquer et les autres.

Sur les diagonales, le nom de chaque covariable ainsi que son histogramme sont présentés. Sous la diagonale se trouve les nuages de points ainsi que la droite de régression pour chaque couple de covariables.


val_min
val_max
etendue
tendue_norm
alt_max
alt_min
surf_unit
classe_mat1
0.93
0.72
-0.16
0.08
0.09
-0.04
0.02
0.93
-0.01
0.06
0.08
-0.05
0.02
0.15
0.04
0.06
-0.06
0.02
-0.19
-0.17
0.03
0.11
0.91
-0.04
0.00
-0.16
0.00
0.05


val_min
val_max
etendue
tendue_norm
orme_morpho
rof_appar_mi
rof_appar_mo
rof_appar_ma
0.93
0.72
-0.16
-0.03
-0.17
-0.20
-0.20
0.93
-0.01
0.00
-0.26
-0.28
-0.27
0.15
0.02
-0.30
-0.31
-0.29
0.06
0.10
0.18
0.22
0.00
0.04
0.05
0.92
0.81
0.96

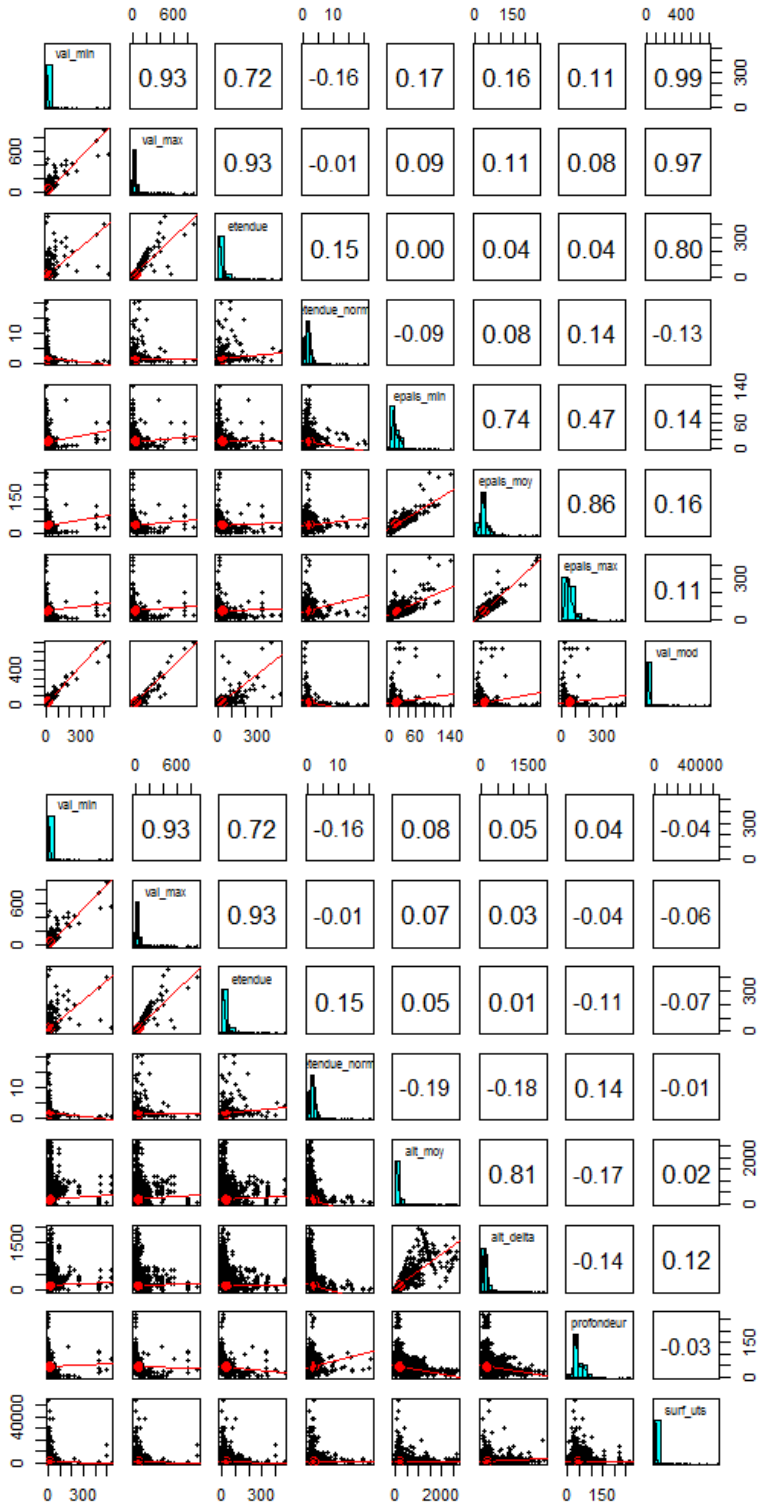
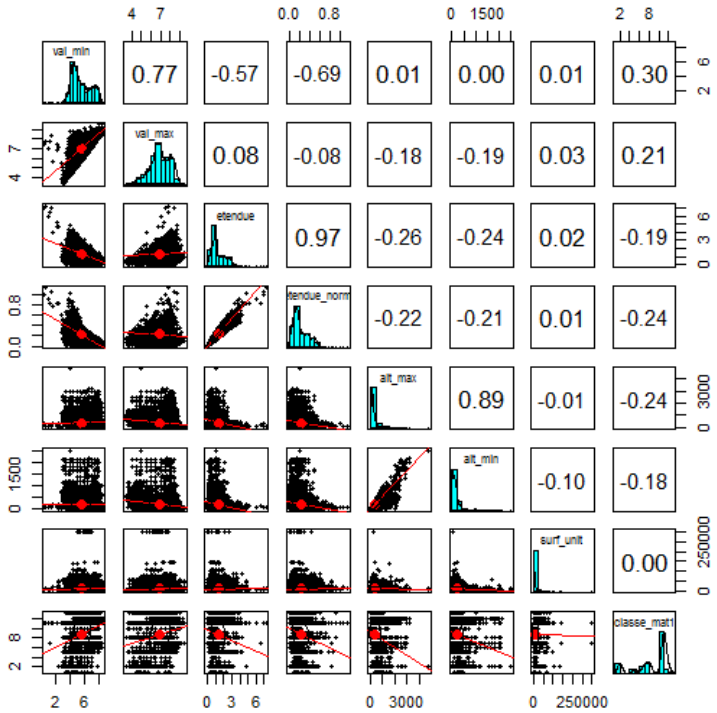
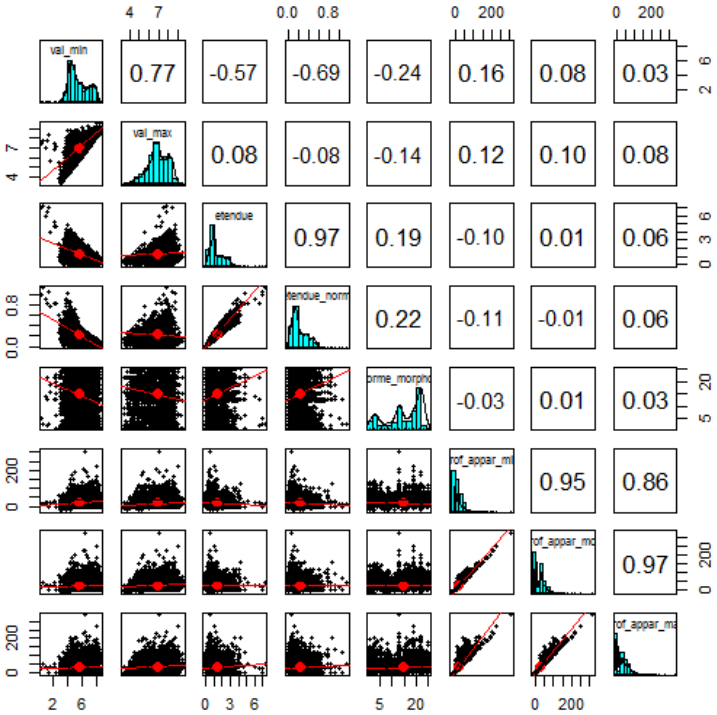


Figure 1: Scatterplot pour le carbone


val_min
val_max
etendue
etendue_norm
alt_max
alt_min
surf_unit
classe_mat1
0.77 -0.57 -0.69 0.01 0.00 0.01 0.30
0.08 -0.08 -0.18 -0.19 0.03 0.21
0.97 -0.26 -0.24 0.02 -0.19
-0.22 -0.21 0.01 -0.24
0.89 -0.01 -0.24
-0.10 -0.18
0.00


val_min
val_max
etendue
etendue_norm
orme_morphol
rof_appar_mi
rof_appar_mo
rof_appar_ma
0.77 -0.57 -0.69 -0.24 0.16 0.08 0.03
0.08 -0.08 -0.14 0.12 0.10 0.08
0.97 0.19 -0.10 0.01 0.06
0.22 -0.11 -0.01 0.06
-0.03 0.01 0.03
0.95 0.86
0.97

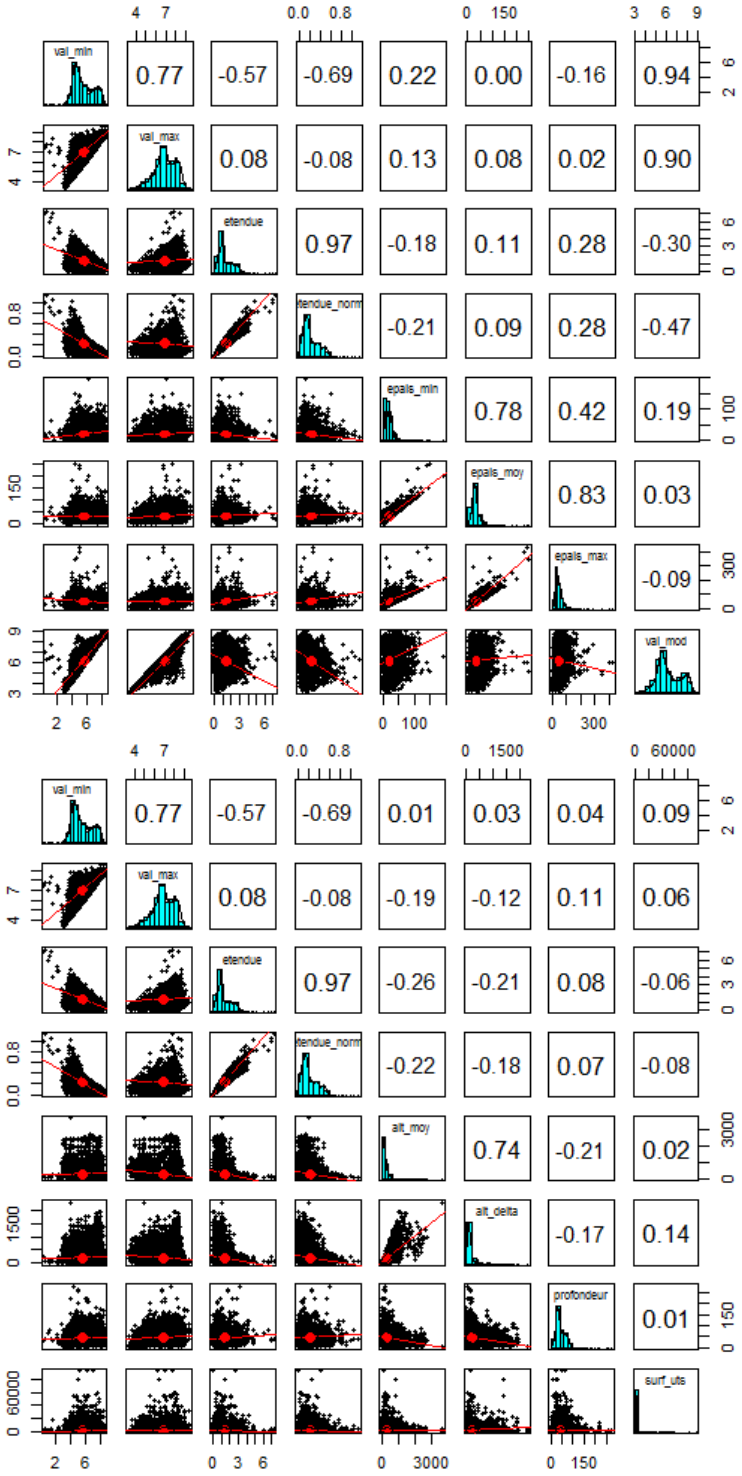


Figure 2: Scatterplot pour le pH

Pour les deux figures, on a :

En haut à gauche les variables val_min, val_max, etendue, etendue_norm, alt_max, alt_min, surf_unit et classe_mat1.

En haut à droite, val_min, val_max, etendue, etendue_norm, forme_morpho, prof_appar_min, prof_appar_moy et prof_appar_max.

En bas à gauche, val_min, val_max, etendue, etendue_norm, epais_min, epais_moy, epais_max et val_mod.
En bas à droite, val_min, val_max, etendue, etendue_norm, alt_moy, alt_delta, profondeur et surf_uts.
On peut déjà remarquer de fortes corrélations.
Pour le carbone on a une corrélation de 0.99 entre val_mod et val_min, de 0.97 entre val_mod et val_max et de 0.80 entre etendue et val_mod.
Pour le pH, on a une corrélation de 0.94 entre val_mod et val_min et de 0.90 entre val_max et val_mod. On remarque une corrélation négative de -0.30 entre etendue et val_mod et de -0.47 entre etendue_norm et val_mod. Ce qui signifie que plus la valeur modale augmente plus l'étendue à tendance à diminuer.

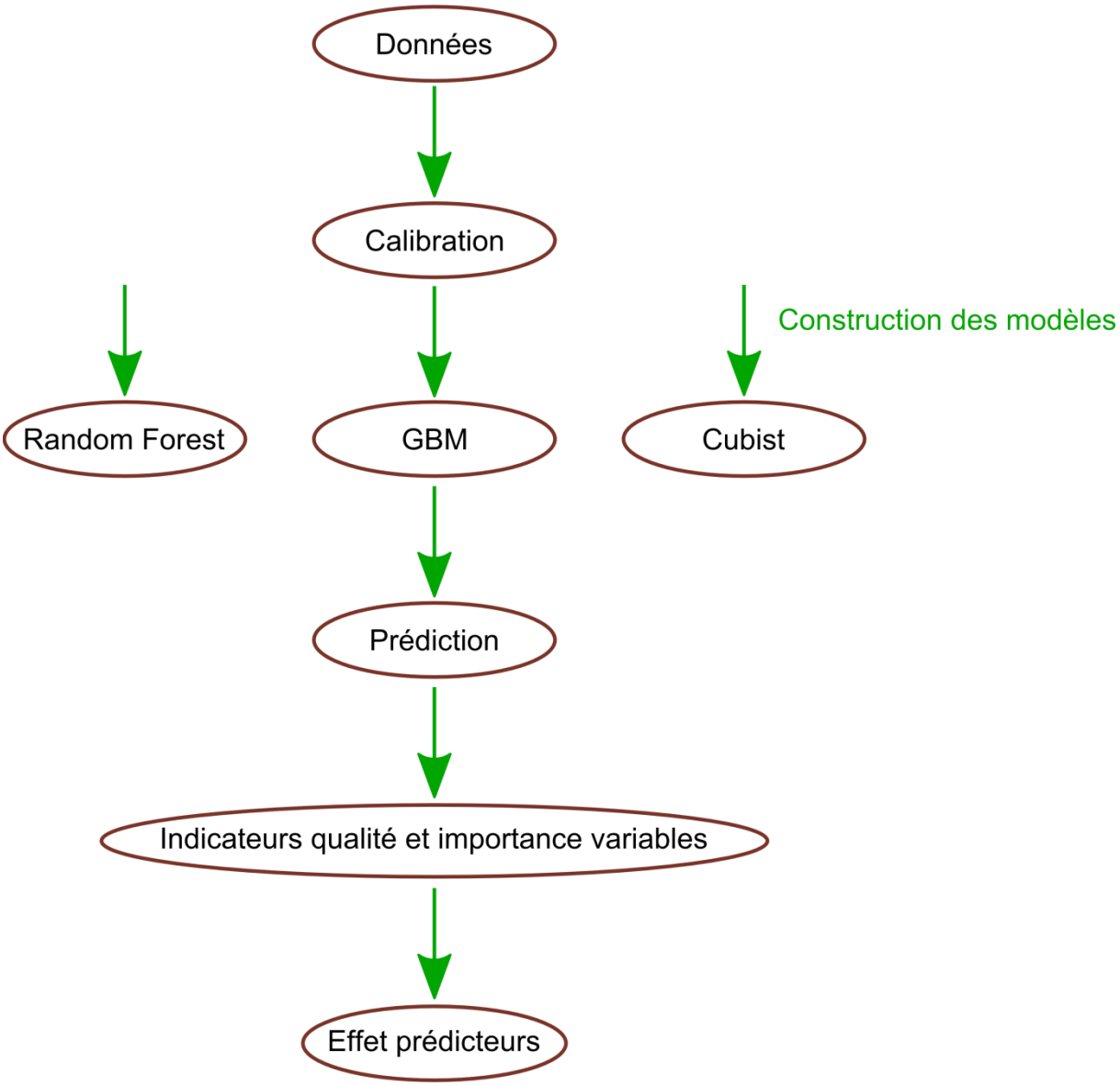


Figure 3: Présentation de la méthode d'analyse

La Figure 3 présente l'enchaînement de la méthode d'analyse des données. A partir des bases de données remaniées, on effectue une calibration puis la construction des modèles à l'aide de trois méthodes différentes. On effectue ensuite la prédiction sur ces différents modèles afin de pouvoir calculer des indicateurs de qualité mais aussi l'importance de chaque covariable. Pour finir, on s'intéresse à l'effet des prédicteurs. Ces différentes méthodes sont présentées ci-dessous.

2.2. Méthodes statistiques

L'analyse est réalisée avec trois outils de fouille de données, Random Forest, Generalized Boosted Regression Models (GBM) et Cubist. Ces méthodes sont utilisées en science du sol dans le contexte du « digital soil mapping » ou « cartographie numérique des sols ».
L'utilisation de Cubist est moins répandue.
Voici la description de ces trois outils.

2.2.1. Random Forest

Cet algorithme, Random Forest (ou Forêts aléatoires), est une amélioration de l'algorithme bagging, proposée par Breiman (2001). Cette amélioration est l'ajout d'une « randomisation ».
Il a pour objectif de rendre plus indépendants les arbres de l'agrégation en ajoutant du hasard dans le choix des variables qui interviennent dans les modèles.
Cette méthode semble plus efficace lorsque le nombre de variables explicatives p est très important. En effet, la variance de la moyenne de $\boldsymbol{N}$ variables indépendantes et identiquement distribuées, chacune de variance $\boldsymbol{\sigma^2}$, est $\boldsymbol{\sigma^2/N}$. Si les variables sont identiquement distribuées mais avec une corrélation ρ des variables prises deux à deux, la variance de la moyenne devient $\boldsymbol{\rho\sigma}^2 + \frac{\mathbf{1-\rho}}{\boldsymbol{N}}\,\boldsymbol{\sigma}^2$.
Comme pour le premier cas, le deuxième terme décroît avec $\boldsymbol{N}$.

L'algorithme est le suivant :

Algorithme 1 : Fonctionnement de Random Forest

Soit x_0 à prévoir et $z = \{(x_1,y_1),\ldots, (x_n,y_n)\}$ un échantillon
Pour** b = 1 à N **faire
Tirer un échantillon bootstrap $\boldsymbol{z_b^*}$
Estimer un arbre sur cet échantillon avec randomisation des variables : la recherche de chaque nœud optimal est précédé d'un tirage aléatoire d'un sous échantillon de m prédicteurs.
Fin Pour
Calculer l'estimation moyenne $\widehat{\boldsymbol{\phi}}_{\boldsymbol{N}}(\boldsymbol{x_0}) = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{\boldsymbol{b=1}}^{\boldsymbol{N}} \widehat{\boldsymbol{\phi}}_{\boldsymbol{z_b}}(\boldsymbol{x_0})$ ou le résultat du vote.

Par défaut, c'est le nombre minimum d'observation par nœuds qui limite la taille de l'arbre, il est fixé à 5 par défaut. Ce sont donc des arbres plutôt complets qui sont considérés, chacun de faible biais mais de variance importante.
La sélection aléatoire d'un nombre réduit de $\boldsymbol{m}$ prédicteurs potentiels à chaque étape de construction d'un arbre, augmente significativement la variabilité en mettant en avant nécessairement d'autres variables. Chaque modèle de base est évidemment moins performant, mais la réunion de tous conduit finalement à de bons résultats.
L'évaluation itérative de l'erreur out-of-bag permet de contrôler le nombre $\boldsymbol{N}$ d'arbres de la forêt et éventuellement d'optimiser le choix de $\boldsymbol{m}$.

2.2.2. Generalized Boosted Regression Models (GBM)

L'algorithme Generalized Boosted Regression Models (GBM), a été mis en œuvre par Friedman (2002) tout d'abord sous le nom Multiple Additive Regression Trees (MART) puis sous celui de GBM. Ce modèle fait partie d'une famille d'algorithmes basés sur une fonction perte supposée différentiable notée Q. Le principe de cet algorithme est de construire une séquence de modèles afin qu'à chaque étape, chaque modèle ajouté à la combinaison, apparaisse comme un pas vers une meilleure solution. L'innovation apportée concerne le pas effectué à chaque étape, qui est franchi dans la direction du gradient de la fonction de perte lui-même approché par un arbre de régression.

L'algorithme pour le cas de la régression est le suivant :

Algorithme 2 : Fonctionnement de GBM

Soit x_0

$$\widehat{\phi_0} = arg\ min_\gamma \sum_{i=1}^{n} Q(y_i, \gamma)$$

Pour *m=1 à M* ***faire***

Calculer $r_{im} = -\left[\frac{\delta Q(y_i, \phi(x_i))}{\delta \phi(x_i)}\right]_{\phi = \phi_{m-1}}$,

Ajuster un arbre de régression aux r_{mi} *donnant les feuilles terminales* R_{jm} *;* $j = 1, \ldots, J_m$.

Pour *m = 1 à M* ***faire***

Calculer $\gamma_{jm} = argmin_\gamma \sum_{x_i \in R_{jm}} Q(y_i, \hat{\phi}_{m-1} + \gamma)$.

Fin Pour

Mise à jour : $\hat{\phi}_m(x) = \hat{\phi}_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{1}\{x \in R_{jm}\}$.

Fin Pour

Résultat : $\hat{\phi}_M(x_0)$.

Explications de l'algorithme :

En entrée, on a un ensemble d'apprentissage $\{(x_i, y_i)\}_{i=1,\dots,n}$, une fonction de perte différentiable $Q(y, \gamma(x))$ et un nombre d'itérations M.
La première étape de cet algorithme est d'initialiser par un terme constant x_0, un arbre à une feuille puis d'initialiser le modèle $\widehat{\phi_0}$ avec une valeur constante.
Les r_{mj} correspondent aux résidus du modèle à l'étape précédente.
Les γ_{jm} sont les termes correctifs, optimisés pour chacune des feuilles R_{jm} définies par l'arbre de régression ajustant les résidus.
L'étape suivante consiste à faire la mise à jour du modèle.

Un coefficient de rétrécissement (ou shrinkage) ν, compris entre 0 et 1, peut être ajouté à la construction du modèle :

$$\hat{\phi}_m(x) = \hat{\phi}_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{1}\{x \in R_{jm}\}$$

Ce coefficient pénalise l'ajout d'un nouveau modèle dans l'agrégation et ralentit l'ajustement. Si sa valeur est petite, inférieure à 0.1, cela entraine un accroissement du nombre d'arbres, mais également une amélioration de la qualité de prévision. Le modèle devient plus performant mais moins généralisable à d'autres contextes.

Cet algorithme permet de réduire la variance ainsi que le biais. Il donne généralement de meilleurs résultats.

2.2.3. Cubist

Cubist est un modèle de régression axé prédiction qui combine les idées de modèle à base de règles, décrites dans Quinlan (1992) avec des corrections supplémentaires basées sur les voisins les plus proches dans l'ensemble d'apprentissage (voir Quinlan (1993) pour plus de détails)

Initialement, une arborescence est créée, mais chaque chemin de l'arbre termine par une règle. Un modèle de régression est adapté pour chaque règle basée sur un sous ensemble de données définis par les règles. L'ensemble des règles est élagué ou éventuellement combiné. Les variables candidates pour les modèles de régression linéaire sont les prédicteurs qui ont été utilisés dans la partie de la règle qui a été élaguée.
Cubist généralise ce modèle pour ajouter du « boosting » (lorsque le paramètre committees est supérieur à 1) et des corrections à base d'instances. Le nombre d'instances est fixé au temps de la prédiction par l'utilisateur et n'est pas nécessaire pour la construction du modèle.

Cette fonction lie R à la version GPL du code C RuleQuest. Le code RuleQuest différencie des valeurs manquantes à partir de valeurs qui ne sont pas applicables. Actuellement, ce package ne fait pas une telle distinction (toutes les valeurs sont traitées comme manquantes). Ceci va produire les résultats légèrement différents.

2.3. Calibration et validation

L'ensemble de ce travail est réalisé en deux temps, la calibration et la validation.

2.3.1. Calibration

Dans un premier temps, comme on peut le voir sur la Figure 3, une étape de calibration est effectuée selon l'algorithme 3. Cette première étape est nécessaire pour le choix des paramètres des différentes méthodes afin qu'elles donnent les meilleurs résultats possibles.

Algorithme 3 : Fonctionnement de la calibration

Définir des ensembles de valeurs de paramètres du modèle pour évaluer
Pour *chaque ensemble de paramètre* ***faire***
    Pour *chaque itération de rééchantillonnage* ***faire***
        Mettre de côté des échantillons spécifiques
        [Facultatif] Prétraiter les données
        Ajuster le modèle sur le reste des échantillons
        Prédire les échantillons mis de côté
    Fin
Calculer la performance moyenne à travers les prédictions
Fin
Déterminer l'ensemble de paramètres optimal
Ajuster le modèle final à toutes les données d'apprentissage en utilisant l'ensemble de paramètres optimal

Dans cet algorithme, 80% des données sont utilisées comme base d'apprentissage et 20% restants comme base de validation. Les paramètres sont sélectionnés en fonction de la $\boldsymbol{RMSE}$:

$$\boldsymbol{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

Avec $\hat{y}_i$ la valeur prédite au temps i, n le nombre de prédictions et y_i la valeur observée.

Une sortie graphique est disponible pour l'interprétation de l'influence des paramètres sur la qualité de prédiction ($\boldsymbol{RMSE}$) du modèle.

2.3.2. Validation

Dans un deuxième temps, on effectue la construction des modèles en réalisant une deuxième validation croisée avec la méthode k-fold. On fait une seconde validation croisée pour avoir une gamme de valeurs la plus homogène possible.
La méthode k-fold consiste à diviser le jeu de données en k échantillons, on sélectionne ensuit un des k échantillons comme ensemble de validation et les k-1 autres échantillons constituent l'ensemble d'apprentissage. On répète l'opération k fois de façon à ce que chaque échantillon serve une fois pour la validation.
Pour illustrer cette méthode, voici un schéma avec k égal à 3 :

Itération 1 :	Validation	Apprentissage	Apprentissage
Itération 2 :	Apprentissage	Validation	Apprentissage
Itération 3 :	Apprentissage	Apprentissage	Validation

Figure 4: schéma explicatif de la méthode k-fold

Pour la création des k échantillons, on utilise l'algorithme cvFolds (cvTools : Cross-validation tools for regression models. Andreas Alfons (2012). R package version 0.3.2 URL http://CRAN.R-project.org/package=cvTools). On choisit le pourcentage p du jeu de données servant comme échantillon de validation. Puis, on calcule le nombre de blocs $\boldsymbol{nb}$ (d'échantillons) à partir de $\boldsymbol{p}$:

$$\boldsymbol{nb} = (\text{taille du jeu de données})/(\text{taille du jeu de données} * \boldsymbol{p})$$

On construit ensuite les k blocs de façon aléatoire.

2.4. Analyse des variables sur le modèle

2.4.1. Importance des variables

Pour chacun des modèles, l'importance des covariables utilisées pour la construction de celui-ci est calculée.
Cette importance des variables est disponible en sortie des modèles mais elle est calculée de manière différente suivant que ce soit le modèle Random Forest, GBM ou Cubist.
Elle est estimée de la manière suivante :

- Pour Random Forest, si un prédicteur est important dans le modèle actuel, alors en attribuant d'autres valeurs à ce prédicteur au hasard mais de manière réaliste (par exemple en permutant les valeurs du prédicteur sur l'ensemble du jeu de données, cela devrait avoir une influence négative sur la prédiction. On calcule donc le MSE (= $RMSE^2$) avec l'ensemble de données d'origine puis avec l'ensemble de données

permuté et on fait la différence. Ainsi plus la différence est élevée, plus le prédicteur est important.

- Pour GBM et Cubist un pourcentage est retourné.

2.4.2. Effet des covariables sur le modèle

Les quatre variables les plus importantes sont étudiées et leur influence sur le modèle final est interprétée et analysée graphiquement à travers la mise en relation entre les données observées et les données prédites par la covariable. La méthode utilisée pour produire ces graphiques est présentée dans l'algorithme 4 présenté ci-dessous.

Algorithme 4 : Représentation graphique de l'effet des variables sur le modèle
Création d'une grille contenant toutes les combinaisons des valeurs (déciles si variable quantitatives) des 4 variables les plus importantes. *Ajout à la grille les autres variables en leur attribuant la valeur médiane.* *Prédiction en fonction de cette grille.* *Calcul de la valeur médiane pour les prédictions.* *Graphique des valeurs prédites en fonction de chacune des 4 variables.*

2.4.3. Analyse de la performance du modèle

Afin de pouvoir savoir si un modèle est performant ou bien de pouvoir comparer des modèles entre eux, on utilise des indicateurs de qualité. Chacun de ces indicateurs se calculent en fonction de $\boldsymbol{x}$ les données observées et de $\boldsymbol{y}$ les valeurs prédites.
On calcule tout d'abord le coefficient de détermination ($\boldsymbol{R^2}$). Il permet de juger de la qualité de la prédiction. Le $\boldsymbol{R^2}$ est compris entre 0 et 1. Plus il est proche de 1, plus le modèle est de bonne qualité. Il se calcule de la manière suivante :

$$\boldsymbol{R^2(x,y) = cor(x,y)^2}$$

Ensuite le Mean Prediction Error ($\boldsymbol{MPE}$), il mesure le biais et doit être proche de 0. Il se calcule de la manière suivante :

$$\boldsymbol{MPE\ (x,y) = mean\ (x-y)}$$

Puis on détermine le $\boldsymbol{RMSE}$, il permet de mesurer la précision du modèle, plus il est faible, plus le modèle est de bonne qualité. Il se calcule de la manière suivant :

$$\boldsymbol{RMSE(x,y) = \sqrt{mean((x-y)^2)}}$$

Enfin on détermine le Ratio of Performance to Deviation ($\boldsymbol{RPD}$), il s'agit de la performance moyenne de prédiction, cela permet de comparer les modèles entre eux. Plus le $\boldsymbol{RPD}$ est élevé, plus le modèle est de bonne qualité. Il se calcule de la manière suivante :

$$\boldsymbol{RPD(x,y) = sd(x)/RMSE(x,y)}$$

2.5. Mise en application, algorithme

L'ensemble de ce travail a été réalisé avec le logiciel R (R Core Team 2013). Les principaux packages utilisés sont *randomForest*, *gbm*, *Cubist*, *caret*, *ggplot2* et *cvTools*.
Dans le but de partager ce travail, trois fonctions ont été développées pour généraliser l'application de la méthode à d'autres sujets d'études. Leur fonctionnement est présenté ci-dessous :

2.5.1. Fonction de calibration

La première fonction, que j'ai nommé FRodas, permet de sélectionner les paramètres optimaux des modèles (voir partie 2.3). Elle utilise la fonction train du package *caret*.
Cette fonction, FRodas, prend en entrée :

Tableau 2: Description des paramètres d'entrée de la fonction FRodas

Nom du paramètre d'entrée	Description
titre	Nom que l'on veut donner au fichier de sauvegarde
datacpt	base de données
folderSave	Nom du chemin permettant d'accéder au dossier où l'on veut sauvegarder
model	Nom du modèle (rf, gbm ou cubist)
transfParam	choix de la transformation des données, aucune ou log (ici nous n'en avons pas utilisé)
tuneGrid	grille contenant les valeurs de réglage possibles
trControl	liste de valeurs qui définissent comment la fonction train agit
vNames	Nom des variables avec en premier la variable à expliquer puis ensuite les variables explicatives

En sortie de cette fonction, on obtient une sauvegarde, dans le dossier choisi en paramètre d'entrée, du modèle obtenu par la fonction *train*.

Algorithme 5 : Description de la fonction FRodas

Si *(transfParam == 'log')* ***Alors***
    Transformer les données
Fin
Si *(model == 'rf')* ***Alors***
    On enlève les lignes ayant des valeurs manquantes
Fin
Sélection des paramètres optimaux avec la fonction train
Sauvegarde des résultats obtenus

La fonction *train* utilise en paramètres d'entrée une base de données contenant les valeurs des variables prédictives, les valeurs à prédire, le modèle voulu (par exemple, Random Forest, GBM ou Cubist), une grille contenant toutes les combinaisons de paramètres possibles et une liste de valeurs qui définissent comment la fonction *train* agit.
Le code de cette fonction est présenté en Annexe 2.

2.5.2. Fonction de construction des modèles

La deuxième fonction, que j'ai nommé FLocalML, permet de construire les modèles avec validation croisée en utilisant les paramètres obtenus à l'aide de la fonction précédente. Elle utilise les packages *randomForest*, *gbm*, *Cubist* et *cvTools*.
Cette fonction prend en entrée :

Tableau 3: Description des paramètres d'entrée de la fonction FLocalML

Nom du paramètre d'entrée	Description
titre	Nom que l'on veut donner au ficher de sauvegarde
datacpt	base de données
folderSave	Nom du chemin permettant d'accéder au dossier où l'on veut sauvegarder
model	Nom du modèle (rf, gbm, cubist)
nbl	Nombre d'itérations
prob	pourcentage du jeu de données pour la validation (ici 0.2)
transfParam	choix de la transformation des données, aucune ou log (ici aucune)
cTune	modèle obtenu en sortie de la fonction FRodas
vNames	Nom des variables avec en premier la variable à expliquer puis ensuite les variables explicatives

En sortie de cette fonction, on a une sauvegarde des modèles obtenus à chaque itération, un tableau avec les valeurs prédites pour chaque itération ainsi que les valeurs observées. On a également les importances des variables.

Algorithme 6 : Description de la fonction FLocalML

On enlève les lignes ayant des valeurs manquantes, on obtient la base d
*Détermination du nombre de blocks (k-fold) : nbrK = nrow(d)/(nrow(d)*prob)*
Création tableau contenant les valeurs prédites pour chaque itération
Création du tableau contenant les importances des variables pour chaque itération
Création des blocks
Pour** (i in 1 :nbl) **Faire
    Pour** (b in 1 : nbrK) **Faire
        Déterminer les bases de test et d'apprentissage
        Tester s'il s'agit de rf, gbm ou cubist
        Faire tourner le modèle correspondant sur la base d'apprentissage
        Sauvegarder le modèle
        Calculer l'importance de variables
        Faire la prédiction sur la base de test
    Fin
Fin
Combiner les valeurs prédites avec les valeurs observées correspondantes

Le code de la fonction est présenté en Annexe 2.

2.5.3. Fonction de représentation graphique de l'effet des variables

La troisième fonction, FPlot permet de tracer l'effet des quatre variables les plus importantes. Elle utilise le package *ggplot2*.
Cette fonction prend en entrée :

Tableau 4: Description des paramètres d'entrée de la fonction FPlot

Nom des paramètres d'entrée	Description
grille	grille contenant les combinaisons des valeurs des 4 variables les plus importantes
vNames	liste contenant le nom des autres variables prédictives
d	base de données
model	modèle à utiliser pour la prédiction
nameModel	nom du modèle utilisé, rf, gbm ou cubist
neighbors	paramètre à renseigner pour la prédiction si le modèle utilisé est cubist, 0 par défaut

En sortie de cette fonction, on a sur une même page les graphiques représentant l'effet des variables sur la prédiction d'une certaine variable.
Pour l'algorithme de cette fonction, voir l'algorithme 4 de la partie 2.4.2.
Le code de cette fonction est présenté en Annexe 2.

3. Résultats

3.1. Statistiques générales sur les variables étudiées

Les statistiques générales des différentes variables sont présentées dans le Tableau 5 et Tableau 6. Ces tableaux présentent la valeur minimale, le premier quartile, la médiane, la moyenne, le troisième quartile, la valeur maximale et la variance de toutes les variables quantitatives pour le carbone et le pH. On note que les minima des profondeurs d'apparition sont négatifs. Cela correspond à une convention de notation dans la base de données pour décrire ce que l'on appelle les humus qui correspondent à des couches exclusivement organiques (litières, etc.) présentes sous forêt.

Tableau 5: statistiques générales pour les variables quantitatives du Carbone

	Min	1er Quartile	Médiane	Moyenne	3e Quartile	Max	Variance
alt_max	12.0	105.0	161.0	261.0	270.0	3320.0	108690.1
alt_min	0.0	5.0	47.0	107.9	116.0	2160.0	36880.2
surf_unit	0.0	1618.0	4410.0	7807.0	9616.0	100900.0	106770441.0
prof_appar_min	-21.0	0.0	5.0	12.1	20.0	115.0	269.0
prof_appar_moy	-10.0	0.0	20.0	21.8	35.0	153.0	588.4
prof_appar_max	-5.0	0.0	30.0	32.9	60.0	197.0	1323.8
epais_min	0.0	8.0	10.0	15.7	20.0	140.0	200.0
epais_moy	0.0	25.0	32.0	35.4	40.0	250.0	463.3
epais_max	2.0	40.0	54.0	63.4	75.0	450.0	1767.5
val_mod	0.2	6.3	11.7	25.6	25.8	700.0	4887.6
alt_moy	6.0	60.5	102.5	184.5	192.5	2660.0	65310.5
alt_delta	4.0	70.0	110.0	153.2	170.0	1960.0	29898.5
profondeur	2.5	30.0	37.0	46.3	60.0	271.5	743.0
surf_uts	0.0	180.9	452.6	1127.0	1162.0	53760.0	4888680.0
val_min	0.0	1.8	5.4	14.3	14.6	530.0	2260.3
val_max	0.9	12.2	22.5	43.9	44.9	900.0	7687.4
etendue	0.1	8.0	16.3	29.6	28.9	464.2	2218.9
etendue_norm	0.0	0.9	1.4	1.5	1.9	20.6	1.3

Tableau 6: statistiques générales pour les variables quantitatives pour le pH

	Min	1er Quartile	Médiane	Moyenne	3e Quartile	Max	Variance
alt_max	0.0	133.0	220.0	398.4	438.0	4800.0	227789.2
alt_min	0.0	45.0	118.0	197.7	215.0	2500.0	87679.6
surf_unit	0.0	1211.0	3685.0	7883.0	9041.0	300300.0	206890256.0
prof_appar_min	-20.0	0.0	10.0	17.9	25.0	305.0	492.1
prof_appar_moy	-10.0	0.0	25.0	25.5	40.0	320.0	763.2
prof_appar_max	-5.0	0.0	30.0	33.7	60.0	335.0	1277.4
epais_min	0.0	10.0	20.0	20.2	25.0	195.0	226.9
epais_moy	0.0	21.0	30.0	33.2	40.0	250.0	349.8
epais_max	2.0	30.0	40.0	50.8	60.0	430.0	1105.0
val_mod	3.2	5.5	6.0	6.3	7.3	9.0	1.4
alt_moy	0.0	95.0	170.0	298.0	332.5	3650.0	142011.5
alt_delta	0.0	60.0	113.0	200.7	204.0	2300.0	62891.4
profondeur	2.0	30.0	38.0	45.9	60.5	275.0	657.9
surf_uts	0.0	205.8	588.3	1768.0	1808.0	93110.0	14530132.0
val_min	0.5	4.5	5.2	5.6	6.7	8.6	1.8
val_max	3.4	6.4	7.0	6.9	7.8	9.5	1.2
etendue	0.0	0.8	1.0	1.4	2.0	7.2	0.7
etendue_norm	0.0	0.1	0.2	0.2	0.3	1.1	0.0

Les teneurs moyennes en carbone sont de 25.6 avec des valeurs minimales et maximales comprises entre 0 et 900. Ceci montre qu'il subsiste des erreurs de saisie dans la base de données initiale : une teneur de 900 pour mille en carbone dans le sol est théoriquement impossible. Il n'entrait pas dans le cadre de mon stage de corriger ces erreurs, mais leur détection sera utile pour la poursuite des travaux d'InfoSol. On note également une forte différence entre la médiane et la moyenne qui traduit une dissymétrie de la distribution.

Le pH moyen est de 6.3 avec des valeurs minimales et maximales comprises entre 0.5 et 9.5.

Pour les deux variables restantes, variables qualitatives classe_mat1 et forme_morpho, le Tableau 7 présente l'effectif par classe pour le carbone et le pH.

Tableau 7: effectifs par classe des variables qualitatives pour le carbone et le pH

carbone				pH			
classe_mat1		forme_morpho		classe_mat1		forme_morpho	
classe	*effectif*	*classe*	*effectif*	*classe*	*effectif*	*classe*	*effectif*
1	1	autre	551	1	73	autre	1015
2	1715	butte témoin	156	2	2737	butte témoin	232
3	18	chenal et alluvion	160	3	85	chenal et alluvion	1450
4	0	colluvionnement	138	4	55	colluvionnement	724
5	667	dépression	110	5	1124	dépôt éolien	484
6	33	dôme	177	6	447	dépression	261
7	204	éboulis	82	7	2021	dôme	264
8	2306	glacis	172	8	2840	éboulis	460
10	0	plaine	90	11	117	glacis	412
11	107	plateau	1658	12	260	karsts	247
12	210	replat	274	13	10567	moraine	602
13	3470	surface d'aplanissement	214	14	1264	plaine	413
14	541	surface structurale	85	99	469	plateau	3102
99	39	talweg	521			relief résiduel	205
		terrasse	218			replat	717
		vallée	1227			surface d'aplanissement	533
		versant	3105			surface structurale	549
		versant concave	164			talweg	712
		versant convexe	64			terrasse	1143
		versant convexo-concave	145			vallée	1621
						vallon	342
						versant	5468
						versant concave	306
						versant convexe	402
						versant convexo-concave	395

Pour le carbone, la classe la plus fréquente pour la variable classe_mat1 est la 13 avec un effectif de 3470. Pour la variable forme_morpho, la classe la plus fréquente est versant avec un effectif de 3105.
En ce qui concerne le pH, pour la variable classe_mat1, la classe la plus fréquente est également la 13 avec un effectif de 10567. Pour la variable forme_morpho, la classe la plus fréquente est également versant avec un effectif de 5468.

3.2. Performance des modèles

Je vais tout d'abord présenter les résultats liés à la calibration (choix des meilleurs paramètres). Pour la méthode Random Forest, un seul paramètre est à choisir, il s'agit de mtry (nombre de variables choisies à chaque division), on choisit de tester ici 5, 10 et 16. Pour la méthode GBM, le paramètre shrinkage est fixé à 0.1, le paramètre n.trees (correspond au nombre d'arbres ou d'itérations) est à choisir entre 1000, 1500 et 5000. Enfin le paramètre interaction.depth (profondeur maximale de l'interaction des variables) est à choisir entre 3, 5 et 9.

Pour la méthode Cubist, le paramètre committees est choisir entre 1, 10, 50 et 100. Le paramètre neighbors est à choisir entre 0, 1, 5 et 9. Comme précisé précédemment, pour chaque modèle, lors de la calibration, on obtient un graphique permettant de représenter la performance des différents choix de paramètre. Je vais ici exposer les graphiques obtenus pour les trois modèles de val_min pour le carbone (Figure 5) et les trois modèles de l'étendue pour le pH (Figure 6).

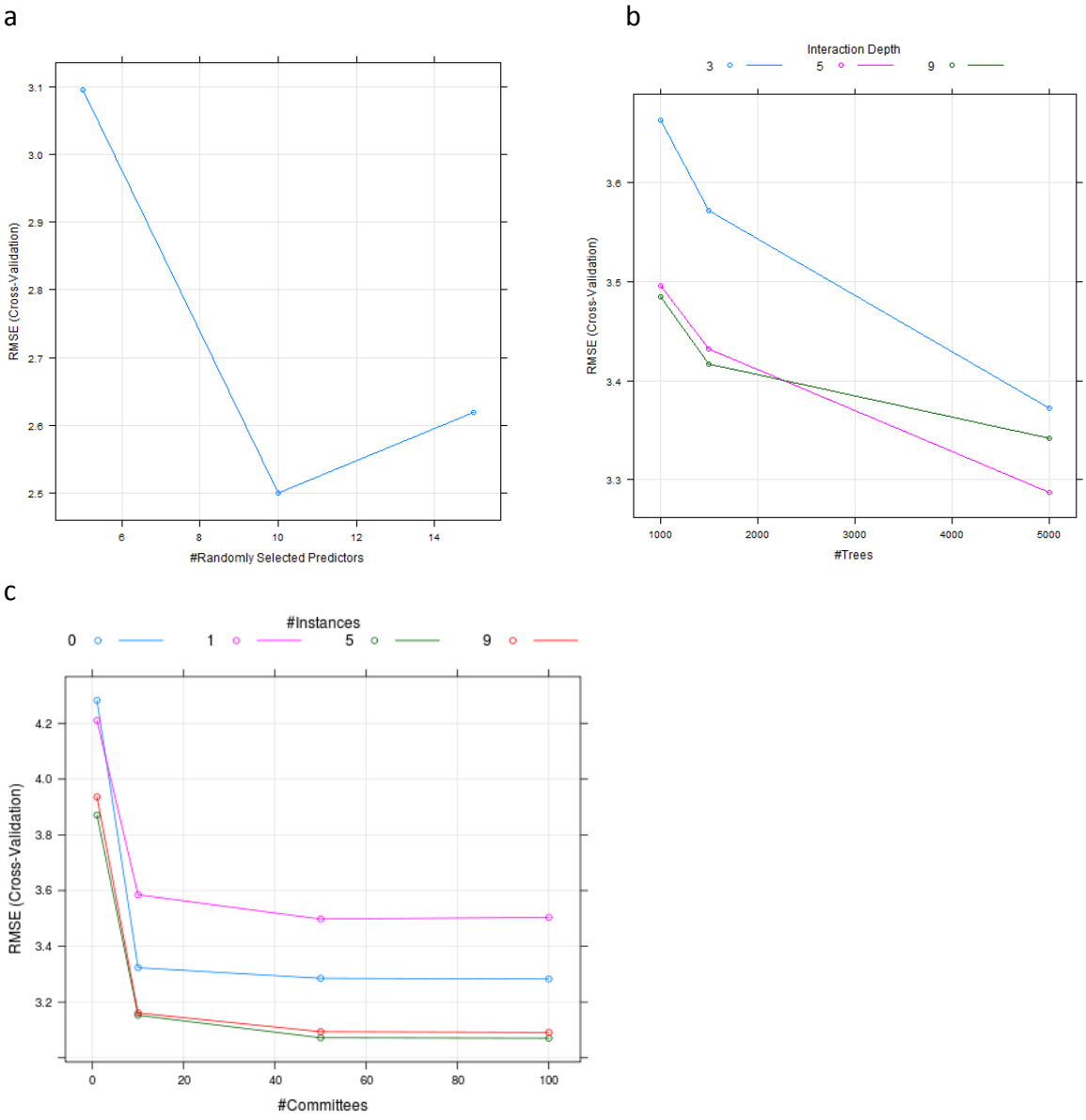


Figure 5: résultats de la calibration pour val_min du carbone pour Random Forest (a), GBM (b) et Cubist (c)

On constate que pour Random Forest, le meilleur paramètre est mtry égale à 10. Pour GBM, les meilleurs paramètres sont n.trees à 5000 et interaction.depth à 5. Pour Cubist, les meilleurs paramètres sont committees à 5 et neighbors à 100.

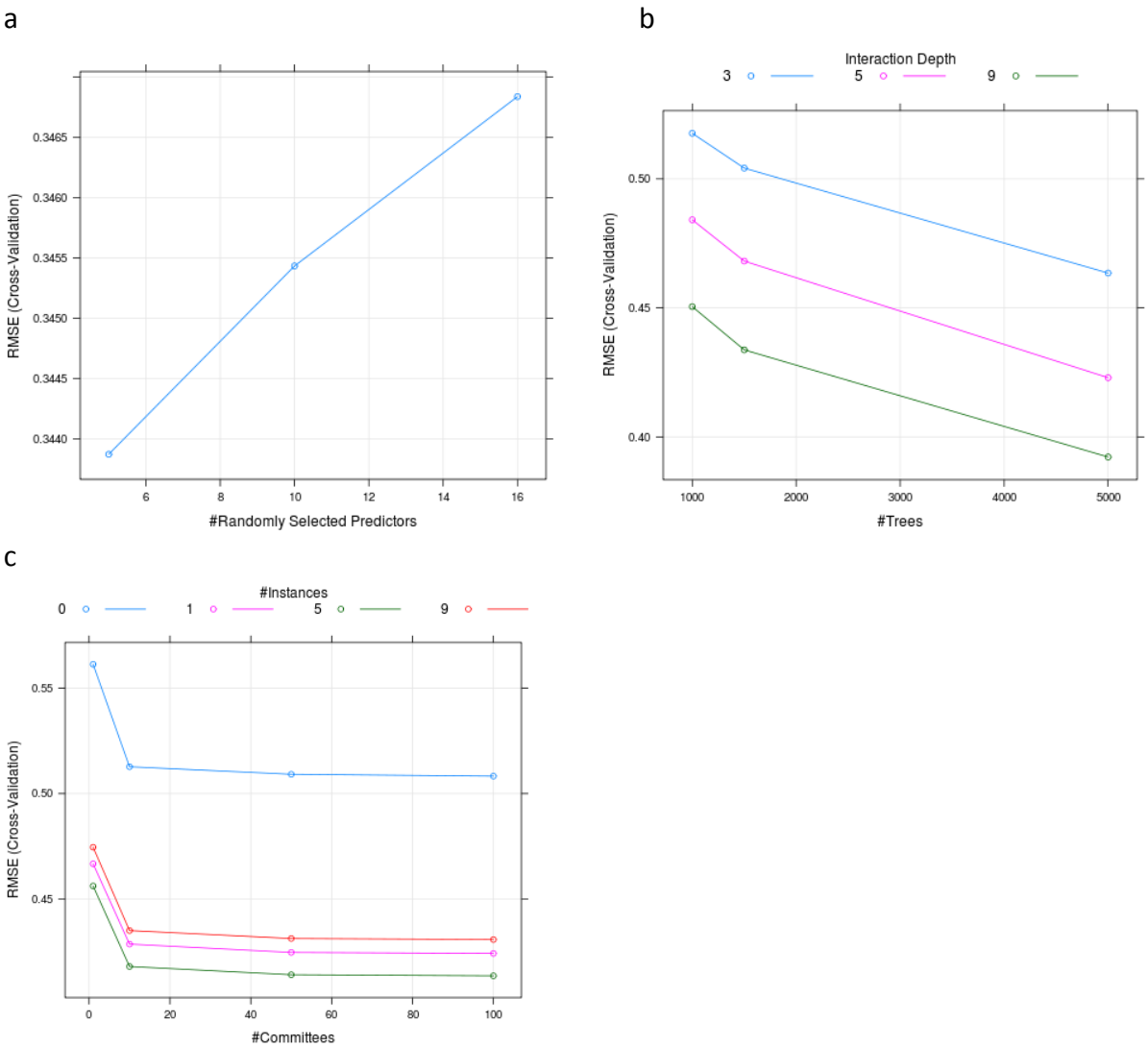


Figure 6: résultats de la calibration pour étendue du pH pour Random Forest (a), GBM (b) et Cubist (c)

On constate que pour Random Forest, le meilleur paramètre est mtry égale à 5. Pour GBM, les meilleurs paramètres sont n.trees à 5000 et interaction.depth à 9. Pour Cubist, les meilleurs paramètres sont committees à 5 et neighbors à 100.
Les graphiques pour les autres modèles sont présents en Annexe 3.

Les résultats obtenus suite à l'utilisation des différentes méthodes expliquées précédemment sont présentés dans le tableau suivant.

Tableau 8: Performance des modèles

		Carbone			pH		
		rf	gbm	cubist	rf	gbm	cubist
val_min	R^2	0.9939	**0.9973**	0.9971	**0.9684**	0.9655	0.9656
	MPE	0.0919	-0.0200	**0.0117**	0.0155	0.0012	**-0.0008**
	RMSE	3.09	**2.35**	2.53	**0.24**	0.25	0.25
	RPD	19.89	**21.76**	19.73	**5.61**	5.39	5.40
val_max	R^2	0.9903	**0.9907**	0.9886	**0.9636**	0.9631	0.9625
	MPE	**-0.0227**	-0.1738	-0.0332	0.0033	-0.0039	**0.0006**
	RMSE	**7.97**	8.12	8.90	**0.21**	0.21	0.21
	RPD	**11.99**	11.14	10.42	**5.25**	5.21	5.18
etendue	R^2	**0.9581**	0.9558	0.9548	**0.8260**	0.8254	0.8184
	MPE	**-0.0420**	-0.2691	-0.0723	0.0020	-0.0054	**-0.0005**
	RMSE	**9.36**	9.56	9.60	0.36	**0.35**	0.36
	RPD	**5.25**	5.16	5.22	2.36	**2.39**	2.35
etendue_norm	R^2	**0.7273**	0.7148	0.6816	**0.8649**	0.8625	0.8584
	MPE	**-0.0059**	-0.0179	-0.0103	0.0002	-0.0011	**-4.2280e-05**
	RMSE	**0.587**	0.598	0.638	**0.058**	0.058	0.059
	RPD	**1.98**	1.92	1.78	**2.70**	2.70	2.66

Le Tableau 8 récapitule les différents indicateurs de qualité pour les modèles obtenus (Random Forest, GBM et Cubist) pour le carbone et le pH.

Pour chaque variable pour le carbone et le pH, les meilleures valeurs d'indicateur parmi tous les modèles ont été mises en gras.

Je vais maintenant commenter les résultats présentés par ce tableau.

On peut remarquer tout d'abord que pour chacune des 8 variables à prédire, à savoir val_min, val_max, etendue et etendue_norm pour le carbone et le pH, les indicateurs des trois modèles sont assez proches.

En général, les indicateurs obtenus pour le carbone sont meilleurs que ceux du pH sauf pour la variable etendue_norm.

Parmi les 8 variables, c'est val_min pour le carbone qui obtient les meilleurs résultats avec un R^2 entre 0.9939 et 0.9973 et un RPD entre 19.73 et 21.76. Au contraire, la variable qui obtient les moins bons résultats est etendue_norm pour le carbone avec un R^2 entre 0.6816 et 0.7273 et un RPD entre 1.78 et 1.98.

Pour le carbone, c'est la variable val_min qui a les meilleurs résultats puis val_max, etendue et enfin etendue_norm. Pour le pH, c'est également la variable val_min qui a les meilleurs résultats puis val_max, etendue_norm et enfin etendue.

On constate que pour le carbone et le pH, une majorité des indicateurs les plus performants proviennent des modèles Random Forest mais cette différence n'est pas significative.

Afin d'étudier la qualité de la prédiction, j'ai tracé pour modèle les valeurs prédites en fonction des vraies valeurs avec pour chaque itération une couleur différente.
Je vais présenter ici seulement les graphiques des trois modèles obtenus pour val_min du carbone ainsi que ceux pour étendue du pH. Pour les autres modèles, les graphiques sont en Annexe 4.

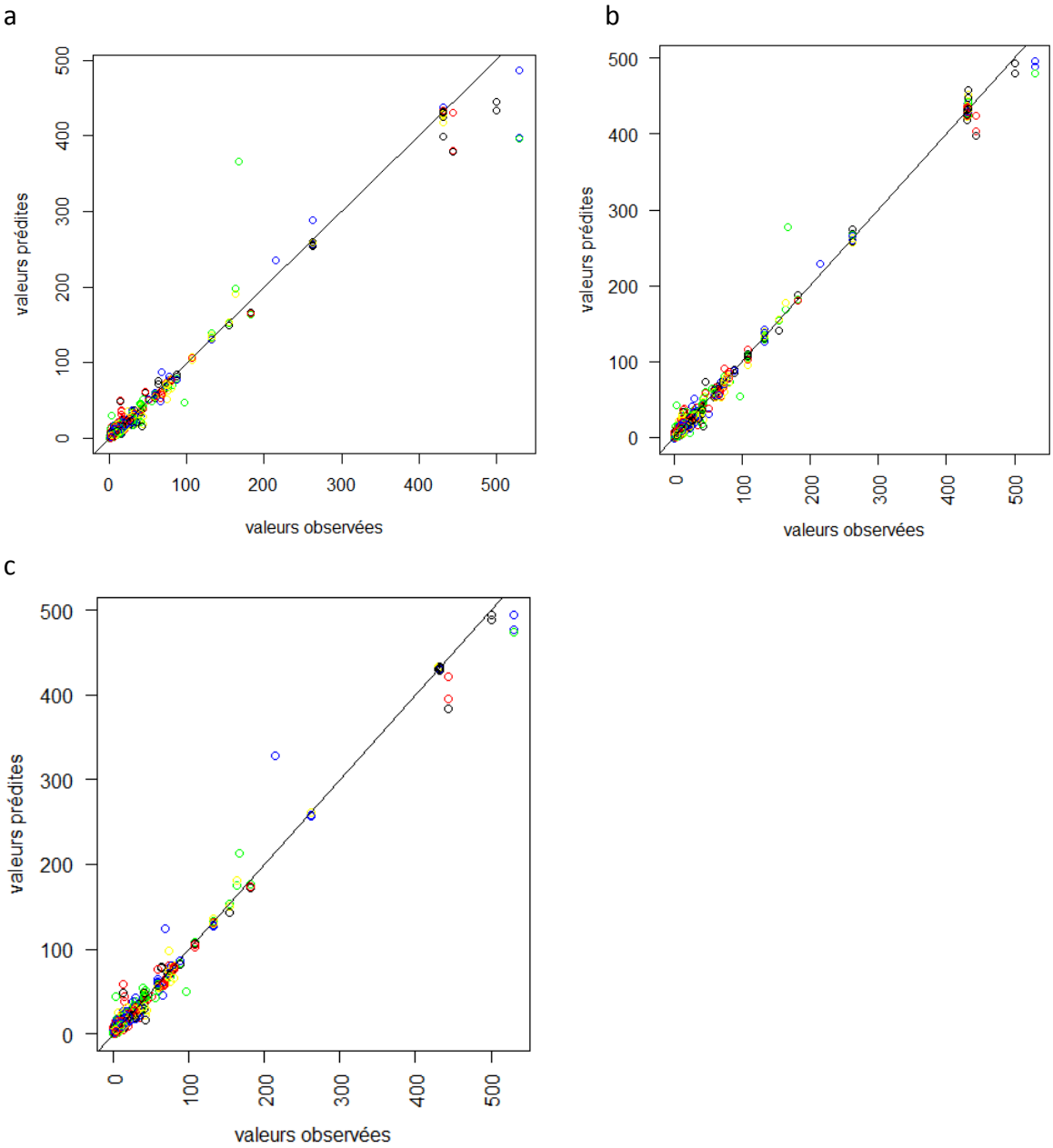


Figure 7: représentation des valeurs prédites en fonction des valeurs observées pour val_min du carbone pour Random Forest (a), GBM (b) et Cubist (c)

Pour ces trois graphiques, on remarque que le nuage de points est bien autour de la droite sauf quelques points. Ceci est cohérent avec les très bons indicateurs de qualité, on a donc pour cette variable de très bonnes prédictions.

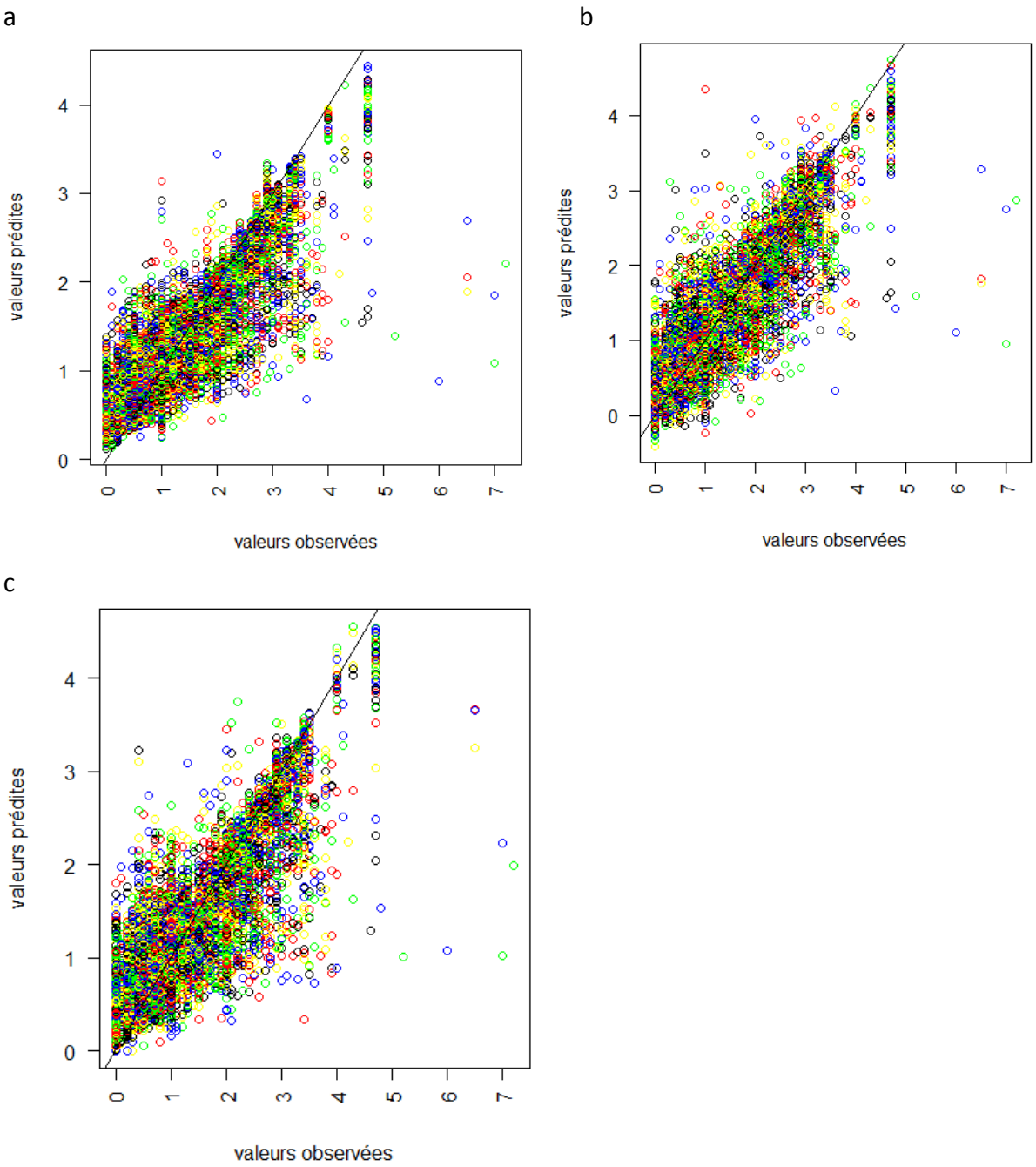


Figure 8: représentation des valeurs prédites en fonction des valeurs observées pour étendue du pH pour Random Forest (a), GBM (b) et Cubist (c)

En revanche pour ces trois graphiques, le nuage de points est un peu plus dispersé et en effet, les indicateurs de qualité sont moins bons que pour le cas précédent.

3.3. Importance des prédicteurs

Suite à la construction de tous ces modèles, l'importance des variables de chacun d'entre eux a été calculée. Je ne vais présenter ici que quelques graphiques d'importance des variables, le reste des graphiques est en Annexe 5.

Comme pour les graphiques précédents, je vais présenter dans un premier temps les graphiques concernant le carbone, pour val_min pour les trois méthodes Random Forest, GBM et Cubist.

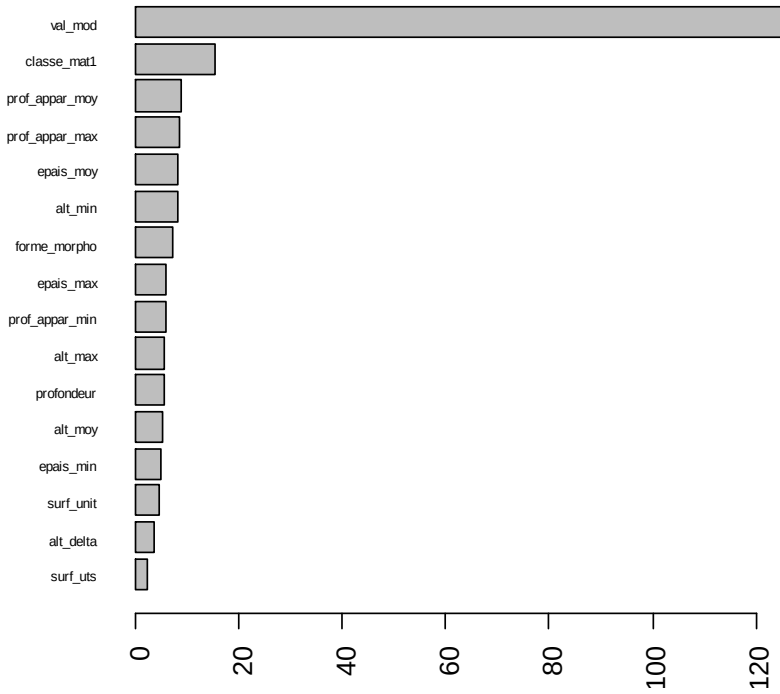


Figure 9: Random Forest pour val_min du carbone

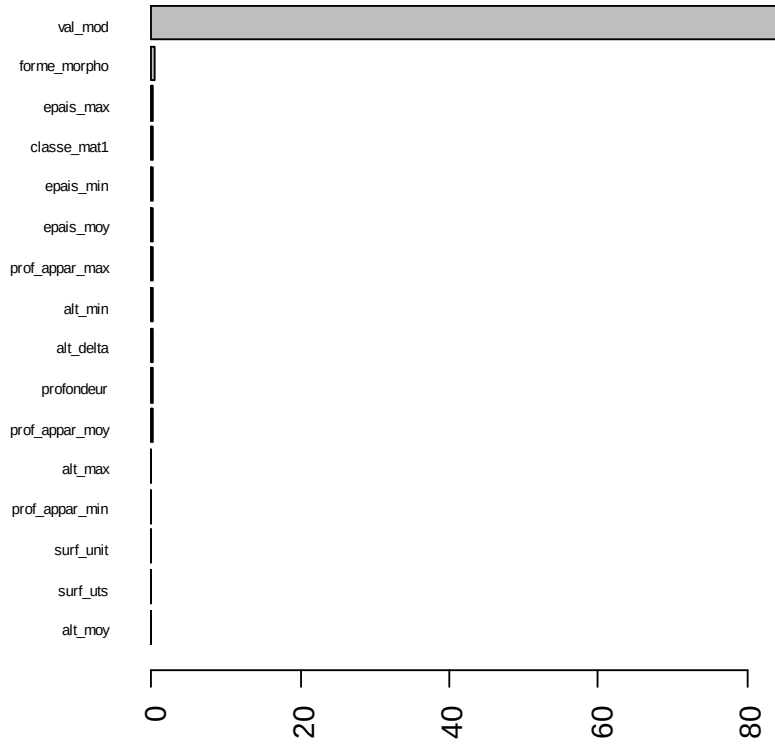


Figure 10: GBM pour val_min du carbone

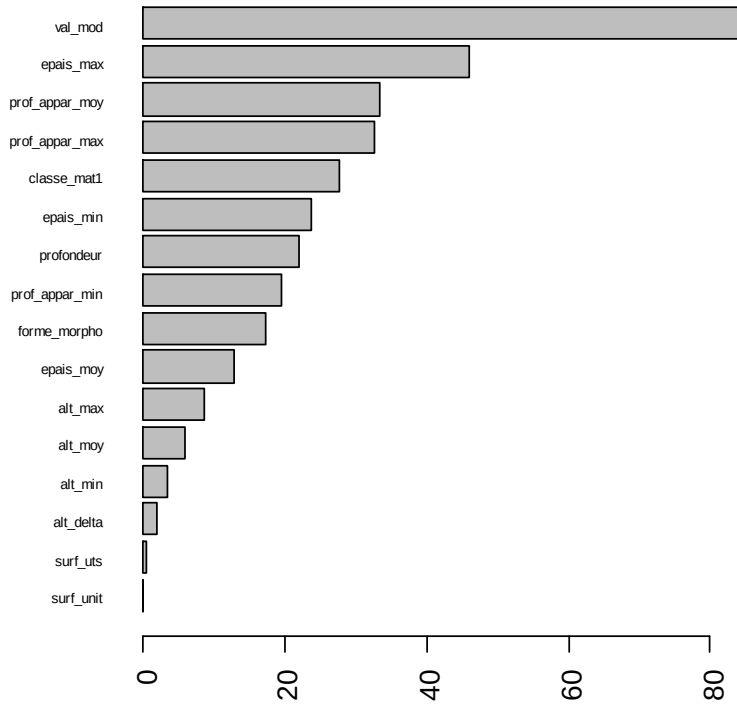


Figure 11: Cubist pour val_min du carbone

On constate que dans les trois cas, c'est la variable val_mod qui a le plus d'importance. On remarque également la différence de calcul des importances entre les trois méthodes. Par exemple pour gbm, la variable val_mod a beaucoup d'importance et toutes les autres ont une importance très faible alors que pour cubist il y a une meilleure répartition.

Dans un deuxième temps, je vais présenter les graphiques concernant le pH pour étendue.

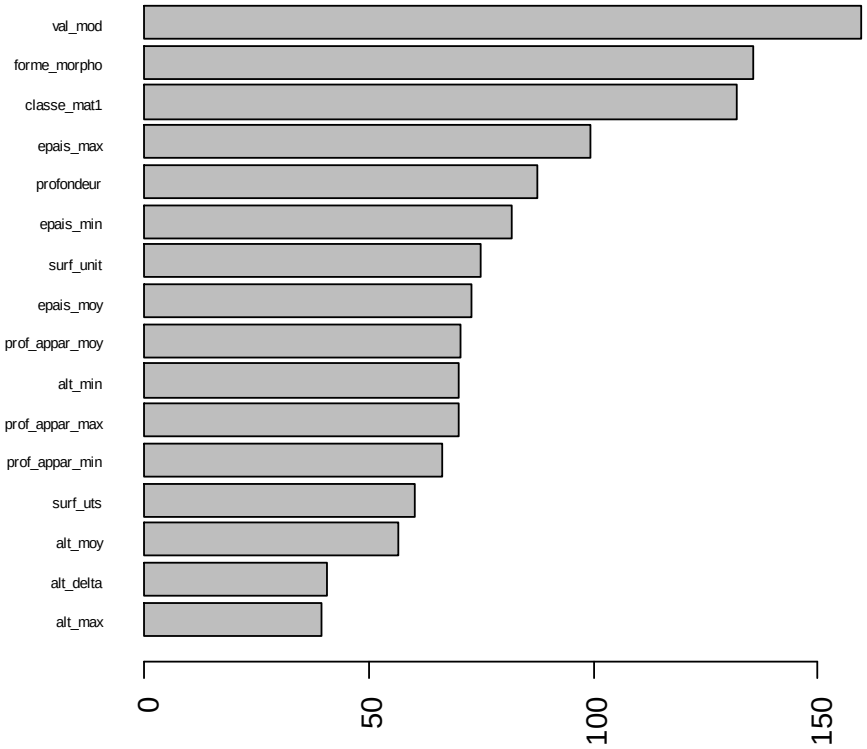


Figure 12: Random Forest pour étendue du pH

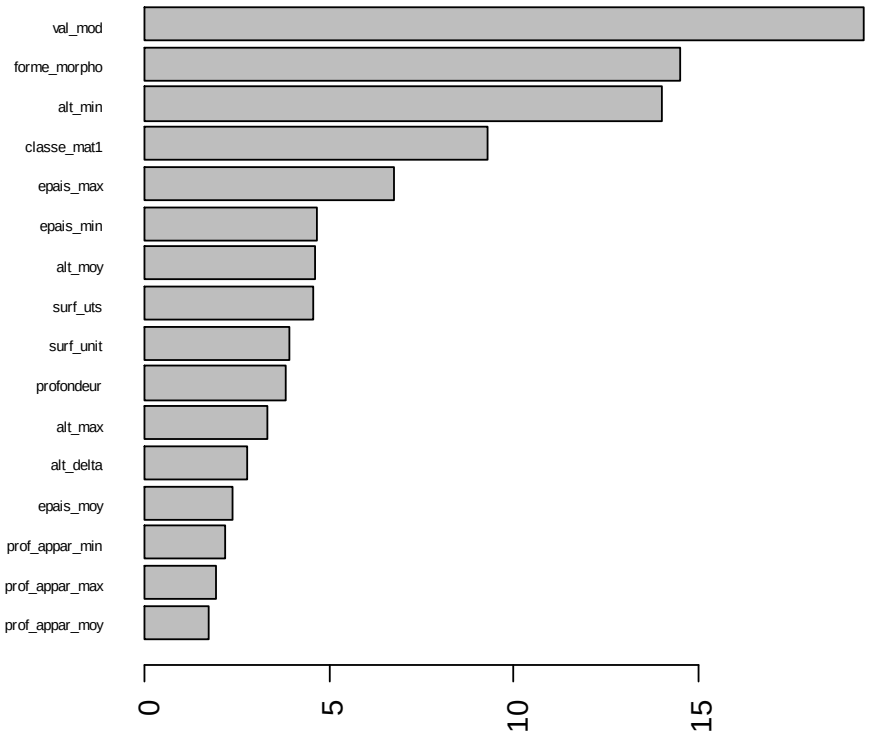


Figure 13: GBM pour étendue du pH

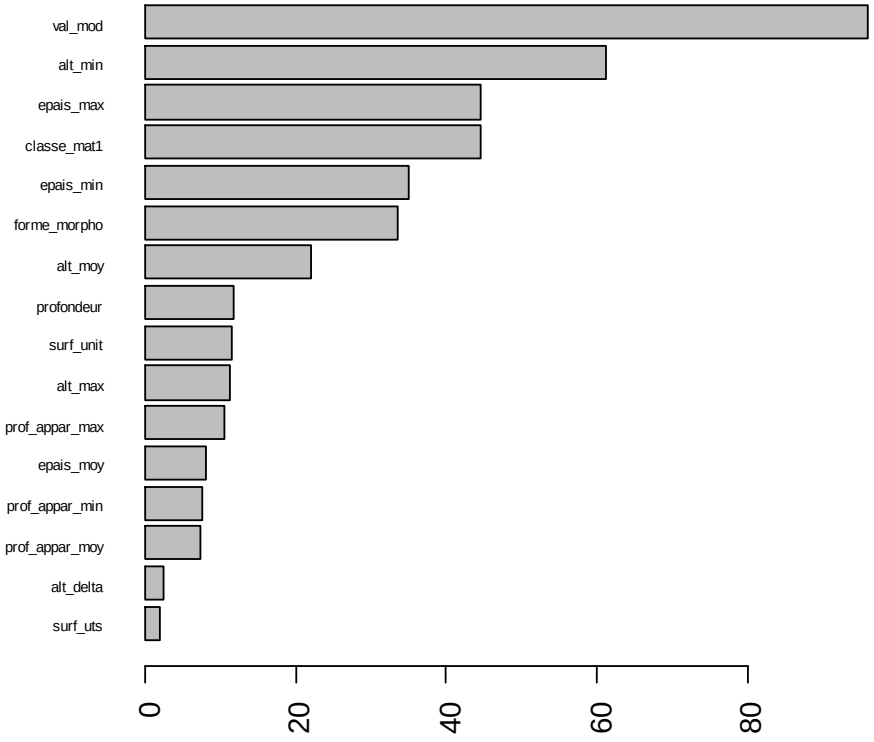


Figure 14: Cubist pour étendue du pH

On constate que dans les trois cas, comme pour le carbone c'est la variable val_mod qui a le plus d'importance. On remarque également une meilleure répartition des importances.

3.4. Effet des prédicteurs

D'après les graphiques précédents, pour chaque modèle, on a sélectionné à chaque fois les quatre meilleurs prédicteurs de façon à étudier leur influence sur la prédiction.

Ainsi pour la variable val_min du carbone, avec la méthode Random Forest, si l'on regarde la Figure 9, on constate que les quatre variables les plus importantes sont : val_mod, classe_mat1, prof_appar_moy et prof_appar_max. On a donc graphiques suivants :

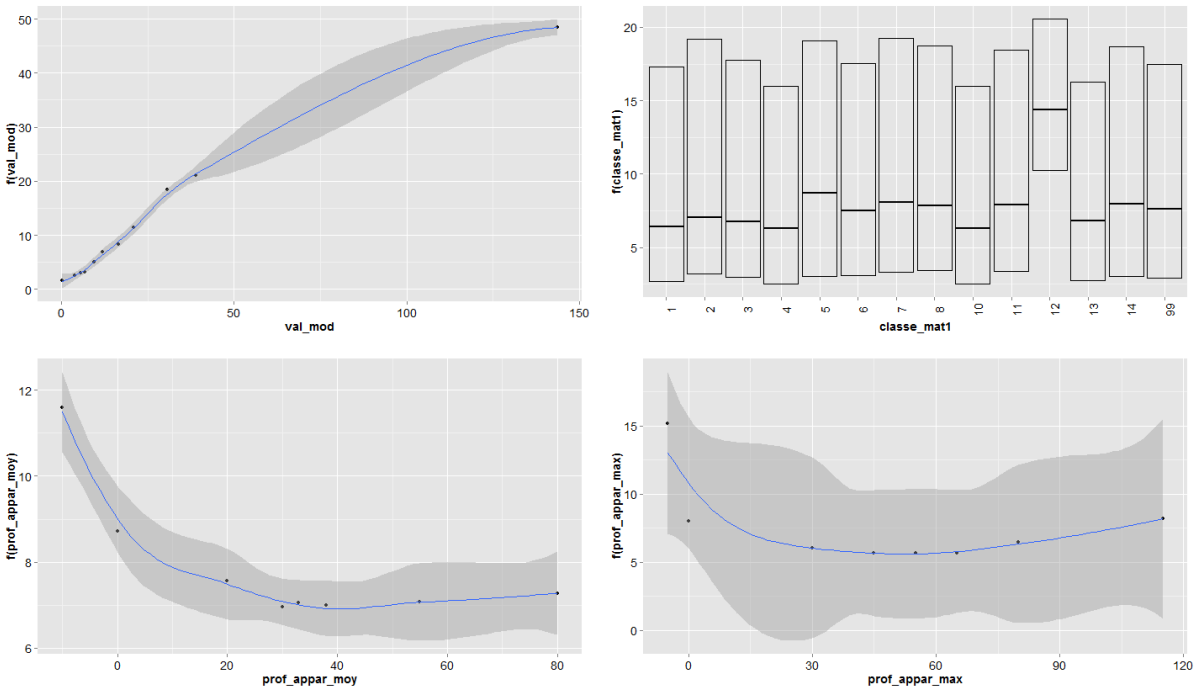


Figure 15: Effet des prédicteurs sur val_min pour le carbone pour la méthode Random Forest

Sur les graphiques, la zone grisée correspond à l'intervalle de confiance. Pour les variables qualitatives, on a pour chaque classe le premier quantile, la médiane et le troisième quantile.

Pour la variable val_min du carbone, avec la méthode GBM, les quatre variables les plus importantes sont : val_mod, forme_morpho, epais_max et classe_mat1. On obtient les graphiques suivants :

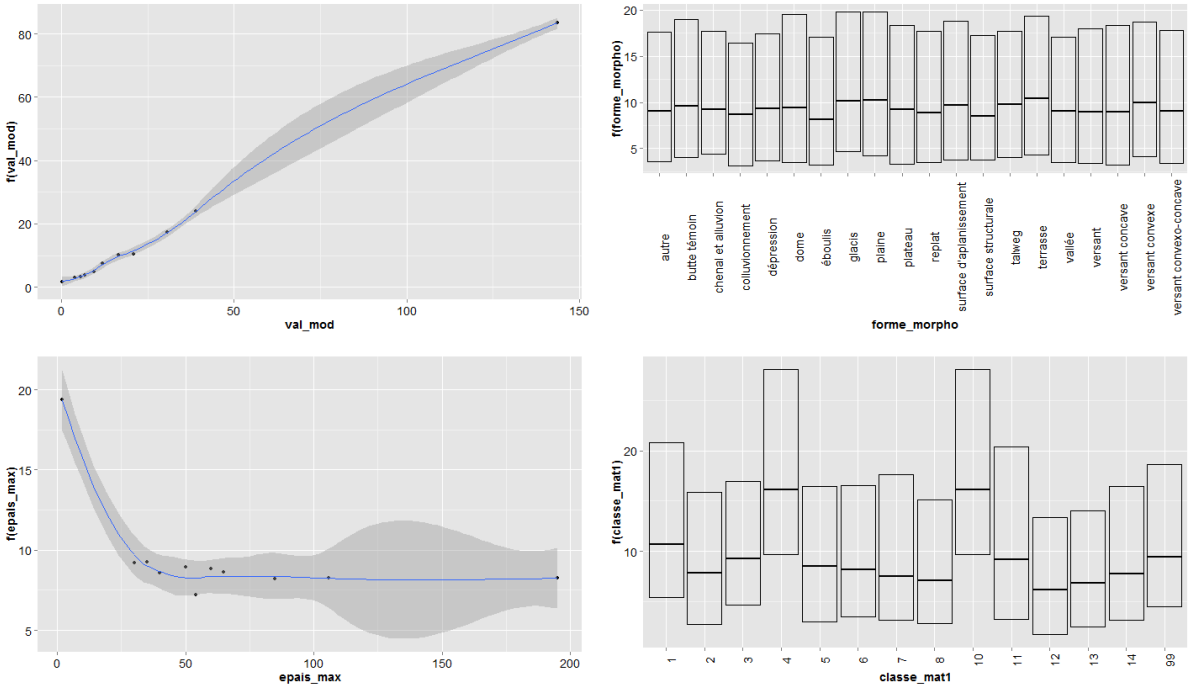


Figure 16: Effet des prédicteurs sur val_min pour le carbone pour la méthode GBM

Pour la variable val_min du carbone, avec la méthode Cubist, les quatre variables les plus importantes sont : val_mod, epais_max, prof_appar_moy et prof_appar_max. On obtient les graphiques suivants :

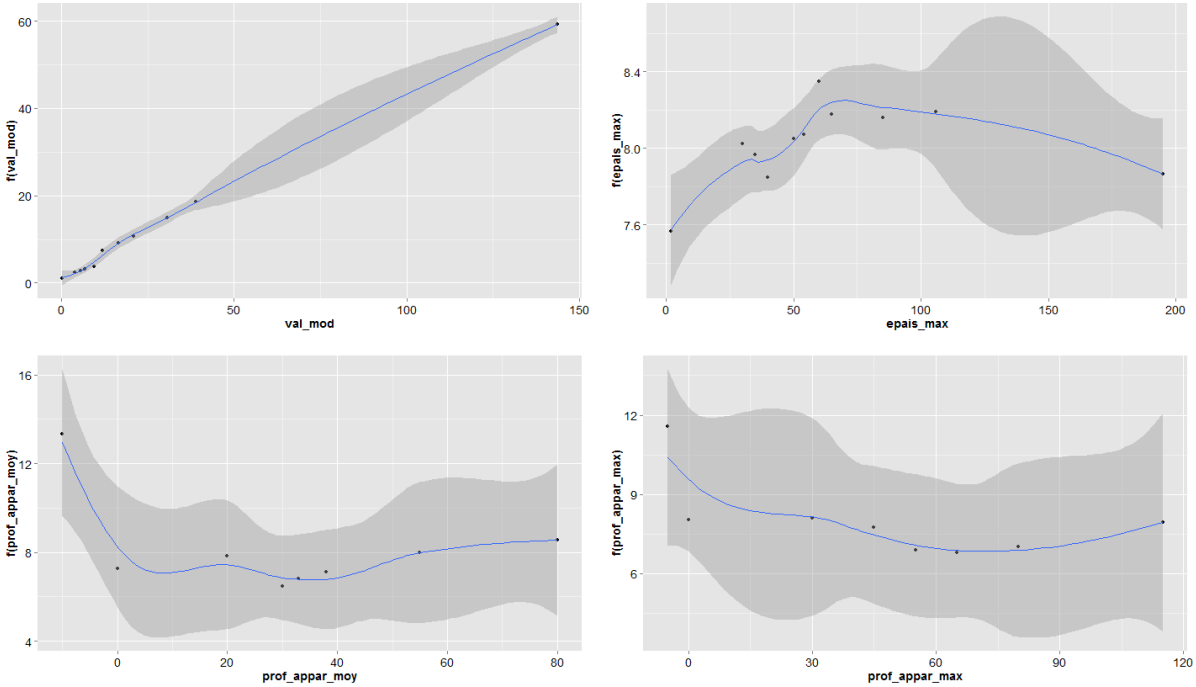


Figure 17: Effet des prédicteurs sur val_min pour le carbone pour la méthode Cubist

Pour ces trois figures, on observe que pour val_mod, plus sa valeur augmente, plus la valeur de val_min augmente, ce qui est cohérant avec les résultats précédents.

Pour la variable étendue du pH, avec la méthode Random Forest, les quatre variables les plus importantes sont : val_mod, forme_morpho, classe_mat1 et epais_max. On obtient les graphiques suivants :

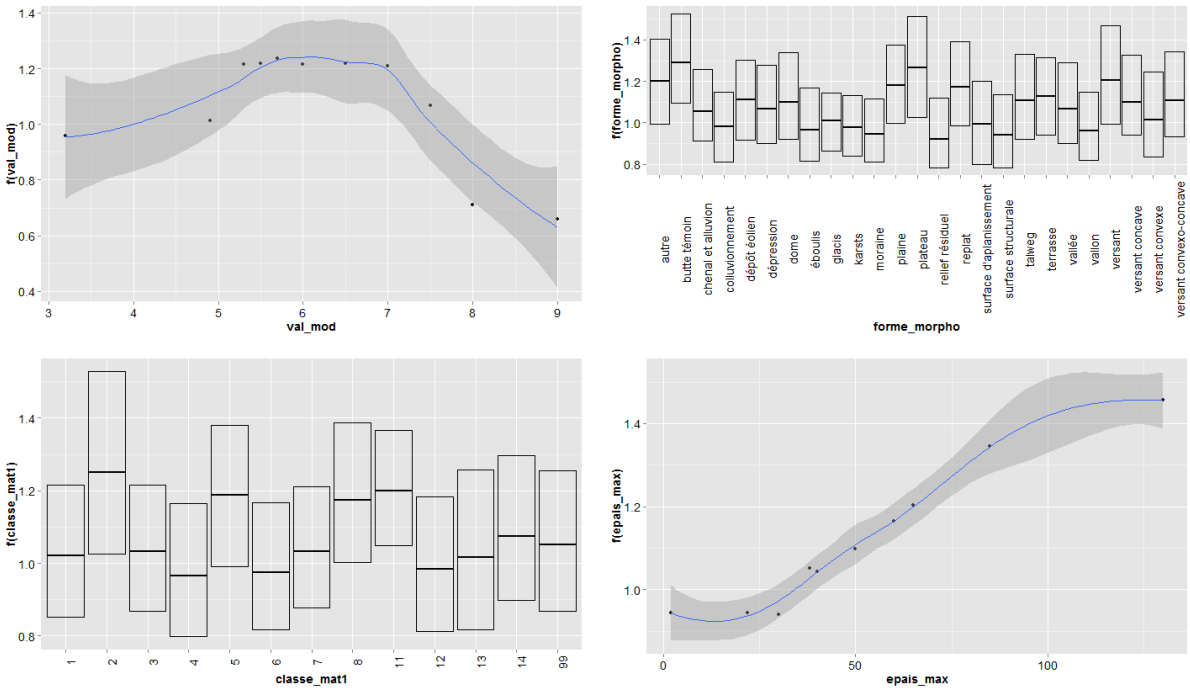


Figure 18: Effet des prédicteurs sur étendue pour le pH pour la méthode Random Forest

Pour la variable étendue du pH, avec la méthode GBM, les quatre variables les plus importantes sont : val_mod, forme_morpho, alt_min et classe_mat1. On obtient les graphiques suivants :

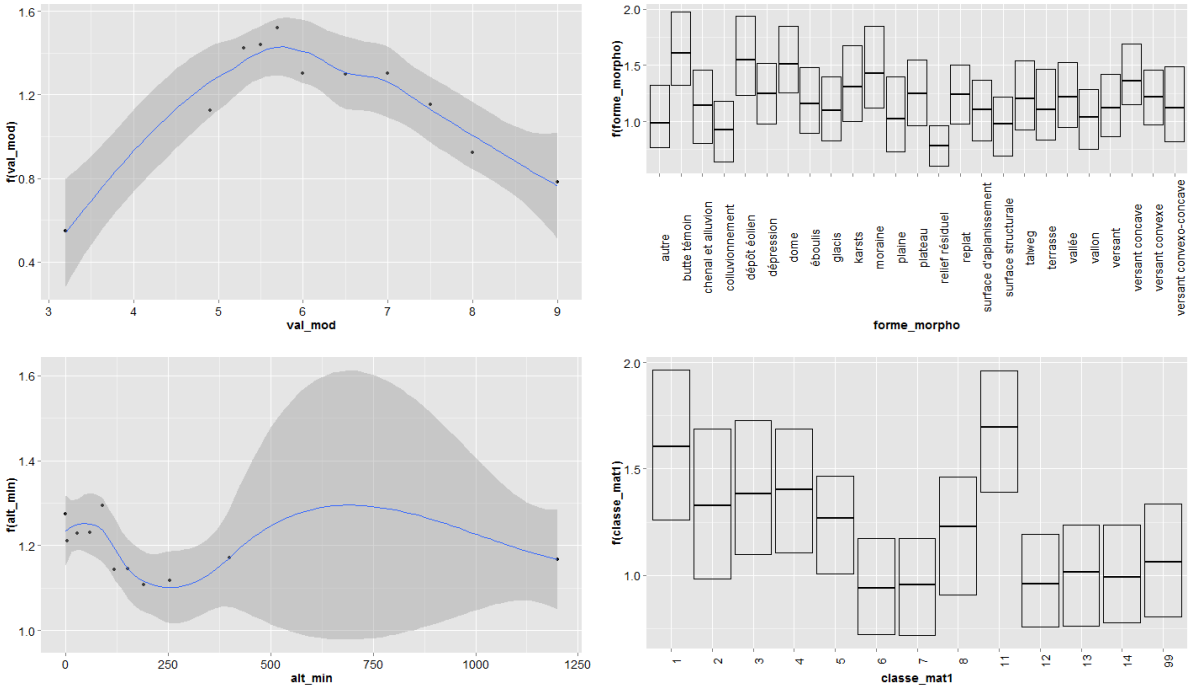


Figure 19: Effet des prédicteurs sur étendue pour le pH pour la méthode GBM

Pour la variable étendue du pH, avec la méthode Cubist, les quatre variables les plus importantes sont : val_mod, alt_min, epais_max et classe_mat1. On obtient les graphiques suivants :

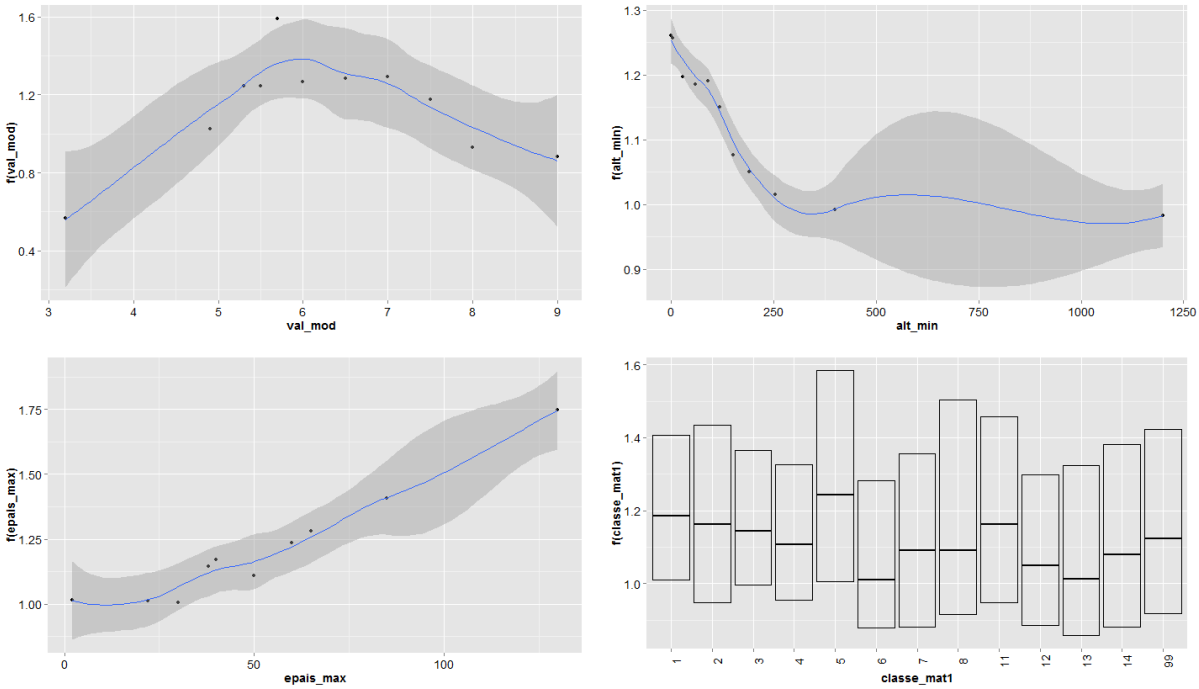


Figure 20: Effet des prédicteurs sur étendue pour le pH pour la méthode Cubist

On constate que pour l'étendue du pH, lorsque la valeur modale augmente, l'étendue augmente également puis diminue dès que la valeur modale est égale à 6 environ.

4. Discussion

Nous discutons ici successivement de la convergence et de la divergence des modèles, de leur interprétation et de leur performance et intérêt.

4.1. Convergence et divergence des modèles

Les modèles donnent des résultats qui sont globalement convergents. En particulier le fait que la valeur modale soit toujours la première en termes d'importance est cohérent et a priori logique. En revanche on constate quelques différences dans l'importance des variables suivantes. Par exemple, pour l'étendue du pH, un effet des matériaux parentaux était attendu, ce qui fut le cas. Mais selon les modèles, on constate que cet effet peut être différent pour un matériau identique. De la même façon, pour le carbone, la classe 12 de matériau parental, qui correspond à des substrats organiques, est logiquement fortement influente avec le modèle Random Forest alors qu'elle n'apparaît pas avec les deux autres.

4.2. Interprétations

Pour le carbone, la très forte dépendance des valeurs minimales et maximales aux valeurs modales était attendue. Il a été montré de nombreuses fois que ce paramètre suivait une loi proche d'une distribution log-normale. Le fait que les valeurs de Carbone soient liées à la profondeur est également logique. En effet la matière organique s'accumule préférentiellement en surface et c'est à ce niveau qu'elle présente le plus de variabilité ; en profondeur elle présente des niveaux plus faibles et en général plus homogènes.
Pour le pH, l'effet du matériau est logique. Les sols sur matériaux calcaires ont généralement de pH plus élevés et dans une gamme de variation réduite. La relation entre la valeur modale et l'étendue du pH est également très intéressante et logique en termes de science du sol. En milieu basique (calcaire ou sodique) le pH est élevé mais ses variations sont très faibles car il est toujours tamponné par le cation Ca++. Inversement, en milieu très acide le pH est très bas, mais ne varie pas non plus car il est contrôlé par l'ion Al+++. C'est donc dans les gammes de pH « neutres » que l'on observe le plus de variations car dans ces milieux, le pH n'est pas stabilisé naturellement et dépend essentiellement des apports agricoles (engrais, chaulage…) qui sont très fortement variables dans le temps et dans l'espace.

4.3. Performance et intérêt

Globalement, les modèles montrent de bonnes performances, même si les modèles sur le carbone semblent meilleurs que ceux sur le pH. Ceci était attendu, car le pH présente des variations saisonnières qui peuvent parfois atteindre des deltas de valeurs de 1, alors

que la variable carbone est temporellement plus stable. Comme les prélèvements et les analyses ont lieu durant toute l'année, il y a donc un « bruit » naturel sur la variable pH. Les résultats obtenus sont en général interprétables au plan naturaliste, ce qui est un gage de qualité pour l'utilisateur.

L'intérêt principal de ce type d'approche est de pouvoir dériver des valeurs par défaut de ces indicateurs de dispersion lorsqu'elles sont manquantes dans les bases de données. On peut toutefois penser que les types de modèles que nous avons utilisés pourraient parfois conduire à un « sur-ajustement » qui donne une fausse idée de leur performance. Pour vérifier cela il faudrait disposer d'une validation externe entièrement indépendante.

Deux solutions pourraient être envisageables :

- Une validation purement externe par un échantillonnage indépendant, qui pourrait par exemple consister en un échantillonnage aléatoire stratifié des strates où l'on ne connaît que la valeur modale et les variables environnementales et à un retour sur le terrain pour y réaliser de nombreuses analyses afin d'estimer les indicateurs de dispersion et les comparer aux valeurs prédites par les modèles. Cette solution pourrait toutefois être d'un coût prohibitif.
- Une autre solution pourrait être le recours à des avis d'experts (par exemple les auteurs des bases de données ou bien des pédologues travaillant dans les milieux similaires) en leur soumettant les valeurs prédites et en leur demandant de les expertiser.

Conclusion

Ce stage m'a permis d'apprendre de nouvelles méthodes en fouille de données, comme GBM et Cubist. J'ai pu découvrir de nouveaux outils sur le logiciel R avec notamment le package *ggplot2*. J'ai également approfondi mes connaissances en programmation R grâce à la création de trois fonctions.

Le stage a apporté à l'organisme d'accueil des méthodes pour l'estimation des indicateurs de dispersion (valeur minimale et maximale ou bien l'étendue) des variables présentes dans les sols telles que le carbone, le pH mais aussi le sable, l'argile, etc. Ces indicateurs de dispersion, souvent absents dans les bases de données sont pourtant très utiles aux analyses de sensibilité des modèles globaux qui traitent de grands enjeux planétaires tels que la réduction de l'effet de serre mais également la production agricole et la sécurité alimentaire.

Les perspectives qui en découlent sont d'une part la possibilité d'application des méthodes utilisées sur d'autres variables. D'autre part, la présentation des travaux effectués lors de la conférence internationale GlobalSoilMap d'octobre, accompagnée d'un article.

Bibliographie

- L. Breiman (2001). *Random Forests*. Machine Learning 45(1). 5-32

- J.H. Friedman (2002). *Stochastic Gradient Boosting*. Computational Statistics and Data Analysis 38(4). 367-378

- Quinlan (1992). *Learning with continuous classes*. Proceedings of the 5th Australian Joint Conference On Artificial Intelligence. 343-348

- Quinlan (1993). *Combining instance-based and model-based learning*. Proceeding of Tenth International Conference on Machine Learning. 236-243

- R Core Team (2013). *R : A language and environment for statistical computing*. R foundation for Statistical Computing, Vienna, Austria. URL http:/www.R-project.org/

- S. Yoo (2012). *Variable selection for hedonic model using machine learning approaches: A case study in Onondaga County, NY*. Landscape and Urban Planning. 293-306

- **Sites internet**:

  - ***INRA:***
    http://www.inra.fr/
    http://www.val-de-loire.inra.fr/

  - ***Logiciel R:***
    http://www.duclert.org
    http://docs.ggplot2.org/current/
    http://www.cookbook-r.com/Graphs/
    http://stackoverflow.com/questions/13788563/r-multiplot-not-working-because-of-error-in-usemethodgetmodelinfo-model
    http://cran.r-project.org/web/packages/

  - ***Fouille de données :***
    http://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/tp_ozone_agreg.pdf
    http://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-agreg.pdf
    http://www.stanford.edu/~stephsus/R-randomforest-guide.pdf
    http://stats.stackexchange.com/questions/12605/measures-of-variable-importance-in-random-forests
    http://stats.stackexchange.com/questions/21152/obtaining-knowledge-from-a-random-forest

- **Cours de Master Statistiques et Probabilités Appliquées :**
  - CHAUVEAU Didier, Statistiques descriptives, M1 2e semestre
  - EMILION Richard, Aide à la décision et Data Mining, M2 1er semestre

Annexes

Annexe 1 : Descriptions des variables des bases de données IGCS

- **no_etude :** numéro de l'étude dans la base de données
- **echelle :** échelle de restitution de l'étude considérée
- **no_ucs :** numéro de l'Unité Cartographique de Sol (UCS)
- **alt_max :** altitude maximale dans l'UCS
- **alt_mod :** altitude modale dans l'UCS
- **alt_min :** altitude minimale dans l'UCS
- **surf_unit :** superficie de l'UCS
- **no_uts :** numéro de l'Unité Typologique de Sol (UTS)
- **nom_mat :** nom des roches mères et des substrats dans l'ordre de leur apparition dans le solum depuis la surface du sol
- **classe_mat1 :** catégorie à laquelle appartient le 1er matériau parental
- **classe_mat2 :** catégorie à laquelle appartient le 2e matériau parental
- **classe_mat3 :** catégorie à laquelle appartient le 3e matériau parental
- **cpcs_nom :** nom du type de sol
- **rp_95_nom :** nom complet du sol selon le Référentiel Pédologique Français de 1995
- **rp_95_ger :** grand ensemble de référence selon le Référentiel Pédologique Français de 1995
- **rp_95_adj1 :** 1er adjectif du nom du sol selon le Référentiel Pédologique Français de 1995
- **rp_95_adj2 :** 2nd adjectif du nom du sol selon le Référentiel Pédologique Français de 1995
- **rp_2008_nom :** nom complet du sol selon le Référentiel Pédologique Français de 2008
- **rp_2008_ger :** grand ensemble de référence selon le Référentiel Pédologique Français de 2008
- **rp_2008_adj1 :** 1er adjectif du nom du sol selon le Référentiel Pédologique Français de 2008
- **rp_2008_adj2 :** 2nd adjectif du nom du sol selon le Référentiel Pédologique Français de 2008
- **no_strate :** numéro de la strate
- **forme_veg :** Type de formation végétale dominant présent dans l'UTS
- **forme_veg1 :** Type de formation végétale dominant présent dans l'UTS
- **forme_veg2 :** Type de formation végétale secondaire présent dans l'UTS
- **forme_veg3 :** Type de formation végétale mineur présent dans l'UTS
- **pourcent :** pourcentage de l'UTS dans l'UCS à laquelle elle appartient
- **forme_morpho :** forme morphologique de l'UTS
- **pente_max :** valeur maximale de la pente dans l'UTS
- **pente_moy :** valeur moyenne de la pente dans l'UTS
- **pente_min :** valeur minimale de la pente dans l'UTS
- **nom_ss_classif :** nom de la strate sans classification
- **prof_appar_min :** profondeur minimale d'apparition de la strate dans l'UTS

- **prof_appar_moy :** profondeur moyenne d'apparition de la strate dans l'UTS
- **prof_appar_max :** profondeur maximale d'apparition de la strate dans l'UTS
- **epais_min :** épaisseur minimale de la strate dans l'UTS
- **epais_moy :** épaisseur moyenne de la strate dans l'UTS
- **epais_max :** épaisseur maximale de la strate dans l'UTS
- **id_strate_quant :** identifiant de l'enregistrement
- **nom_var :** nom de la variable pour la strate considérée
- **val_min :** valeur minimale prise par la variable quantitative définie dans le champ nom_var
- **val_mod :** valeur modale (la plus fréquente) prise par la variable quantitative définie dans le champ nom_var
- **val_max :** valeur maximale prise par la variable quantitative définie dans le champ nom_var
- **unite :** unité correspondant au résultat de la détermination saisie
- **info_var :** information sur la manière dont est renseignée la variable
- **id_methode_physique :** numéro identifiant la méthode physique de la valeur quantitative
- **id_methode :** numéro identifiant la méthode de détermination de la valeur quantitative
- **no_methode :** numéro de la méthode
- **nom_methode :** nom de la méthode

Annexe 2 : Code R

- Remaniement de la base de données IGCS pour le carbone :

```
# Chargement de la base de données du carbone
dataC <-
read.table("D:/murciano/Strates_GSM/resultats_requetes/strates_GSM_carbone.
csv",sep=";",header=TRUE,na.string = "")
bc <- dataC

# Suppression et création de covariables
bc <- bc[,-c(5,11,12,13,16:21,23:26,29:36,44,48,50,52:55)]
bc <- cbind(bc, (bc[,4]+bc[,5])/2)
names(bc)[27] <- "alt_moy"
bc <- cbind(bc, bc[,4]-bc[,5])
names(bc)[28] <- "alt_delta"
bc <- cbind(bc, bc[,19]+(bc[,16]/2))
names(bc)[29] <- "profondeur"
bc <- cbind(bc, (bc[,13]*bc[,6])/100)
names(bc)[30] <- "surf_uts"
bc <- cbind(bc, bc[,24]-bc[,22])
names(bc)[31] <- "etendue"
bc <- cbind(bc, (bc[,24]-bc[,22])/bc[,23])
names(bc)[32] <- "etendue_norm"

# J'enleve nom_mat et rp_95_nom
bc <- bc[,-c(8,10)]

# Enregistrement de la nouvelle base de données
write.table(bc,"D:/murciano/base/bc.csv",sep=";",na="",row.names=FALSE)
bc <- read.table("D:/murciano/base/bc.csv",sep=";",header=TRUE,na.string =
"")
bc[,8] <- as.factor(bc[,8])
bc[,12] <- as.factor(bc[,12])

# modification des classes de la variable forme_morpho
levels(bc[,12])[1:8] <- éboulis"
levels(bc[,12])[2:3] <- "autre"
levels(bc[,12])[3] <- "colluvionnement"
levels(bc[,12])[4:6] <- "glacis"
levels(bc[,12])[5:7] <- "autre"
levels(bc[,12])[5] <- "butte témoin"
levels(bc[,12])[6:9] <- "surface d'aplanissement"
levels(bc[,12])[7] <- "surface structurale"
levels(bc[,12])[8:9] <- "autre"
levels(bc[,12])[8:14] <- "chenal et alluvion"
levels(bc[,12])[9:10] <- "terrasse"
levels(bc[,12])[10:26] <- "autre"
levels(bc[,12])[10] <- "plateau"
levels(bc[,12])[11] <- "plaine"
levels(bc[,12])[12] <- "replat"
levels(bc[,12])[13] <- "autre"
levels(bc[,12])[13] <- "dépression"
levels(bc[,12])[14] <- "vallée
levels(bc[,12])[15] <- "talweg"
levels(bc[,12])[16] <- "dome"
levels(bc[,12])[17] <- "autre"
levels(bc[,12])[17] <- "versant"
```

```
levels(bc[,12])[18] <- "versant concave"
levels(bc[,12])[19] <- "versant convexe"
levels(bc[,12])[20] <- "versant convexo-concave"
levels(bc[,12])[21:25] <- "autre"

# Enregistrement de la nouvelle base
write.table(bc,"D:/murciano/base/bc_fmorpho.csv",sep=";",na="",row.names=FA
LSE)
bcfm <-
read.table("D:/murciano/base/bc_fmorpho.csv",sep=";",header=TRUE,na.string
= "")
bcfm[,8] <- as.factor(bcfm[,8])
```

- Remaniement de la base de données IGCS pour le pH :

```
# Chargement de la base de données du carbone
dataph <-
read.table("D:/murciano/Strates_GSM/resultats_requetes/strates_GSM_ph.csv",
sep=";",header=TRUE,na.string = "")
bph <- dataph

# Suppression et création de covariables
bph <- bph[,-c(5,11,12,13,16:21,23:26,29:36,44,48,50,52:55)]
bph <- cbind(bph, (bph[,4]+bph[,5])/2)
names(bph)[27] <- "alt_moy"
bph <- cbind(bph, bph[,4]-bph[,5])
names(bph)[28] <- "alt_delta"
bph <- cbind(bph, bph[,19]+(bph[,16]/2))
names(bph)[29] <- "profondeur"
bph <- cbind(bph, (bph[,13]*bph[,6])/100)
names(bph)[30] <- "surf_uts"
bph <- cbind(bph, bph[,24]-bph[,22])
names(bph)[31] <- "etendue"
bph <- cbind(bph, (bph[,24]-bph[,22])/bph[,23])
names(bph)[32] <- "etendue_norm"

# J'enleve nom_mat et rp_95_nom
bph <- bph[,-c(8,10)]

# Enregistrement de la nouvelle base de données
write.table(bph,"D:/murciano/base/bph.csv",sep=";",na="",row.names=FALSE)
bph <- read.table("D:/murciano/base/bph.csv",sep=";",header=TRUE,na.string
= "")
bph[,8] <- as.factor(bph[,8])
bph[,12] <- as.factor(bph[,12])

# modification des classes de la variable forme_morpho
levels(bph[,12])[1:11] <- "éboulis"
levels(bph[,12])[2:3] <- "autre"
levels(bph[,12])[3] <- "colluvionnement"
levels(bph[,12])[4] <- "autre"
levels(bph[,12])[4:7] <- "glacis"
levels(bph[,12])[5:9] <- "relief résiduel"
levels(bph[,12])[6] <- "butte témoin"
levels(bph[,12])[7:11] <- "surface d'aplanissement"
levels(bph[,12])[8] <- "surface structurale"
levels(bph[,12])[9] <- "dépôt éolien"
levels(bph[,12])[10:11] <- "autre"
```

```
levels(bph[,12])[10:23] <- "chenal et alluvion"
levels(bph[,12])[11:12] <- "terrasse"
levels(bph[,12])[12] <- "vallon"
levels(bph[,12])[13:23] <- "autre"
levels(bph[,12])[13:21] <- "karsts"
levels(bph[,12])[14:18] <- "autre"
levels(bph[,12])[14:17] <- "moraine"
levels(bph[,12])[15:20] <- "autre"
levels(bph[,12])[15] <- "plateau"
levels(bph[,12])[16] <- "plaine"
levels(bph[,12])[17] <- "replat"
levels(bph[,12])[18] <- "autre"
levels(bph[,12])[18] <- "dépression"
levels(bph[,12])[19] <- "vallée
levels(bph[,12])[20] <- "talweg"
levels(bph[,12])[21] <- "dome"
levels(bph[,12])[22:23] <- "autre"
levels(bph[,12])[22] <- "versant"
levels(bph[,12])[23] <- "versant concave"
levels(bph[,12])[24] <- "versant convexe"
levels(bph[,12])[25] <- "versant convexo-concave"
levels(bph[,12])[26] <- "versant"
levels(bph[,12])[26:28] <- "autre"

# Enregistrement de la nouvelle base
write.table(bph,"D:/murciano/base/bph_fmorpho.csv",sep=";",na="",row.names=
FALSE)
bphfm <-
read.table("D:/murciano/base/bph_fmorpho.csv",sep=";",header=TRUE,na.string
= "")
bphfm[,8] <- as.factor(bphfm[,8])
```

- Satterplots

```
# Scatterplot pour le carbone et le pH
vNames <-
c("val_min","val_max","etendue","etendue_norm","alt_max","alt_min","surf_un
it","classe_mat1","forme_morpho","prof_appar_min","prof_appar_moy","prof_ap
par_max","epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta
","profondeur","surf_uts")
d <- bphfm[vNames][complete.cases(bphfm[vNames]),]
d <- bcfm[vNames][complete.cases(bcfm[vNames]),]
require(psych)
pairs.panels(d[,1:8],smooth=F,lm=T)
pairs.panels(d[,c(1:4,9:12)],smooth=F,lm=T)
pairs.panels(d[,c(1:4,13:16)],smooth=F,lm=T)
pairs.panels(d[,c(1:4,17:20)],smooth=F,lm=T)

# Statistiques générales
summary(d$alt_max)
var(d$alt_max)
```

- Fonction FRodas:

```
FRodas <- function(titre,
                   datacpt,
                   folderSave,
                   model,
```

```
                    transfParam,
                    tuneGrid,
                    trControl,
                    vNames
)

# Description
  # fonction permettant de sélectionner les paramètres optimaux à l'aide de
la fonction train
  # du package caret pour les modèles randomForest, gbm et cubist.


# Input
  # titre: Nom que l'on veut donner au fichier de sauvegarde .RData
  # datacpt:base de donn饰
  # folderSave:Nom du chemin permettant d'accéder au dossier où l'on veut
sauvegarder
  # model:Nom du modèle (rf, gbm, cubist)
  # transfParam : choix de la transormation des données (aucune ou log)
  # tuneGrid : grille contenant les valeurs de réglage possibles
  # trControl : liste de valeurs qui définissent comment la fonction train
agit
  # vNames : Nom des variables avec en premier la variable à expliquer puis
ensuite les variables explicatives

# Output
  # Sortie Modèle : sauvegarde dans le dossier choisi en paramètre d'entrée
du modèle obtenu par train.



{
  require(caret)

  # for the parallel backend
  require(doMC)
  registerDoMC()
  set.seed(1)

  # Nbr de processeurs (ici 4)
  options(cores=4)

  #transformation des données
  if(transfParam=="log"){
    datacpt[vNames[1]] <- log(datacpt[vNames[1]])
  }else{}

  # Si le modèle choisit est rf, on enlève les NA
  if(model=="rf"){
    # Sélection du jeu complet de données
    d <- datacpt[vNames][complete.cases(datacpt[vNames]),]
  }else{d <- datacpt[vNames]}

  # sélection des paramètres optimaux
  namemodel <- paste("cTune_",model,sep="")
  assign(namemodel,train(x = d[,vNames[-1]], y =
d[,vNames[1]],model,tuneGrid = tuneGrid,trControl = trControl,verbose =
F,keep.data = T))

  # sauvegarde des résultats obtenus
```

```
  save(list=namemodel,file=paste(folderSave,titre,".RData",sep=""))

}
# Fin de la fonction
```

- Application de la fonction FRodas

```
source("/home/stats/Violaine/Fonctions/FModel2.R")

### Pour randomForest:
require(randomForest)
### Pour gbm:
require(gbm)
### Pour cubist:
require(Cubist)

## Pour val_min:

vNames <-
c("val_min","alt_max","alt_min","surf_unit","classe_mat1","forme_morpho","p
rof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

cTuneCubistVMin <- FRodas(titre = "cubistValMin",
                       datacpt = bcfm,
                       folderSave = "/home/stats/Violaine/",
                       model = "cubist",
                       transfParam = FALSE,
                       tuneGrid = expand.grid(.committees=
c(1,10,50,100),.neighbors=c(0,1,5,9)),
                       trControl = trainControl(method = "cv"),
                       vNames = vNames)

## Pour val_max:

vNames <-
c("val_max","alt_max","alt_min","surf_unit","classe_mat1","forme_morpho","p
rof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

cTuneCubistVMax <- FRodas(titre = "cubistValMax",
                       datacpt = bcfm,
                       folderSave = "/home/stats/Violaine/",
                       model = "cubist",
                       transfParam = FALSE,
                       tuneGrid = expand.grid(.committees=
c(1,10,50,100),.neighbors=c(0,1,5,9)),
                       trControl = trainControl(method = "cv"),
                       vNames = vNames)

## Pour etendue:

vNames <-
c("etendue","alt_max","alt_min","surf_unit","classe_mat1","forme_morpho","p
rof_appar_min","prof_appar_moy","prof_appar_max",
```

```
"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

cTuneCubistEtendue <- FRodas(titre = "cubistetendue",
                      datacpt = bcfm,
                      folderSave = "/home/stats/Violaine/",
                      model = "cubist",
                      transfParam = FALSE,
                      tuneGrid = expand.grid(.committees=
c(1,10,50,100),.neighbors=c(0,1,5,9)),
                      trControl = trainControl(method = "cv"),
                      vNames = vNames)

## Pour etendue_norm:
mask <- is.na(bcfm[,"etendue_norm"])
bcfm <- bcfm[!mask,]

vNames <-
c("etendue_norm","alt_max","alt_min","surf_unit","classe_mat1","forme_morph
o","prof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

cTuneCubistEtendueNorm <- FRodas(titre = "cubistetendueNorm",
                      datacpt = bcfm,
                      folderSave = "/home/stats/Violaine/",
                      model = "cubist",
                      transfParam = FALSE,
                      tuneGrid = expand.grid(.committees=
c(1,10,50,100),.neighbors=c(0,1,5,9)),
                      trControl = trainControl(method = "cv"),
                      vNames = vNames)

# On a ici le code pour cubist, pour random forest, on remplace model =
"rf", le titre et tuneGrid = expand.grid(.ntrees = c(500,1000,1500),.mtry =
c(3,5,10,15))
# Pour GBM, on remplace model = "gbm", le titre et tuneGrid =
expand.grid(.interaction.depth = c(3,5,9),.n.trees =
c(1000,1500,5000),.shrinkage = 0.1)
# On fait de même pour le pH, en remplaçant la base de données.
```

- Fonction FLocalML

```
FLocalML <- function(titre,
                     datacpt,
                     folderSave,
                     model,
                     transfParam,
                     nbl,
                     prob,
                     cTune,
                     vNames
)



# Description
```

```
  # Fonction permettant de faire du RandomForest, gbm ou cubist avec
validation croisée
  # avec les paramètres des fonctions obtenus par la fonction FRodas.
  # Modification de la version 1. Validation croisée k-fold en utilisant le
paquet cvTools
  # et la fonction cvFolds

# Input
  # titre: Nom que l'on veut donner au fichier de sauvegarde .RData
  # datacpt: base de données
  # folderSave: Nom du chemin permettant d'accéder au dossier où l'on veut
sauvegarder
  # model: Nom du modèle (rf, gbm, cubist)
  # nbl : Nombre d'itérations
  # prob : pourcentage du jeu de données de validation (i.e. 0.2)
  # transfParam : choix de la transformation des données (aucune ou log)
  # cTune : modèle obtenu en sortie de la fonction FRodas
  # vNames : Nom des variables avec en premier la variable à expliquer puis
ensuite les variables explicatives

# Output
  # Sortie Modèle: sauvegarde des modèles obtenus à chaque itération,
  # tableau avec les valeurs prédites pour chaque itérations ainsi que
  # les valeurs observées ainsi que les importances des variables.
{

  #Pour avoir toujours les mêmes bases test/apprentissage (pour mieux
comparer)
  set.seed(1)

  #transformation des données
  if(transfParam=="log"){
    datacpt[vNames[1]] <- log(datacpt[vNames[1]])
  }else{}

  ## Suppression des NA si randomForest utilisé
  #if(model=="rf"){
    # Sélection du jeu complet de données
    d <- datacpt[vNames][complete.cases(datacpt[vNames]),]
  #}else{d <- datacpt[vNames]}

  # Détermination du nombre de block (k-fold)
  nbrK <- nrow(d)/(nrow(d)*prob)

  # création tableau contenant les valeurs prédites pour chaque itération
  restPred <- array(NA, dim = c(nrow(d), round(nbl*nbrK), 1),list(id =
seq(1,nrow(d),1), loop = 1:round(nbl*nbrK), mod = model))
  # tableau contenant les importances des variables pour chaque itération.
  restImpVar <- array(dim=c(length(d)-1, round(nbl*nbrK)))
  rownames(restImpVar) <-names(d[,order(names(d[,c(2:(length(d)))]))+1])

  # Création des blocks
  kfold <- cvFolds(nrow(d),nbrK,type="random",R=nbl)


  for(i in 1:nbl){
    for(b in 1:nbrK){
      print(i*b)
```

```
      baseAp <- d[kfold$subsets[kfold$which!=b,i],]
#kfold$subsets[kfold$which!=b,i] permet d'obtenir les indices des lignes
des groupes autres que b
      baseTest <- d[kfold$subsets[kfold$which==b,i],]

      # Pour randomForest
      if(model=="rf"){
        formula <- as.formula(paste(names(baseAp[vNames[1]]), " ~ ",
paste(names(baseAp[,vNames[-1]]),collapse=" + ")))
        modelML <- randomForest(formula, data = baseAp, ntree=1000,
mtry=cTune$bestTune$.mtry,importance=TRUE)
        #sauvegarde du modèle pour chaque itération
        save(modelML,file=paste(folderSave,titre,i*b,".RData",sep=""))
        # importance des variables
        impML = importance(modelML)
        restImpVar[,i*b]=impML[order(rownames(impML)),1]
        # Prédiction sur la base de validation (20%)
        Valpred <- predict(modelML,baseTest[,vNames[-1]])

      }else if(model=="gbm"){
        formula <- as.formula(paste(names(baseAp[vNames[1]]), " ~ ",
paste(names(baseAp[,vNames[-1]]),collapse=" + ")))
        modelML <- gbm(formula, data = baseAp, distribution="gaussian",
n.trees=cTune$bestTune$.n.trees,shrinkage=cTune$bestTune$.shrinkage,
interaction.depth=cTune$bestTune$.interaction.depth, train.fraction = 1,
n.minobsinnode = 10, cv.folds = 10,keep.data=TRUE, verbose=F)
        #sauvegarde du modèle pour chaque itération
        save(modelML,file=paste(folderSave,titre,i*b,".RData",sep=""))
        #importance des variables
        impML = summary(modelML)
        restImpVar[,i*b]=impML[order(impML[,1]),2]
        # Prédiction sur la base de validation (20%)
        best.iter <- gbm.perf(modelML,method="cv")
        Valpred <- predict(modelML,baseTest[,vNames[-1]],best.iter)

      }else if(model=="cubist"){
        modelML <- cubist(x=baseAp[,vNames[-1]],y=baseAp[,vNames[1]], data
=
baseAp,committees=cTune$bestTune$.committees ,neighbors=cTune$bestTune$.nei
ghbors)
        #sauvegarde du modèle pour chaque itération
        save(modelML,file=paste(folderSave,titre,i*b,".RData",sep=""))
        #importance relative:
        impML = modelML$usage
        restImpVar[,i*b]=impML[order(impML[,3]),1]
        # Prédiction sur la base de validation (20%)
        Valpred <- predict(modelML,baseTest[,vNames[-
1]],neighbors=cTune$bestTune$.neighbors)

      }else{}

      if(transfParam=="log"){
        Valpred <- exp(Valpred)
      }else{}

      restPred[kfold$subsets[kfold$which==b,i], i*b, model] <- Valpred
    }
  }

  #permet de combiner restPred avec les valeurs observées correspondantes.
```

```
  if(transfParam=="log"){
    restPred <- cbind(as.data.frame(restPred),exp(d[vNames[1]]))
  }else{
    restPred <- cbind(as.data.frame(restPred),d[,vNames[1]])
  }

  return(list(restPred,restImpVar))

}
```

- Application de la fonction FLocalML

```
source("D:/murciano/code R/FModel2.R")

# on charge la base de données:
bcfm <-
read.table("D:/murciano/base/bc_fmorpho.csv",sep=";",header=TRUE,na.string
= "")
bcfm[,8] <- as.factor(bcfm[,8])

#pour ph:
bphfm <-
read.table("D:/murciano/base/bph_fmorpho.csv",sep=";",header=TRUE,na.string
= "")
bphfm[,8] <- as.factor(bphfm[,8])

require(cvTools)
### Pour randomForest:
require(randomForest)
### Pour gbm:
require(gbm)
### Pour cubist:
require(Cubist)

# Suivant que ce soit du pH ou du carbone, du random forest du gbm ou
cubist, on adapte les paramètres.

## Pour val_min:
cTune = load("X:/Violaine/resultats/ph/gbmValMinPh.RData")
cTune #contient "cTune_rf"

vNames <-
c("val_min","alt_max","alt_min","surf_unit","classe_mat1","forme_morpho","p
rof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

gbmVMinPh <- FLocalML(titre = "gbmValMinPh",
                    datacpt = bphfm,
                    folderSave = "D:/murciano/resultats2/",
                    model = "gbm",
                    transfParam = FALSE,
                    nbl = 1,
                    prob = 0.2,
                    cTune = cTune_gbm,
                    vNames = vNames
)
```

```
save(gbmVMinPh,file="D:/murciano/resultats2/modelGbmVMinPh.RData")

## Pour val max:
cTune = load("X:/Violaine/resultats/ph/gbmValMaxPh.RData")
cTune #contient "cTune_rf"

vNames <-
c("val_max","alt_max","alt_min","surf_unit","classe_mat1","forme_morpho","p
rof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

gbmVMaxPh <- FLocalML(titre = "gbmValMaxPh",
                    datacpt = bphfm,
                    folderSave = "D:/murciano/resultats2/",
                    model = "gbm",
                    transfParam = FALSE,
                    nbl = 1,
                    prob = 0.2,
                    cTune = cTune_gbm,
                    vNames = vNames
)

save(gbmVMaxPh,file="D:/murciano/resultats2/modelGbmVMaxPh.RData")

## Pour etendue:
cTune = load("X:/Violaine/resultats/ph/gbmetenduePh.RData")
cTune #contient "cTune_rf"

vNames <-
c("etendue","alt_max","alt_min","surf_unit","classe_mat1","forme_morpho","p
rof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

gbmetenduePh <- FLocalML(titre = "gbmetenduePh",
                       datacpt = bphfm,
                       folderSave = "D:/murciano/resultats2/",
                       model = "gbm",
                       transfParam = FALSE,
                       nbl = 1,
                       prob = 0.2,
                       cTune = cTune_gbm,
                       vNames = vNames
)

save(gbmetenduePh,file="D:/murciano/resultats2/modelGbmEtenduePh.RData")

## Pour etendue_norm:
cTune = load("X:/Violaine/resultats/ph/cubistetendueNormPh.RData")
cTune #contient "cTune_rf"

mask <- is.na(bcfm[,"etendue_norm"])
bcfm <- bcfm[!mask,]
```

```
vNames <-
c("etendue_norm","alt_max","alt_min","surf_unit","classe_mat1","forme_morph
o","prof_appar_min","prof_appar_moy","prof_appar_max",

"epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta","profon
deur","surf_uts")

cubistetendueNormPh <- FLocalML(titre = "cubistetendueNormPh",
                           datacpt = bphfm,
                           folderSave = "D:/murciano/resultats2/",
                           model = "cubist",
                           transfParam = FALSE,
                           nbl = 1,
                           prob = 0.2,
                           cTune = cTune_cubist,
                           vNames = vNames
)

save(cubistetendueNormPh,file="D:/murciano/resultats2/modelCubistEtendueNor
mPh.RData")

##################
#Importance relative par validation croisée(moyenne des importances)
impCv <- apply(cubistetendueNormPh[2][[1]],1,mean)
par(las=2)
barplot(impCv[order(impCv)],horiz=TRUE,cex.names=0.5)

# tableau des valeurs prédites
valPred <-  cubistetendueNormPh[1][[1]]

# graphe une couleur par itération
x1 = valPred[!is.na(valPred[,1]),6]
y1 = valPred[!is.na(valPred[,1]),1]
x2 = valPred[!is.na(valPred[,2]),6]
y2 = valPred[!is.na(valPred[,2]),2]
x3 = valPred[!is.na(valPred[,3]),6]
y3 = valPred[!is.na(valPred[,3]),3]
x4 = valPred[!is.na(valPred[,4]),6]
y4 = valPred[!is.na(valPred[,4]),4]
x5 = valPred[!is.na(valPred[,5]),6]
y5 = valPred[!is.na(valPred[,5]),5]

plot(x=c(x1,x2,x3,x4,x5),y=c(y1,y2,y3,y4,y5),col=c("red","blue","black","gr
een","yellow"),xlab="valeurs observ餘",ylab="valeurs pr餘tes")
abline(0,1)


##indicateurs qualité
Rcarre = c()
mpe = c()
rmse = c()
rpd = c()
for (i in 1:5){
x = valPred[!is.na(valPred[,i]),6]
y = valPred[!is.na(valPred[,i]),i]
Rcarre = c(Rcarre,R2(x,y))
mpe = c(mpe,MPE(x,y))
rmse = c(rmse,RMSE(x,y))
rpd = c(rpd,sd(x)/RMSE(x,y))
}
```

```
Rcarre
summary(Rcarre)
mpe
summary(mpe)
rmse
summary(rmse)
rpd
summary(rpd)
```

- Code fonction multiplot (qui permet de représenter plusieurs graphique su une même fenêtre avec le package ggplot2)

```
# Multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot
objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  require(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout),
ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this
subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
```

```
  }
}
```

- Fonction FPlot

```
FPlot <- function(grille,
                  vNames,
                  d,
                  model,
                  nameModel,
                  neighbors = 0)
  # Description
    # Fonction permettant de tracer l'effet des 4 variables les plus
importantes
    # avec le package ggplot2.

  # Input
    # grille: grille contenant les combinaisons des valeurs des 4 variables
les plus importantes
    # vNames : liste contenant le nom des autres variables prédictives
    # d: base de données
    # model : modèle à utiliser pour la prédiction
    # nameModel : nom du modèle utilisé gbm, rf ou cubist
    # neighbors : paramètre à renseigner pour la prédiction si le modèle
utilisé est cubist, 0 par défaut

  # Output
    # Trace sur une même page les graphiques représentant l'effet de
    # variables sur la prédiction d'une certaine variable
{

  require(ggplot2)
  require(gbm)
  require(randomForest)
  require(Cubist)
  source("D:/murciano/code R/code_multiplot.R")


  # On rajoute à la grille les autres variables (vNames) en leur attribuant
la valeur médiane:
  for (v in vNames){
    if (is.factor(d[,v])==TRUE){
      t = table(d[,v])
      ft = t[order(t,decreasing=T)]
      grille[,v] <- d[d[,v]==names(ft[1]),v][1]
    }else{
      grille[,v] <- median(d[,v])
    }
  }

  # Prédiction:
  if (nameModel == "gbm"){
    best.iter <- gbm.perf(model,method="cv")
    pred <- predict(model,grille,best.iter)
  }else if (nameModel == "rf"){
    pred <- predict(model,grille)
  }else if (nameModel == "cubist"){
    pred <- predict(model,grille,neighbors = neighbors)
```

```
  }

  # On ajoute à la grille les valeurs prédites.
  grille$pred <- pred

  # Calcul de la valeur médiane pour les prédictions:
    # 1

    if (is.factor(grille[,1])==TRUE){
      quant1 <-
aggregate(grille$pred,by=list(grille[,1]),function(x){quantile(x,0.25)})

      quant2 <- aggregate(grille$pred,by=list(grille[,1]),median)

      quant3 <-
aggregate(grille$pred,by=list(grille[,1]),function(x){quantile(x,0.75)})

      predVar1 <- cbind(quant1,quant2[,2],quant3[,2])

      colnames(predVar1) <- c("nomvar","q1","med","q3")

      p1 <- ggplot(predVar1,aes(x=nomvar,y=med,ymin=q1,ymax=q3)) +
geom_crossbar()+ theme(axis.text.x  = element_text(angle=90, vjust=1,
size=14, colour="black"),axis.title.x =
element_text(face="bold",colour="black",size = 14),
axis.text.y  = element_text(size=14, colour="black"),
axis.title.y = element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[1]) + ylab(paste("f(",names(grille)[1],")",sep=""))

      }else{
        predVar1 <- aggregate(grille$pred,by=list(grille[,1]),median)
        colnames(predVar1) <- c("nomvar1","pred1")

        p1 <- ggplot(predVar1,aes(nomvar1,pred1)) + geom_point() +
geom_smooth() + theme(axis.text.x  = element_text(size=14, colour="black"),
axis.title.x = element_text(face="bold",colour="black",size = 14),
axis.text.y  = element_text(size=14, colour="black"),
axis.title.y = element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[1]) + ylab(paste("f(",names(grille)[1],")",sep=""))
    }

    # 2

    if (is.factor(grille[,2])==TRUE){
      quant1 <-
aggregate(grille$pred,by=list(grille[,2]),function(x){quantile(x,0.25)})

      quant2 <- aggregate(grille$pred,by=list(grille[,2]),median)

      quant3 <-
aggregate(grille$pred,by=list(grille[,2]),function(x){quantile(x,0.75)})

      predVar2 <- cbind(quant1,quant2[,2],quant3[,2])

      colnames(predVar2) <- c("nomvar","q1","med","q3")

      p2 <- ggplot(predVar2,aes(x=nomvar,y=med,ymin=q1,ymax=q3)) +
geom_crossbar()+ theme(axis.text.x  = element_text(angle=90, vjust=1,
size=14, colour="black"),axis.title.x =
element_text(face="bold",colour="black",size = 14), axis.text.y  =
```

```
element_text(size=14, colour="black"),axis.title.y =
element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[2]) + ylab(paste("f(",names(grille)[2],")",sep=""))

    }else{
      predVar2 <- aggregate(grille$pred,by=list(grille[,2]),median)

      colnames(predVar2) <- c("nomvar2","pred2")

      p2 <- ggplot(predVar2,aes(nomvar2,pred2)) + geom_point() +
geom_smooth() + theme(axis.text.x  = element_text(size=14, colour="black"),
axis.title.x = element_text(face="bold",colour="black",size = 14),
axis.text.y  = element_text(size=14, colour="black"), axis.title.y =
element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[2]) + ylab(paste("f(",names(grille)[2],")",sep=""))
    }

    # 3

    if (is.factor(grille[,3])==TRUE){
      quant1 <-
aggregate(grille$pred,by=list(grille[,3]),function(x){quantile(x,0.25)})

      quant2 <- aggregate(grille$pred,by=list(grille[,3]),median)

      quant3 <-
aggregate(grille$pred,by=list(grille[,3]),function(x){quantile(x,0.75)})

      predVar3 <- cbind(quant1,quant2[,2],quant3[,2])

      colnames(predVar3) <- c("nomvar","q1","med","q3")

      p3 <- ggplot(predVar3,aes(x=nomvar,y=med,ymin=q1,ymax=q3)) +
geom_crossbar()+ theme(axis.text.x  = element_text(angle=90, vjust=1,
size=14, colour="black"), axis.title.x =
element_text(face="bold",colour="black",size = 14), axis.text.y  =
element_text(size=14, colour="black"), axis.title.y =
element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[3]) + ylab(paste("f(",names(grille)[3],")",sep=""))

    }else{
      predVar3 <- aggregate(grille$pred,by=list(grille[,3]),median)

      colnames(predVar3) <- c("nomvar3","pred3")

      p3 <- ggplot(predVar3,aes(nomvar3,pred3)) + geom_point() +
geom_smooth() + theme(axis.text.x  = element_text(size=14, colour="black"),
axis.title.x = element_text(face="bold",colour="black",size = 14),
axis.text.y  = element_text(size=14, colour="black"),
axis.title.y = element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[3]) + ylab(paste("f(",names(grille)[3],")",sep=""))
    }

    # 4

    if (is.factor(grille[,4])==TRUE){
      quant1 <-
aggregate(grille$pred,by=list(grille[,4]),function(x){quantile(x,0.25)})

      quant2 <- aggregate(grille$pred,by=list(grille[,4]),median)
```

```
      quant3 <-
aggregate(grille$pred,by=list(grille[,4]),function(x){quantile(x,0.75)})

      predVar4 <- cbind(quant1,quant2[,2],quant3[,2])

      colnames(predVar4) <- c("nomvar","q1","med","q3")

      p4 <- ggplot(predVar4,aes(x=nomvar,y=med,ymin=q1,ymax=q3)) +
geom_crossbar()+ theme(axis.text.x  = element_text(angle=90, vjust=1,
size=14, colour="black"), axis.title.x =
element_text(face="bold",colour="black",size = 14),
axis.text.y  = element_text(size=14, colour="black"),
axis.title.y = element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[4]) + ylab(paste("f(",names(grille)[4],")",sep=""))

    }else{
      predVar4 <- aggregate(grille$pred,by=list(grille[,4]),median)

      colnames(predVar4) <- c("nomvar4","pred4")

      p4 <- ggplot(predVar4,aes(nomvar4,pred4)) + geom_point() +
geom_smooth() + theme(axis.text.x  = element_text(size=14, colour="black"),
axis.title.x = element_text(face="bold",colour="black",size = 14),
axis.text.y  = element_text(size=14, colour="black"),
axis.title.y = element_text(face="bold",colour="black",size = 14)) +
xlab(names(grille)[4]) + ylab(paste("f(",names(grille)[4],")",sep=""))
    }

  multiplot(p1, p3, p2, p4, cols=2)

}
```

- Application de la fonction FPlot

```
require(randomForest)
require(gbm)
require(Cubist)
require(ggplot2)
source("D:/murciano/code R/code_multiplot.R")

#Carbone
bcfm <-
read.table("D:/murciano/base/bc_fmorpho.csv",sep=";",header=TRUE,na.string
= "")
bcfm[,8] <- as.factor(bcfm[,8])

vNames <-
c("val_min","val_max","etendue","etendue_norm","alt_max","alt_min","surf_un
it","classe_mat1","forme_morpho","prof_appar_min","prof_appar_moy","prof_ap
par_max","epais_min","epais_moy","epais_max","val_mod","alt_moy","alt_delta
","profondeur","surf_uts")

#chargement de la base de données:
d <- bcfm[vNames][complete.cases(bcfm[vNames]),]

source("D:/murciano/code R/FPlot.R")

## Pour val_min, rf:
```

```
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]),
                      prof_appar_moy =
quantile(d[,"prof_appar_moy"],c(seq(0,0.9,0.1),0.98)),
                      prof_appar_max =
quantile(d[,"prof_appar_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","forme_morpho","prof_appar_min",
"epais_min","epais_moy","epais_max","alt_moy","alt_delta","profondeur",
"surf_uts")

model = load("D:/murciano/resultats2/carbone/rf/val_min/rfValMin1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour val_min, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta",
"profondeur", "surf_uts")

model = load("D:/murciano/resultats2/carbone/gbm/val_min/gbmValMin1.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"gbm")

## Pour val_min, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
                      prof_appar_moy =
quantile(d[,"prof_appar_moy"],c(seq(0,0.9,0.1),0.98)),
                      prof_appar_max =
quantile(d[,"prof_appar_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <- c("alt_max","alt_min","surf_unit","classe_mat1","forme_morpho",
"prof_appar_min","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model =
load("D:/murciano/resultats2/carbone/cubist/val_min/cubistValMin1.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"cubist",neighbors = 5)

## Pour val_max, rf:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
```

```
                        epais_min = quantile(d[,"epais_min"],
c(seq(0,0.9,0.1),0.98)),
                        classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","alt_min","surf_unit","forme_morpho","prof_appar_min",
"prof_appar_moy","prof_appar_max","epais_moy","alt_moy","alt_delta",
"profondeur", "surf_uts")

model = load("D:/murciano/resultats2/carbone/rf/val_max/rfValMax1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour val_max, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                        classe_mat1 = levels(d[,"classe_mat1"]),
                        alt_max = quantile(d[,"alt_max"],
c(seq(0,0.9,0.1),0.98)),
                        forme_morpho = levels(d[,"forme_morpho"]))

vNames1 <-
c("alt_min","surf_unit","prof_appar_min","prof_appar_moy","prof_appar_max",
"epais_min","epais_moy","epais_max","alt_moy","alt_delta","profondeur",
"surf_uts")

model = load("D:/murciano/resultats2/carbone/gbm/val_max/gbmValMax1.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"gbm")

## Pour val_max, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                        profondeur = quantile(d[,"profondeur"],
c(seq(0,0.9,0.1),0.98)),
                        epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
                        prof_appar_moy =
quantile(d[,"prof_appar_moy"],c(seq(0,0.9,0.1),0.98)))

vNames1 <- c("alt_max","alt_min","surf_unit","classe_mat1","forme_morpho",
"prof_appar_min","prof_appar_max","epais_min","epais_moy","alt_moy",
"alt_delta","surf_uts")

model =
load("D:/murciano/resultats2/carbone/cubist/val_max/cubistValMax1.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"cubist",neighbors = 5)

## Pour etendue, rf:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                        epais_min = quantile(d[,"epais_min"],
c(seq(0,0.9,0.1),0.98)),
                        classe_mat1 = levels(d[,"classe_mat1"]),
                        forme_morpho = levels(d[,"forme_morpho"]))
```

```
vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_moy","epais_max","alt_moy","alt_delta","profondeur"
,"surf_uts")

model = load("D:/murciano/resultats2/carbone/rf/etendue/rfetendue1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour etendue, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model =
load("D:/murciano/resultats2/carbone/gbm/etendue/gbmetendue1.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"gbm")

## Pour etendue, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
                      prof_appar_max =
quantile(d[,"prof_appar_max"],c(seq(0,0.9,0.1),0.98)),
                      profondeur =
quantile(d[,"profondeur"],c(seq(0,0.9,0.1),0.98)))

vNames1 <- c("alt_max","alt_min","surf_unit","classe_mat1","forme_morpho",
"prof_appar_min","prof_appar_moy","epais_min","epais_moy","alt_moy",
"alt_delta","surf_uts")

model =
load("D:/murciano/resultats2/carbone/cubist/etendue/cubistetendue1.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"cubist",neighbors = 5)

## Pour etendue_norm, rf:
grille <- expand.grid(classe_mat1 = levels(d[,"classe_mat1"]),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      epais_min = quantile(d[,"epais_min"],
c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_moy","epais_max","alt_moy","alt_delta","profondeur"
,"surf_uts")
```

```
model =
load("D:/murciano/resultats2/carbone/rf/etendue_norm/rfetendueNorm1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour etendue_norm, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      classe_mat1 = levels(d[,"classe_mat1"]),
                      profondeur = quantile(d[,"profondeur"],
c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","epais_max","alt_moy","alt_delta",
"surf_uts")

model =
load("D:/murciano/resultats2/carbone/gbm/etendue_norm/gbmetendueNorm1.RData
")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"gbm")

## Pour etendue_norm, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"],
c(seq(0,0.9,0.1),0.98)),
                      epais_min = quantile(d[,"epais_min"],
c(seq(0,0.9,0.1),0.98)),
                      epais_max = quantile(d[,"epais_max"],
c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","alt_min","surf_unit","forme_morpho","prof_appar_min",
"prof_appar_moy","prof_appar_max","epais_moy","alt_moy","alt_delta",
"profondeur","surf_uts")

model =
load("D:/murciano/resultats2/carbone/cubist/etendue_norm/cubistetendueNorm1
.RData")

FPlot(grille = grille,vNames = vNames1,d = d,model = modelML,nameModel =
"cubist",neighbors = 1)

##########################################################################
##########################################################################
# ph:
bphfm <-
read.table("D:/murciano/base/bph_fmorpho.csv",sep=";",header=TRUE,na.string
= "")
bphfm[,8] <- as.factor(bphfm[,8])

vNames <-
c("val_min","val_max","etendue","etendue_norm","alt_max","alt_min",
"surf_unit","classe_mat1","forme_morpho","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","epais_max","val_mod","alt_moy",
"alt_delta","profondeur","surf_uts")
```

```
#chargement de la base de données:
d <- bphfm[vNames][complete.cases(bphfm[vNames]),]

source("D:/murciano/code R/FPlot.R")

## Pour val_min, rf:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      classe_mat1 = levels(d[,"classe_mat1"]),
                      epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model = load("D:/murciano/resultats2/ph/rf/val_min/rfValMinPh1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour val_min, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      classe_mat1 = levels(d[,"classe_mat1"]),
                      epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model = load("D:/murciano/resultats2/ph/gbm/val_min/gbmValMinPh1.RData")

FPlot(grille,vNames1,d,modelML,"gbm")

## Pour val_min, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      alt_moy =
quantile(d[,"alt_moy"],c(seq(0,0.9,0.1),0.98)),
                      epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","alt_min","surf_unit","forme_morpho","prof_appar_min",
"prof_appar_moy","prof_appar_max","epais_min","epais_moy","alt_delta",
"profondeur","surf_uts")

model =
load("D:/murciano/resultats2/ph/cubist/val_min/cubistValMinPh1.RData")

FPlot(grille,vNames1,d,modelML,"cubist",neighbors = 5)

## Pour val_max, rf:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      classe_mat1 = levels(d[,"classe_mat1"]),
```

```
                        epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model = load("D:/murciano/resultats2/ph/rf/val_max/rfValMaxPh1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour val_max, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                        forme_morpho = levels(d[,"forme_morpho"]),
                        classe_mat1 = levels(d[,"classe_mat1"]),
                        alt_min =
quantile(d[,"alt_min"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","surf_unit","prof_appar_min","prof_appar_moy","prof_appar_max",
"epais_min","epais_moy","epais_max","alt_moy","alt_delta","profondeur",
"surf_uts")

model = load("D:/murciano/resultats2/ph/gbm/val_max/gbmValMaxPh1.RData")

FPlot(grille,vNames1,d,modelML,"gbm")

## Pour val_max, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                        alt_min =
quantile(d[,"alt_min"],c(seq(0,0.9,0.1),0.98)),
                        classe_mat1 = levels(d[,"classe_mat1"]),
                        epais_min =
quantile(d[,"epais_min"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","surf_unit","forme_morpho","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_moy","epais_max","alt_moy","alt_delta","profondeur"
,"surf_uts")

model =
load("D:/murciano/resultats2/ph/cubist/val_max/cubistValMaxPh1.RData")

FPlot(grille,vNames1,d,modelML,"cubist",neighbors = 5)

## Pour etendue, rf:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                        forme_morpho = levels(d[,"forme_morpho"]),
                        classe_mat1 = levels(d[,"classe_mat1"]),
                        epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model = load("D:/murciano/resultats2/ph/rf/etendue/rfetenduePh1.RData")
```

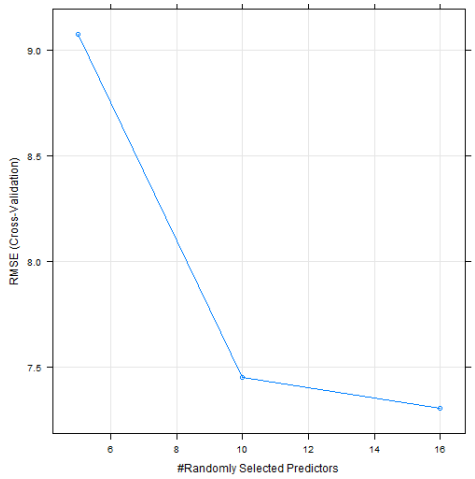
```
FPlot(grille,vNames1,d,modelML,"rf")

## Pour etendue, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      alt_min =
quantile(d[,"alt_min"],c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","surf_unit","prof_appar_min","prof_appar_moy","prof_appar_max",
"epais_min","epais_moy","epais_max","alt_moy","alt_delta","profondeur",
"surf_uts")

model = load("D:/murciano/resultats2/ph/gbm/etendue/gbmetenduePh1.RData")

FPlot(grille,vNames1,d,modelML,"gbm")

## Pour etendue, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      alt_min =
quantile(d[,"alt_min"],c(seq(0,0.9,0.1),0.98)),
                      epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","surf_unit","forme_morpho","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model =
load("D:/murciano/resultats2/ph/cubist/etendue/cubistetenduePh1.RData")

FPlot(grille,vNames1,d,modelML,"cubist",neighbors = 5)

## Pour etendue_norm, rf:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      classe_mat1 = levels(d[,"classe_mat1"]),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)))

vNames1 <-
c("alt_max","alt_min","surf_unit","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model =
load("D:/murciano/resultats2/ph/rf/etendue_norm/rfetendueNormPh1.RData")

FPlot(grille,vNames1,d,modelML,"rf")

## Pour etendue_norm, gbm:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      alt_min =
quantile(d[,"alt_min"],c(seq(0,0.9,0.1),0.98)),
                      forme_morpho = levels(d[,"forme_morpho"]),
                      classe_mat1 = levels(d[,"classe_mat1"]))
```

```
vNames1 <-
c("alt_max","surf_unit","prof_appar_min","prof_appar_moy","prof_appar_max",
"epais_min","epais_moy","epais_max","alt_moy","alt_delta","profondeur",
"surf_uts")

model =
load("D:/murciano/resultats2/ph/gbm/etendue_norm/gbmetendueNormPh1.RData")

FPlot(grille,vNames1,d,modelML,"gbm")

## Pour etendue_norm, cubist:
grille <- expand.grid(val_mod = quantile(d[,"val_mod"], seq(0,1,0.1)),
                      alt_min =
quantile(d[,"alt_min"],c(seq(0,0.9,0.1),0.98)),
                      epais_max =
quantile(d[,"epais_max"],c(seq(0,0.9,0.1),0.98)),
                      classe_mat1 = levels(d[,"classe_mat1"]))

vNames1 <-
c("alt_max","surf_unit","forme_morpho","prof_appar_min","prof_appar_moy",
"prof_appar_max","epais_min","epais_moy","alt_moy","alt_delta","profondeur"
,"surf_uts")

model =
load("D:/murciano/resultats2/ph/cubist/etendue_norm/cubistetendueNormPh1.RD
ata")

FPlot(grille,vNames1,d,modelML,"cubist",neighbors = 5)
```

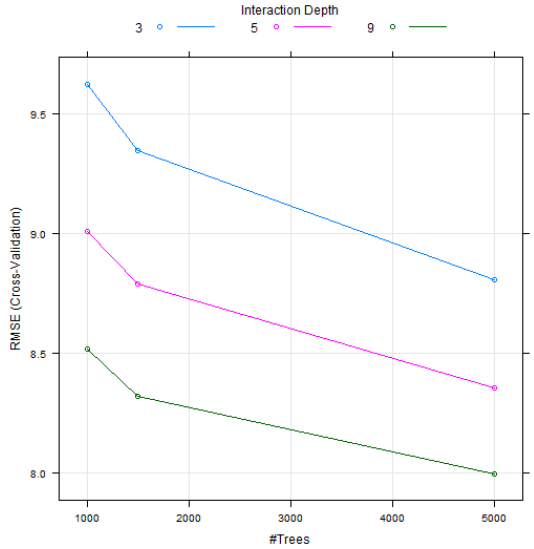
Annexe 3 : Graphiques des calibrations des modèles

Carbone :

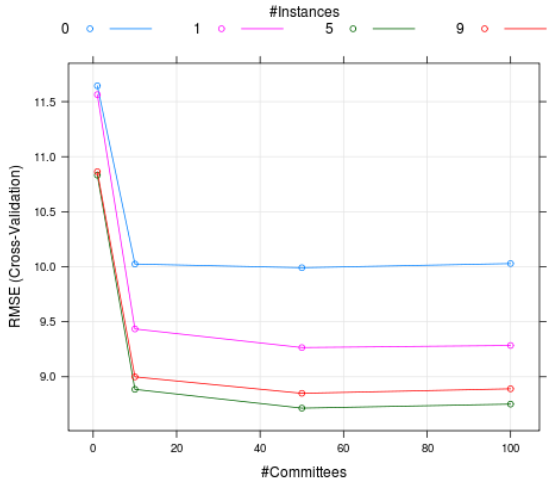
- Val_max, Random Forest :



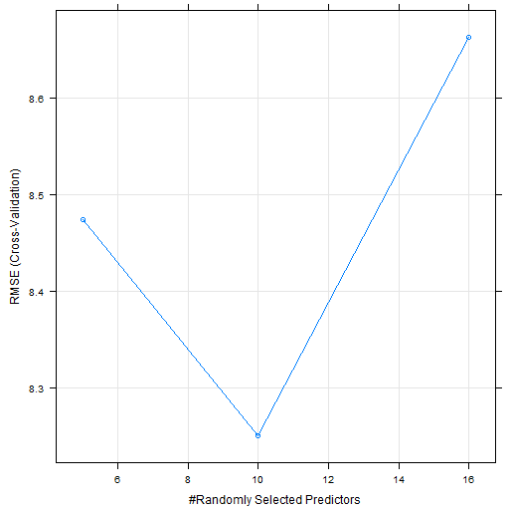
- Val_max, GBM :



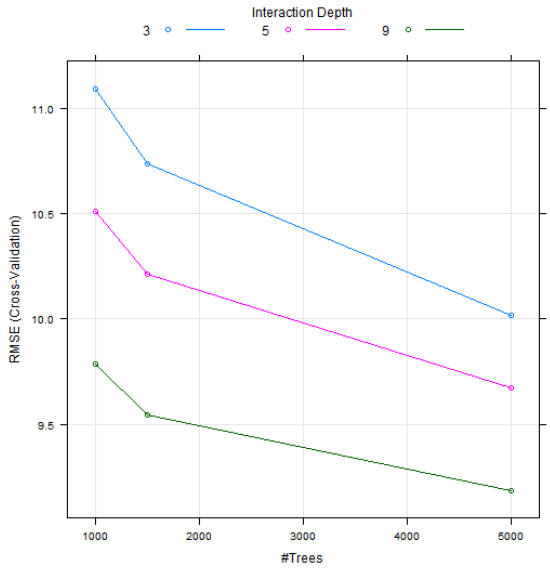
- Val_max, Cubist :



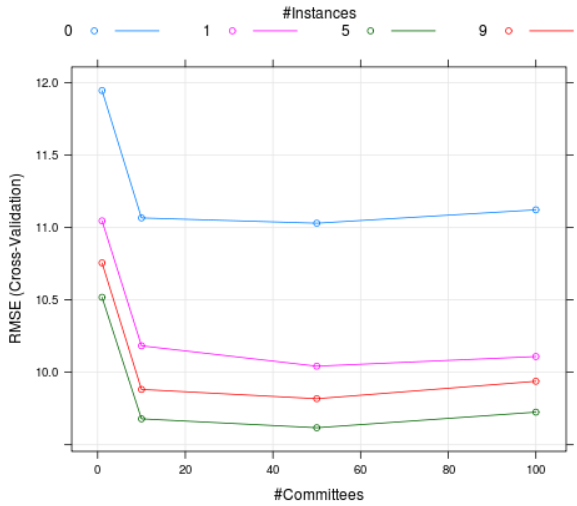
- Etendue, Random Forest :



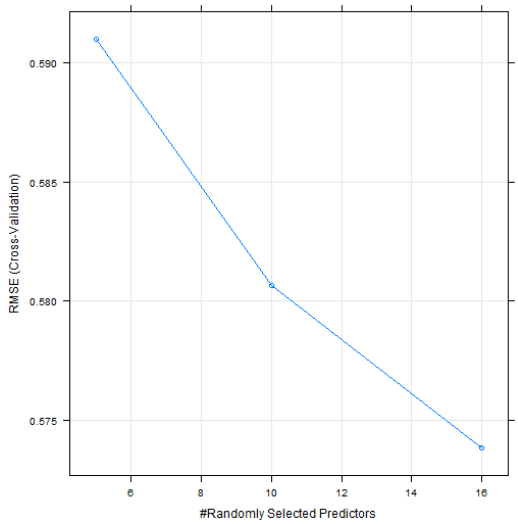
- Etendue, GBM :



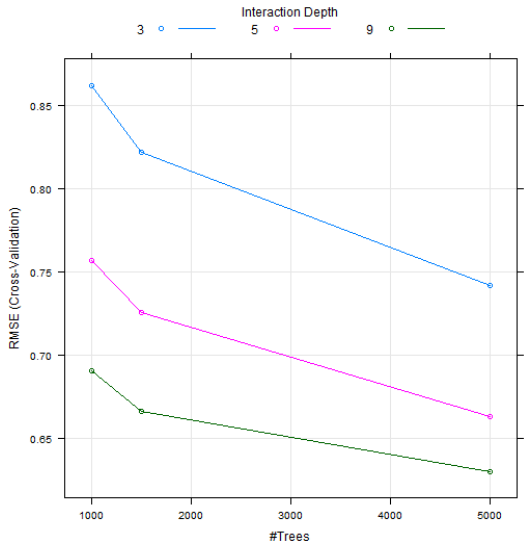
- Etendue, Cubist :



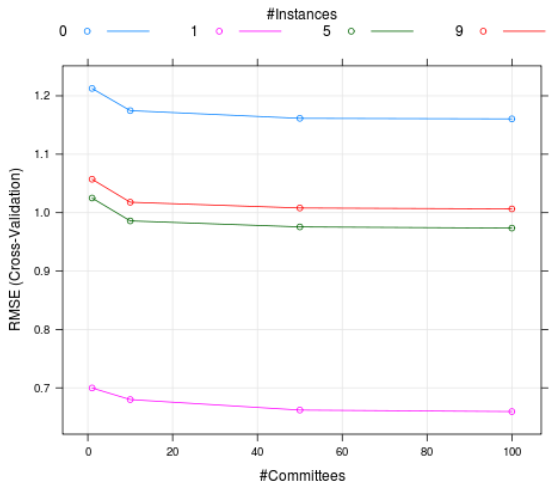
- Etendue_norm, Random Forest :



- Etendue_norm, GBM :

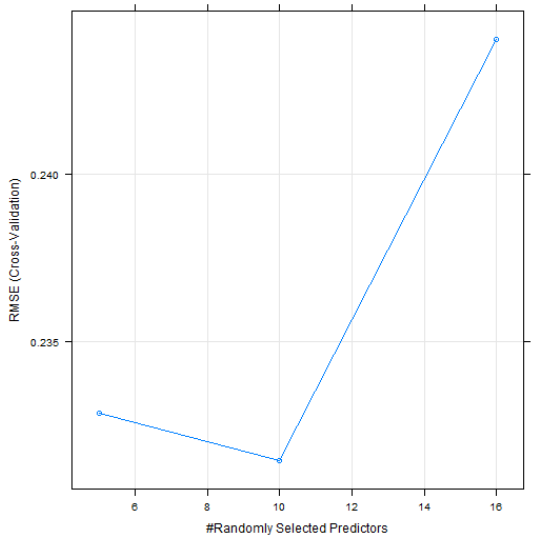


- Etendue_norm, Cubist :

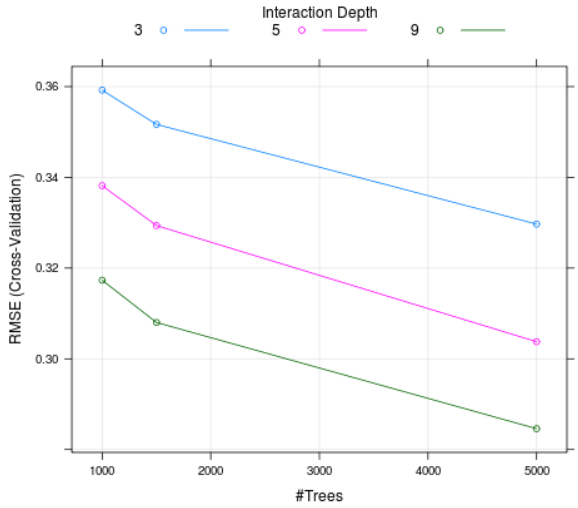


pH :

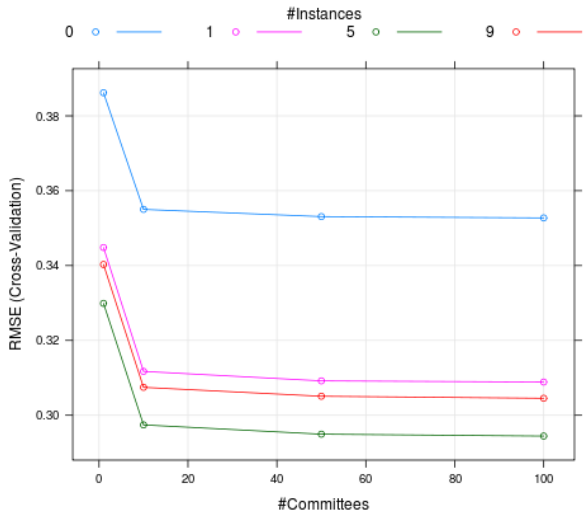
- Val_min, Random Forest :



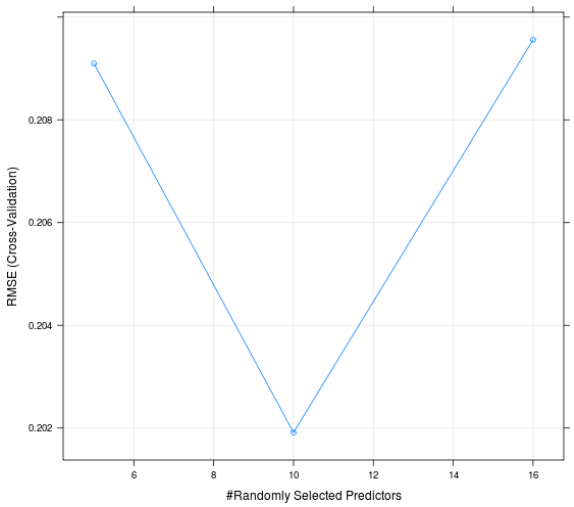
- Val_min, GBM :



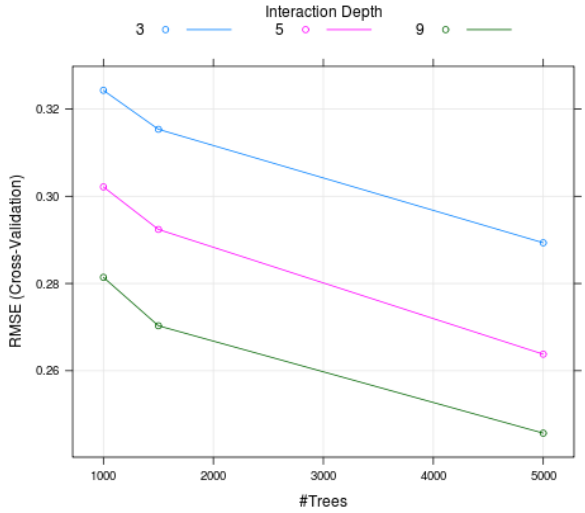
- Val_min, Cubist :



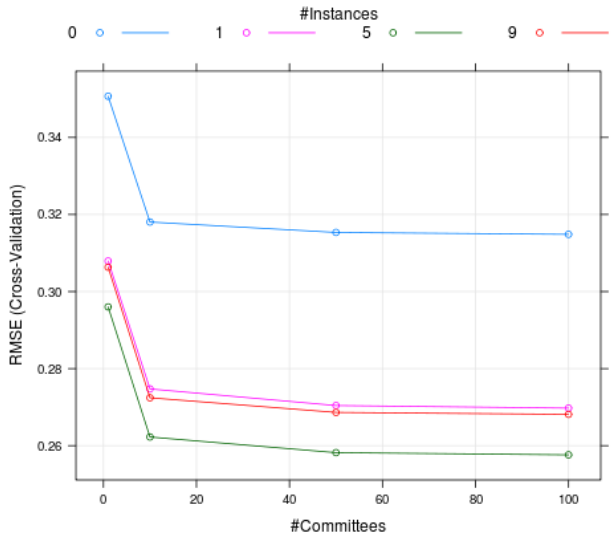
- Val_max, Random Forest :



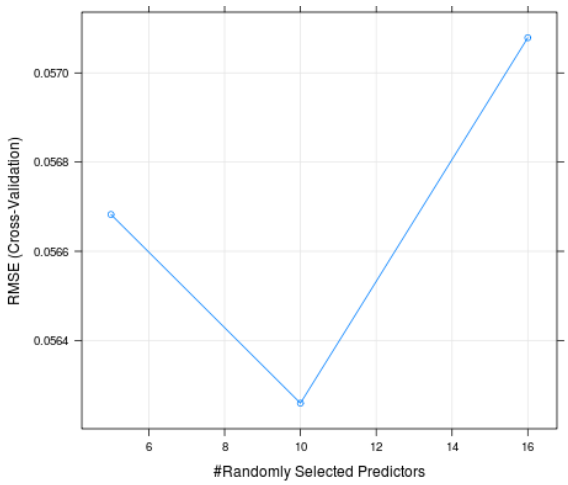
- Val_max, GBM :



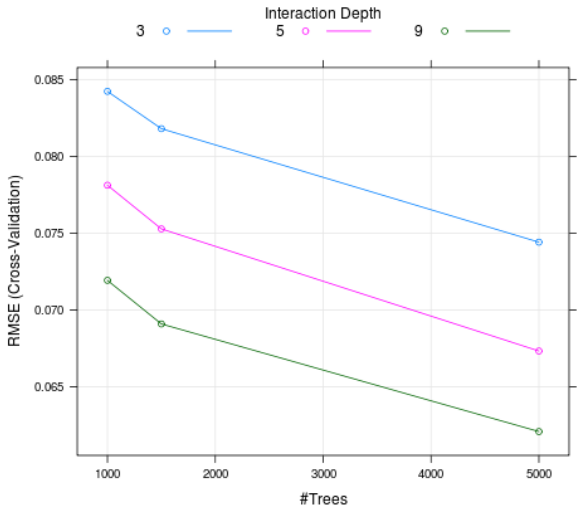
- Val_max, Cubist :



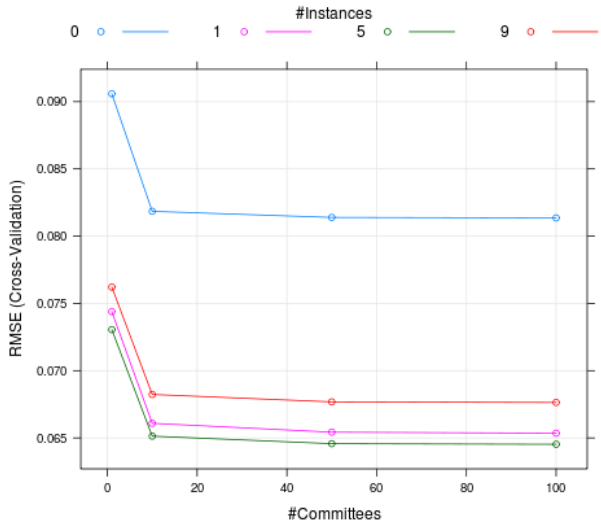
- Etendue_norm, Random Forest :



- Etendue_norm, GBM :



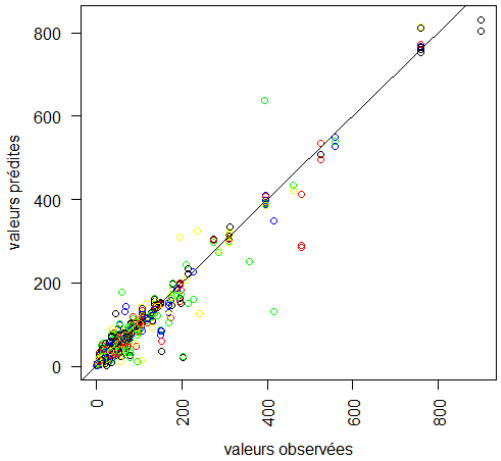
- Etendue_norm, Cubist :



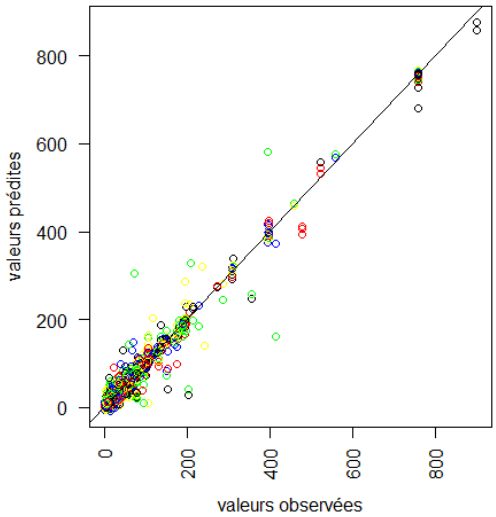
Annexe 4 : Graphes des prédictions en fonction des observations

Carbone :

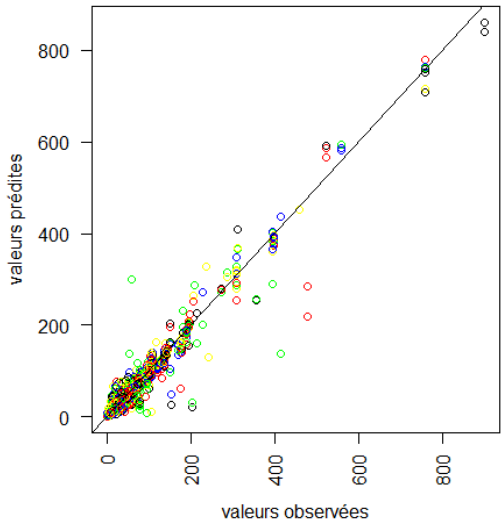
- Val_max, Random Forest :



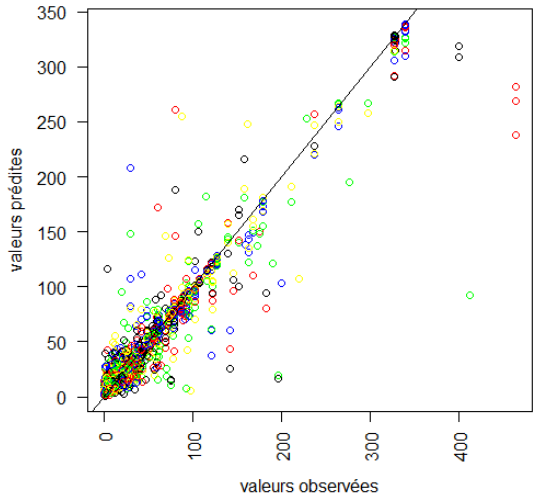
- Val_max, GBM :



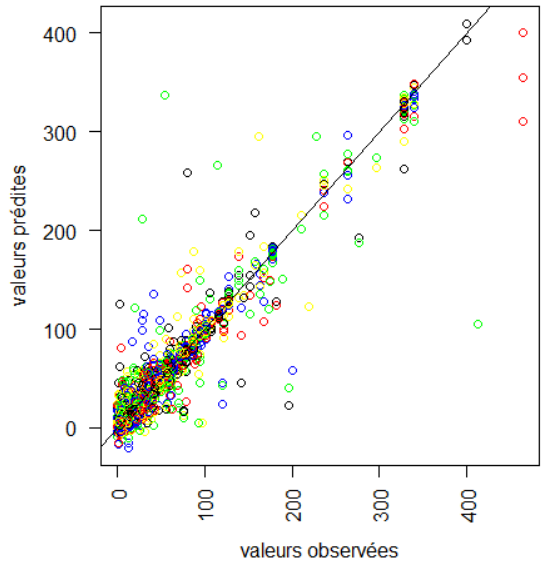
- Val_max, Cubist :



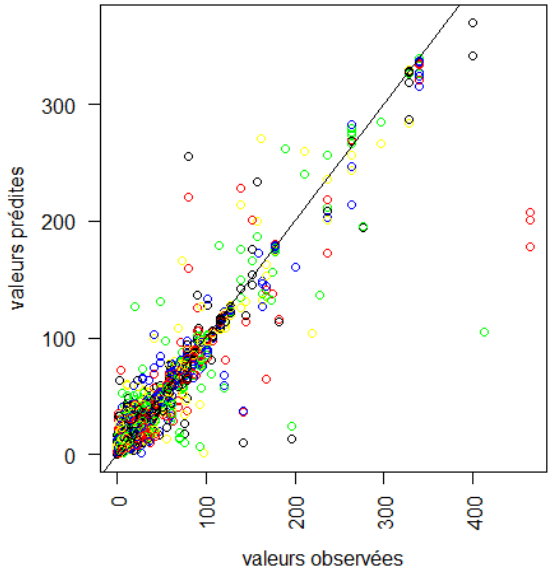
- Etendue, Random Forest :



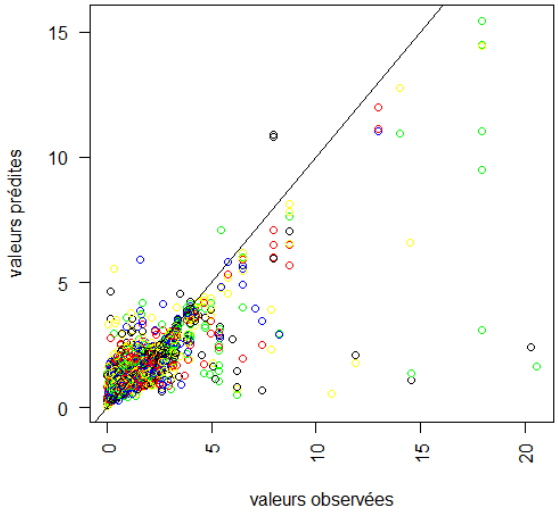
- Etendue, GBM :



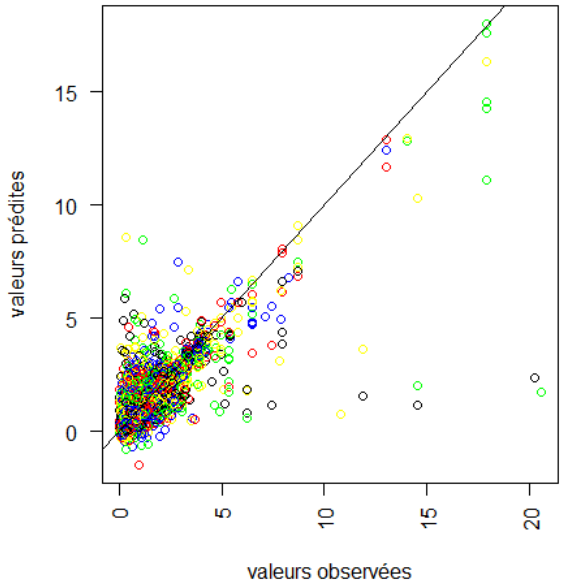
- Etendue, Cubist :



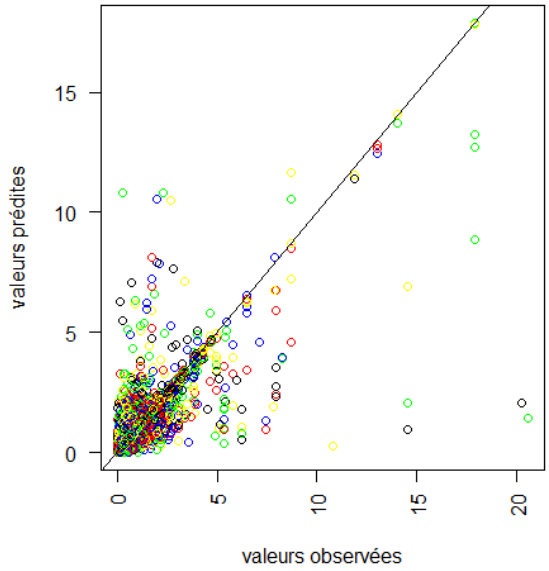
- Etendue_norm, Random forest :



- Etendue_norm, GBM :

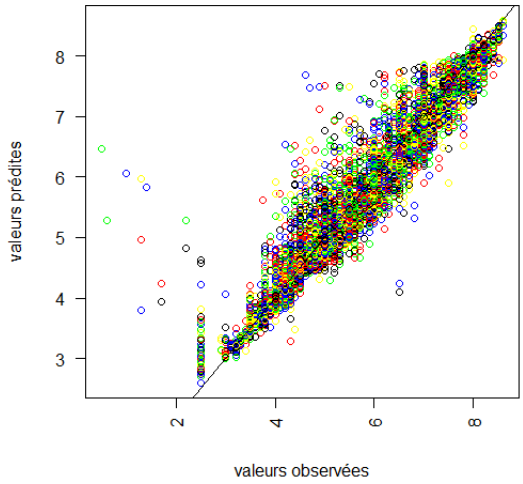


- Etendue_norm, Cubist :

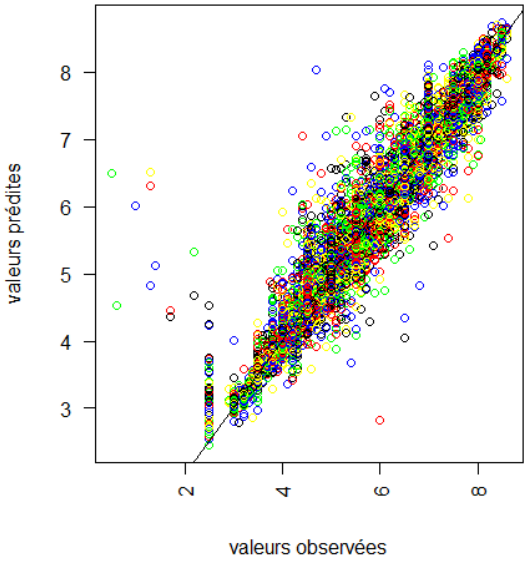


PH :

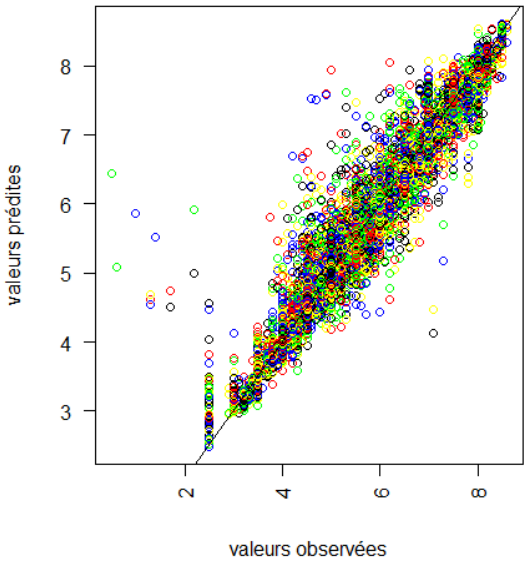
- Val_min, Random Forest :



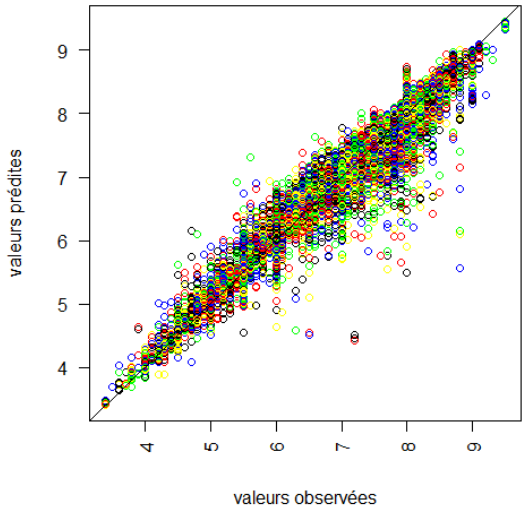
- Val_min, GBM :



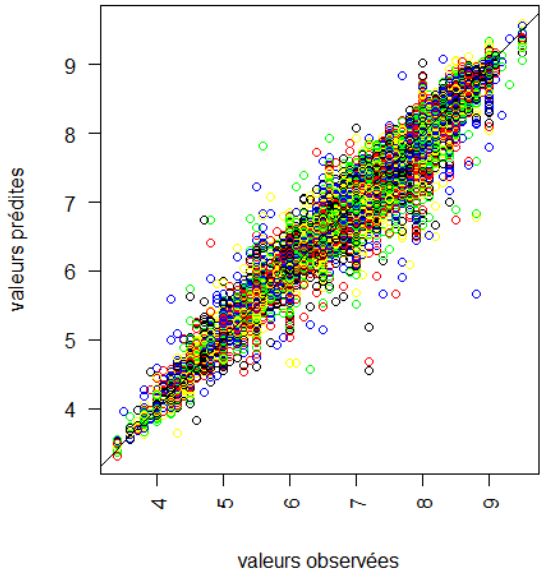
- Val_min, Cubist :



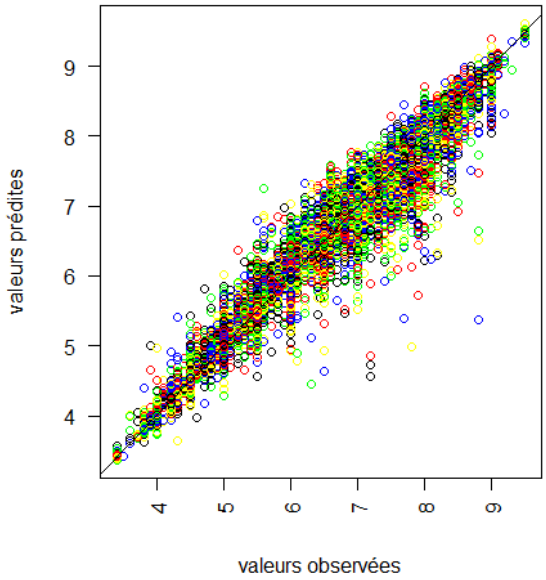
- Val_max, Random Forest :



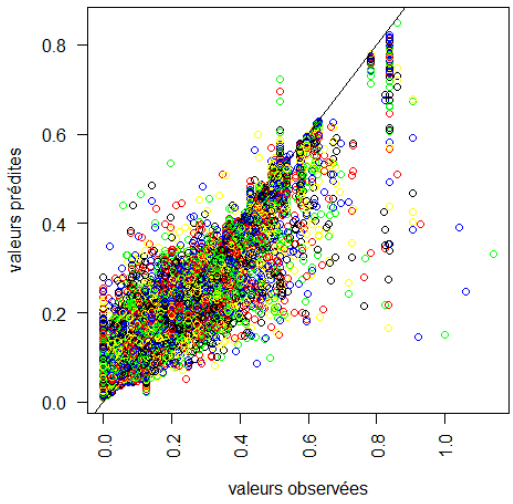
- Val_max, GBM :



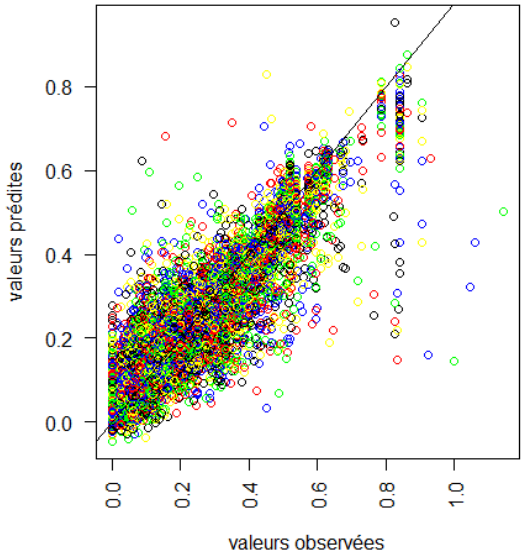
- Val_max, Cubist :



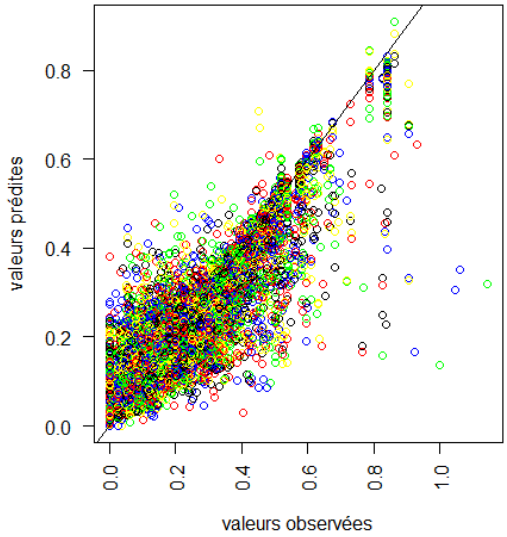
- Etendue_norm, Random Forest :



- Etendue_norm, GBM :



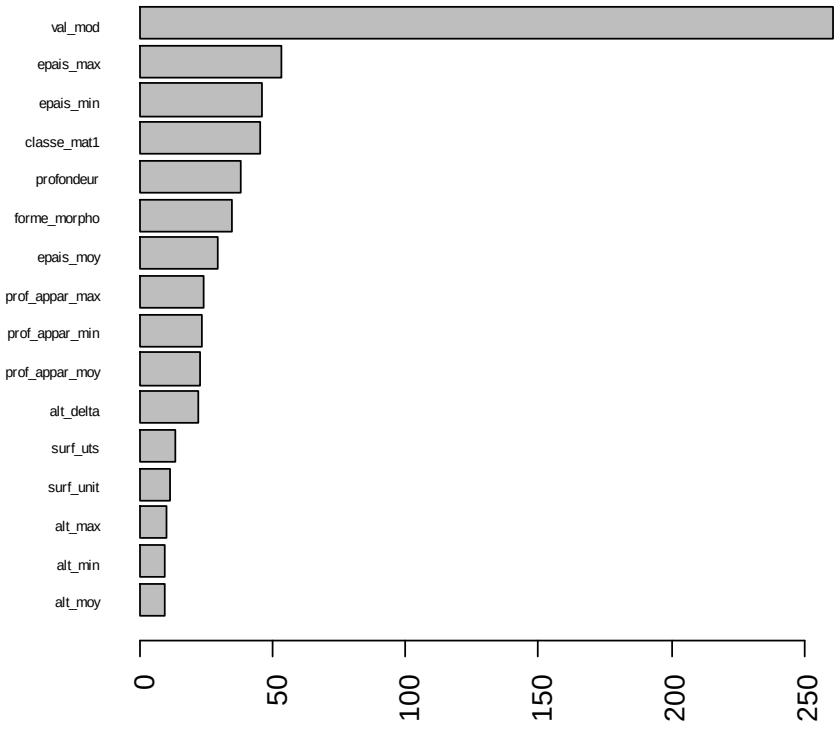
- Etendue_norm, Cubist :



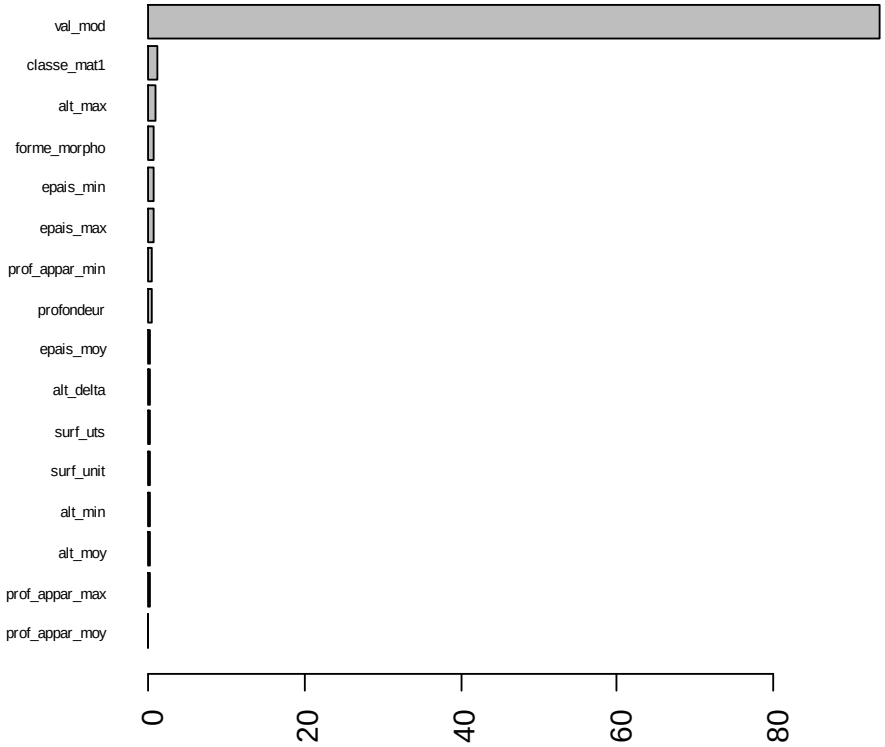
Annexe 5 : Représentation des importances des variables

Carbone :

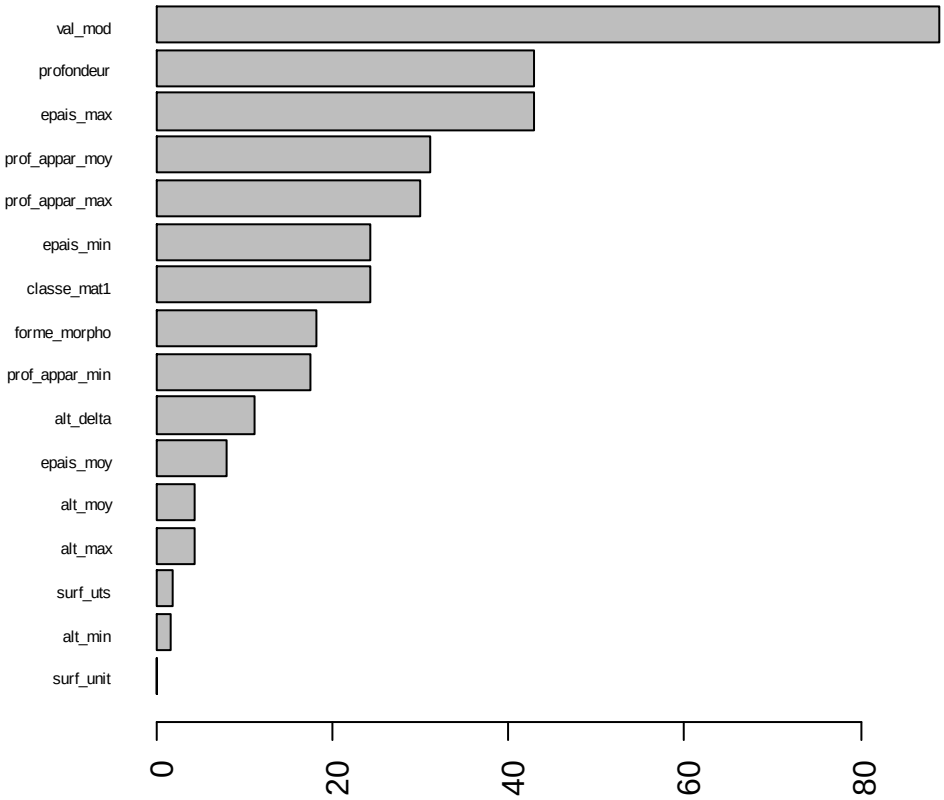
- Val_max, Random Forest :



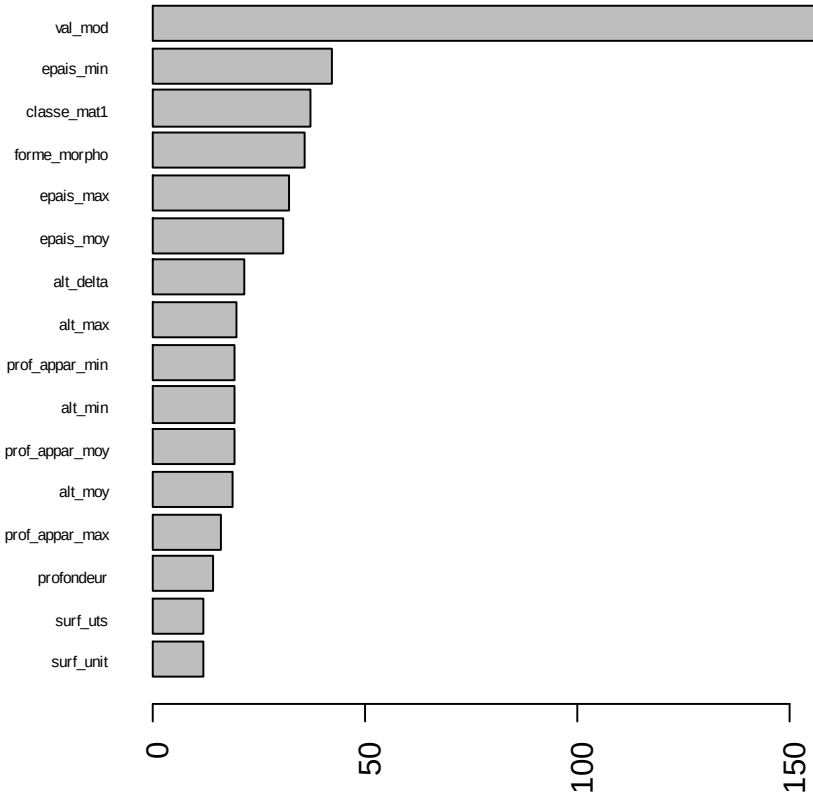
- Val_max, GBM :



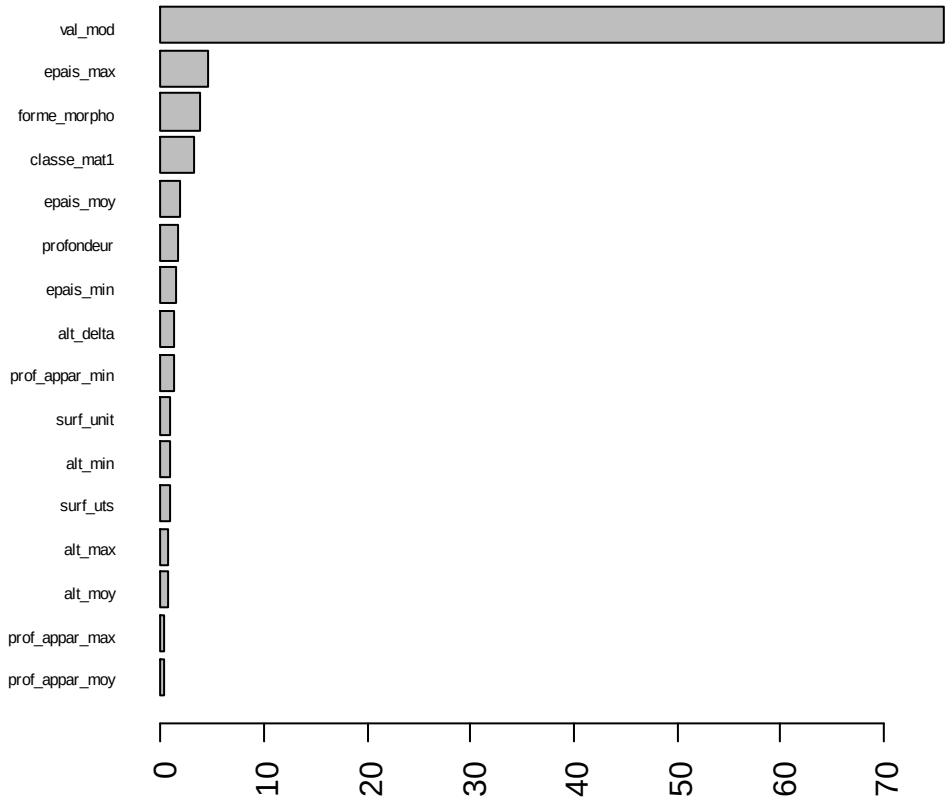
- Val_max, Cubist :



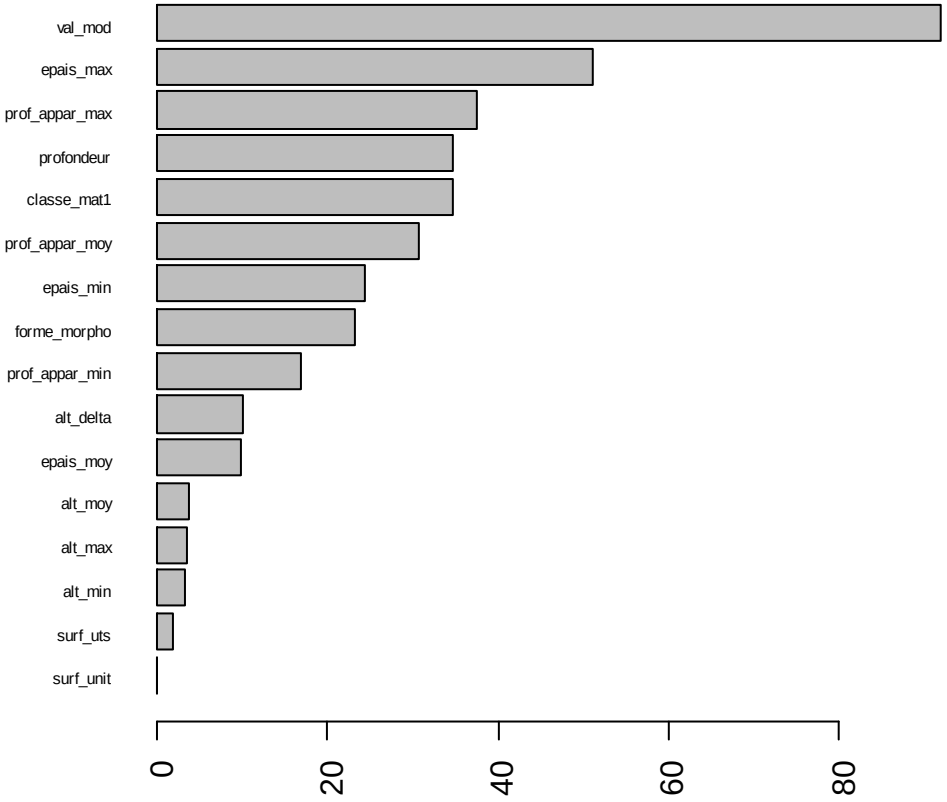
- Etendue, Random Forest :



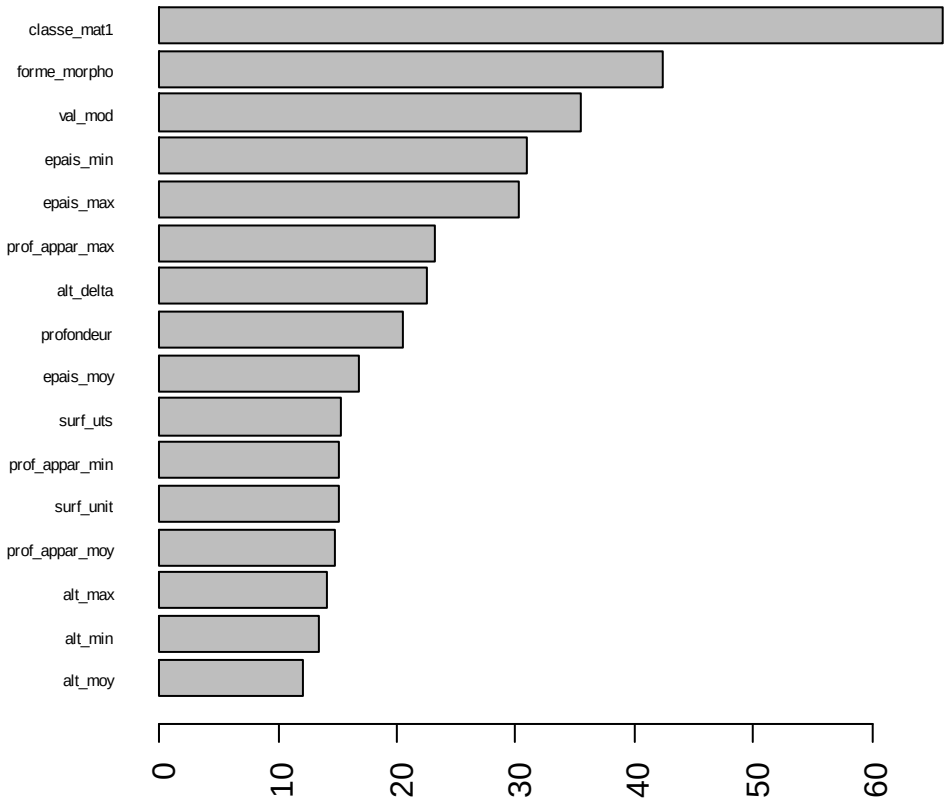
- Etendue, GBM :



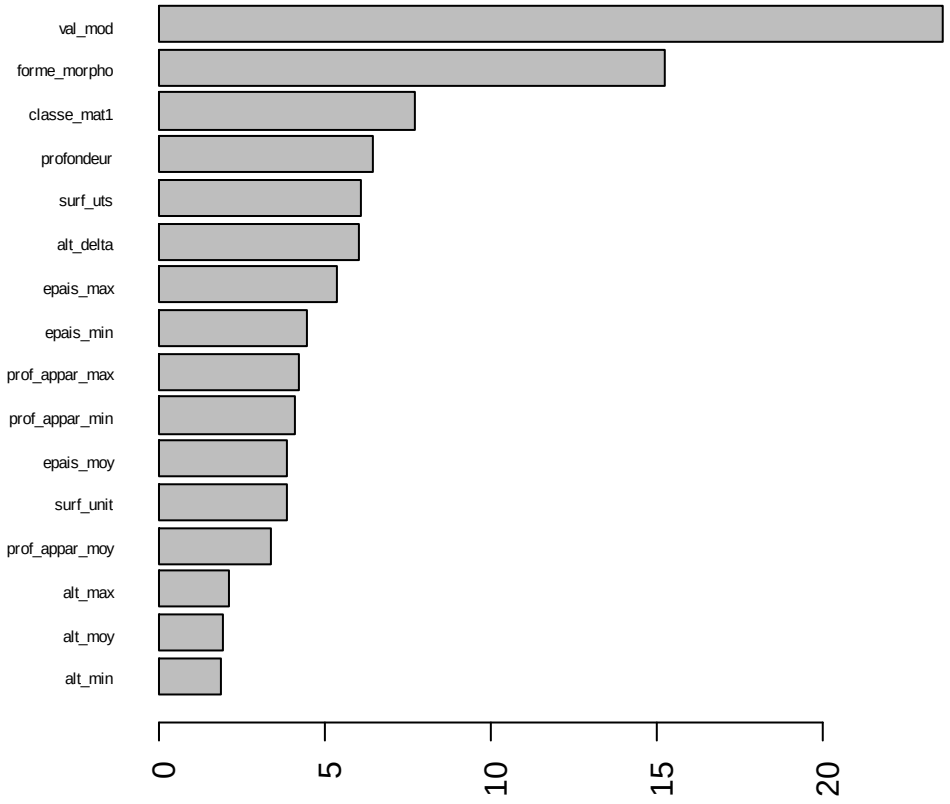
- Etendue, Cubist :



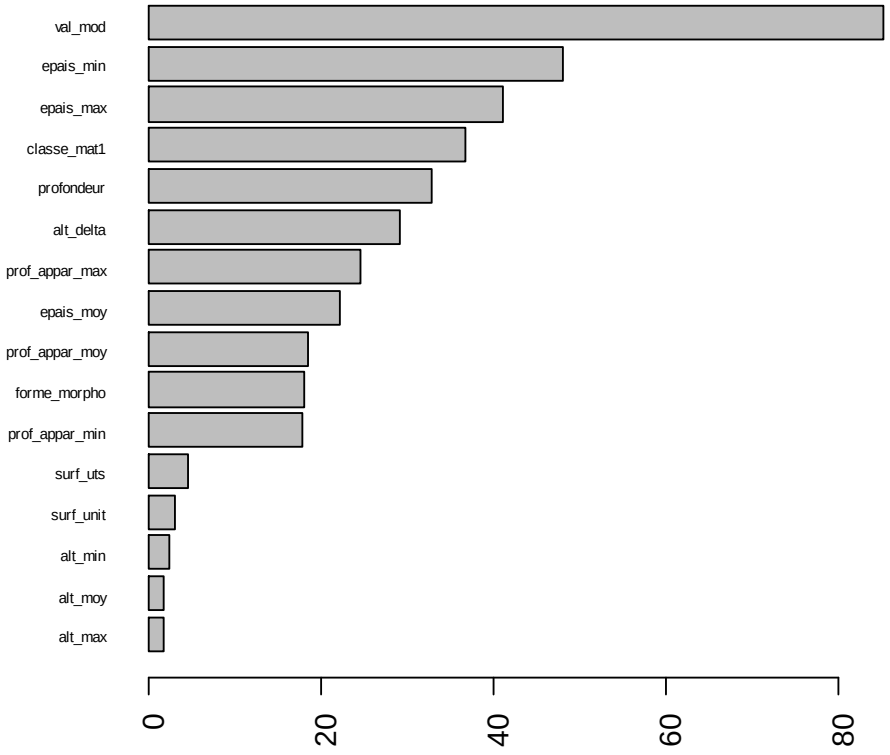
- Etendue_norm, Random Forest :



- Etendue_norm, GBM :

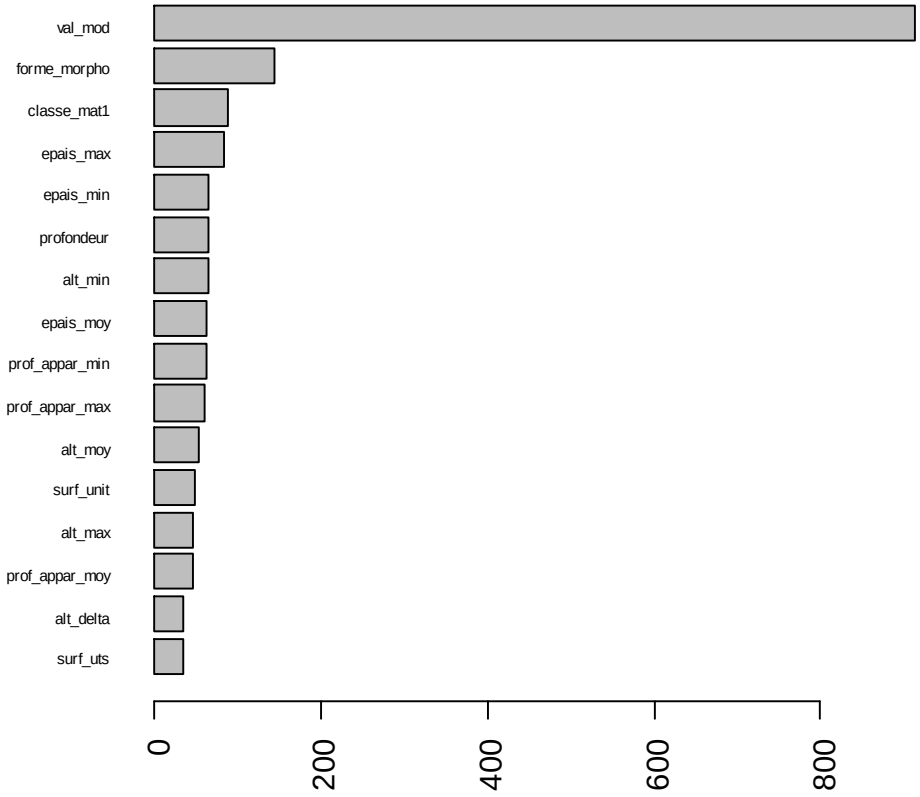


- Etendue_norm, Cubist :

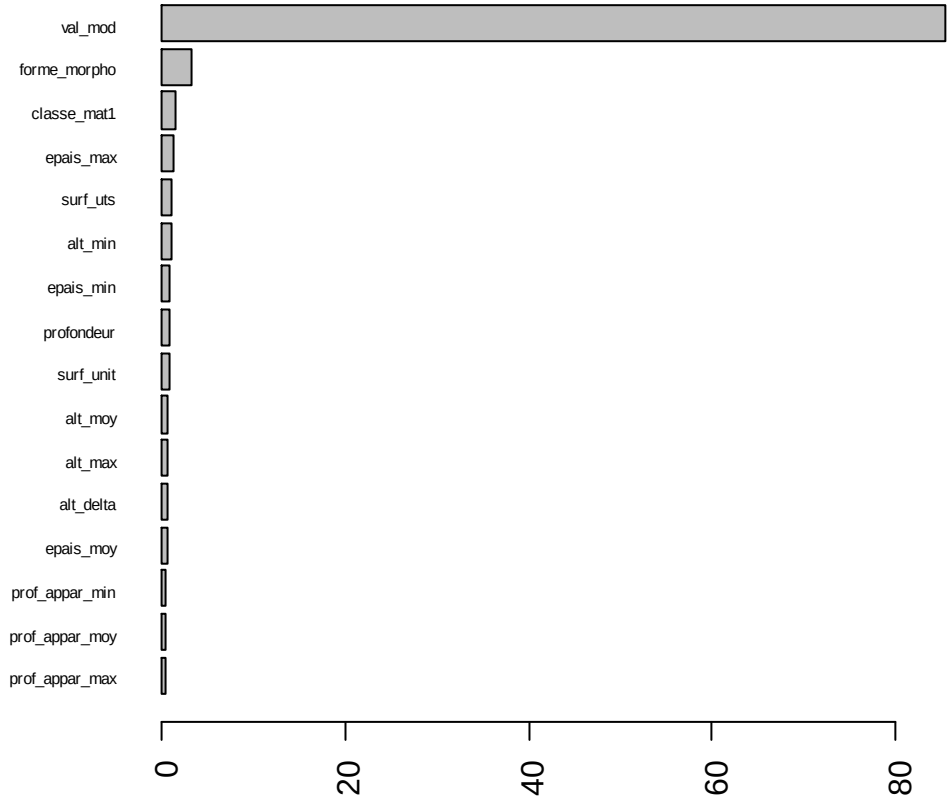


PH :

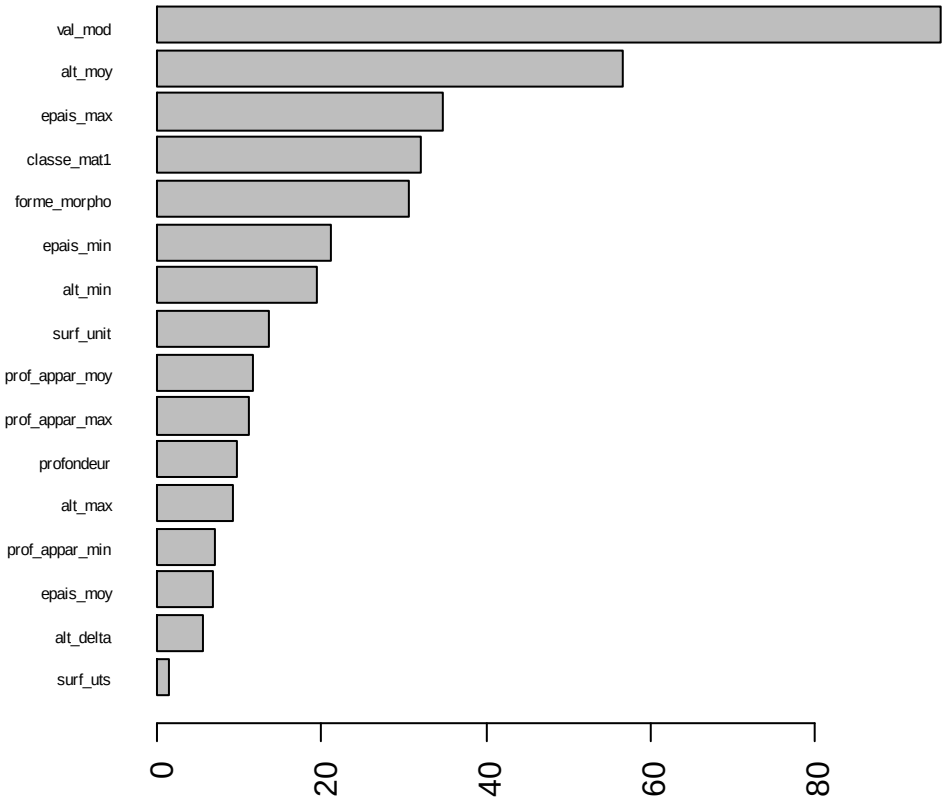
- Val_min, Random Forest :



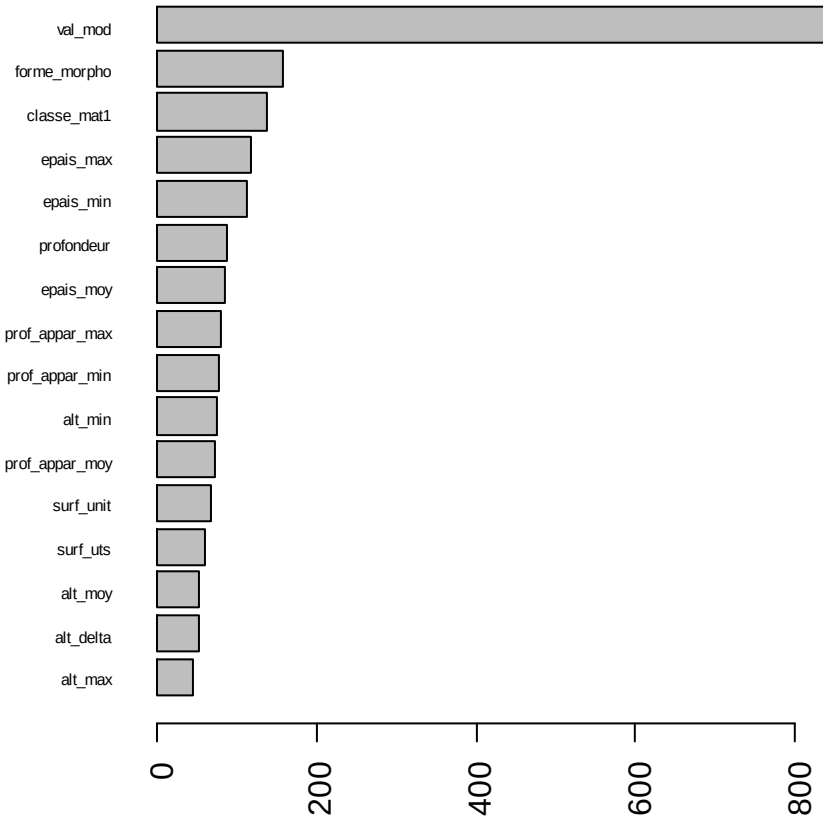
- Val_min, GBM :



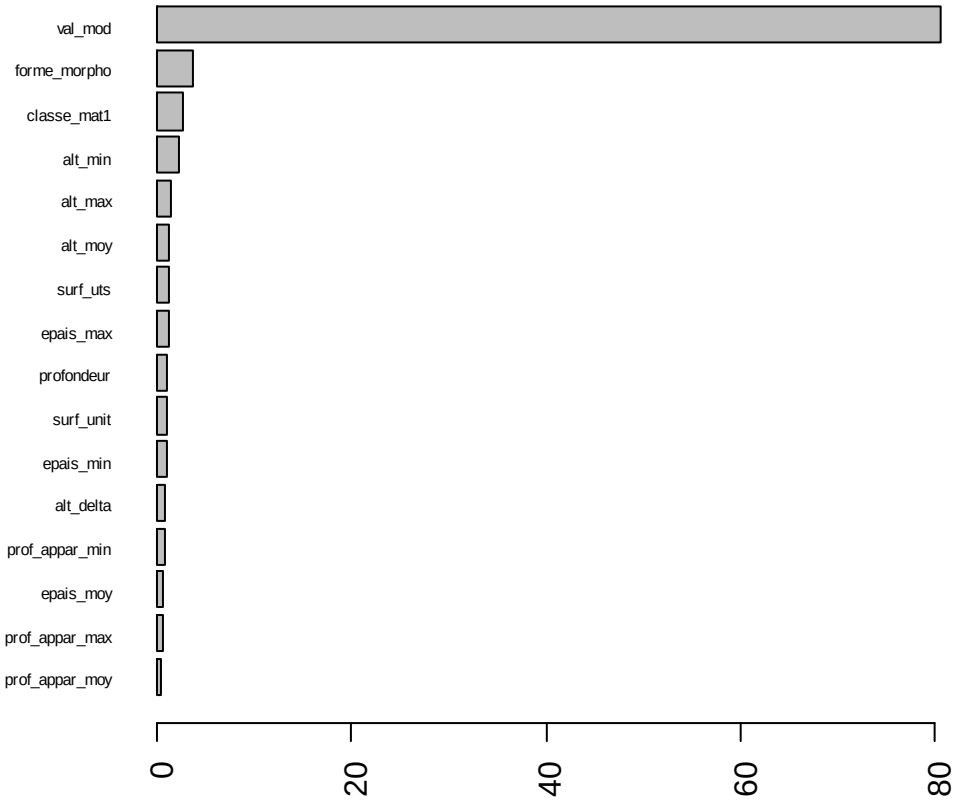
- Val_min, Cubist :



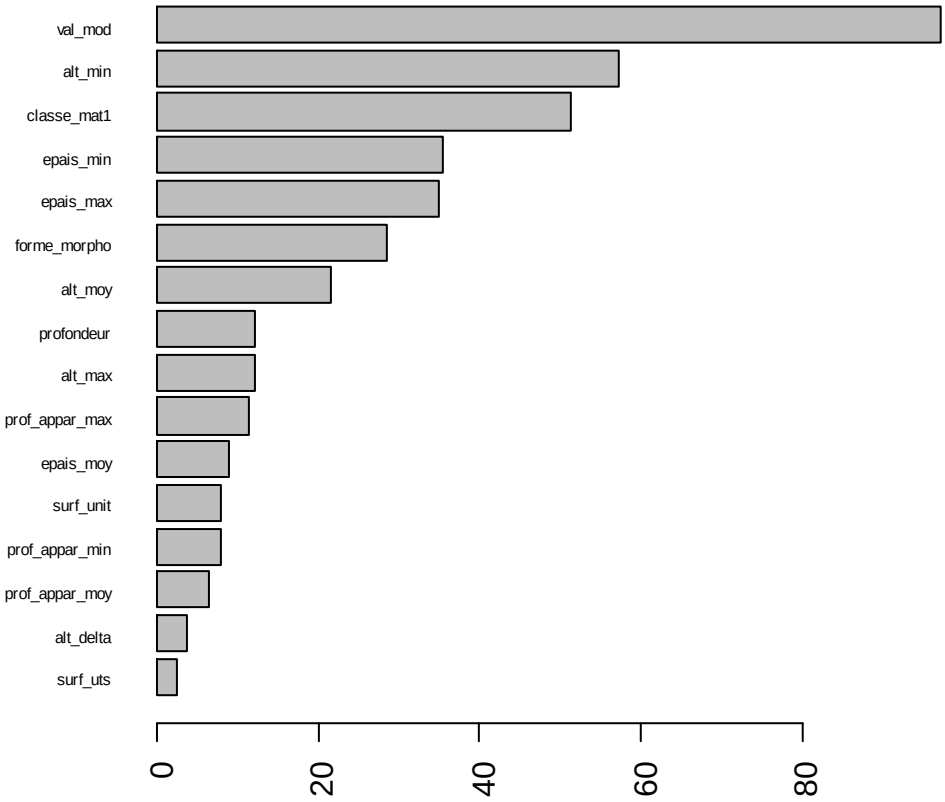
- Val_max, Random Forest :



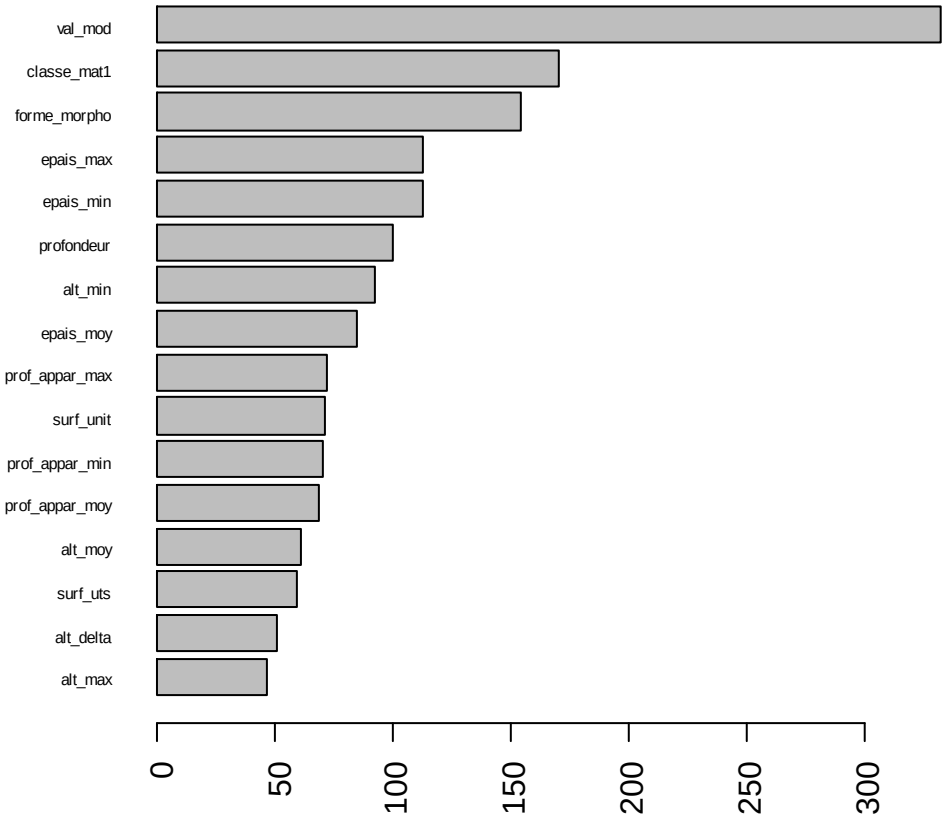
- Val_max, GBM :



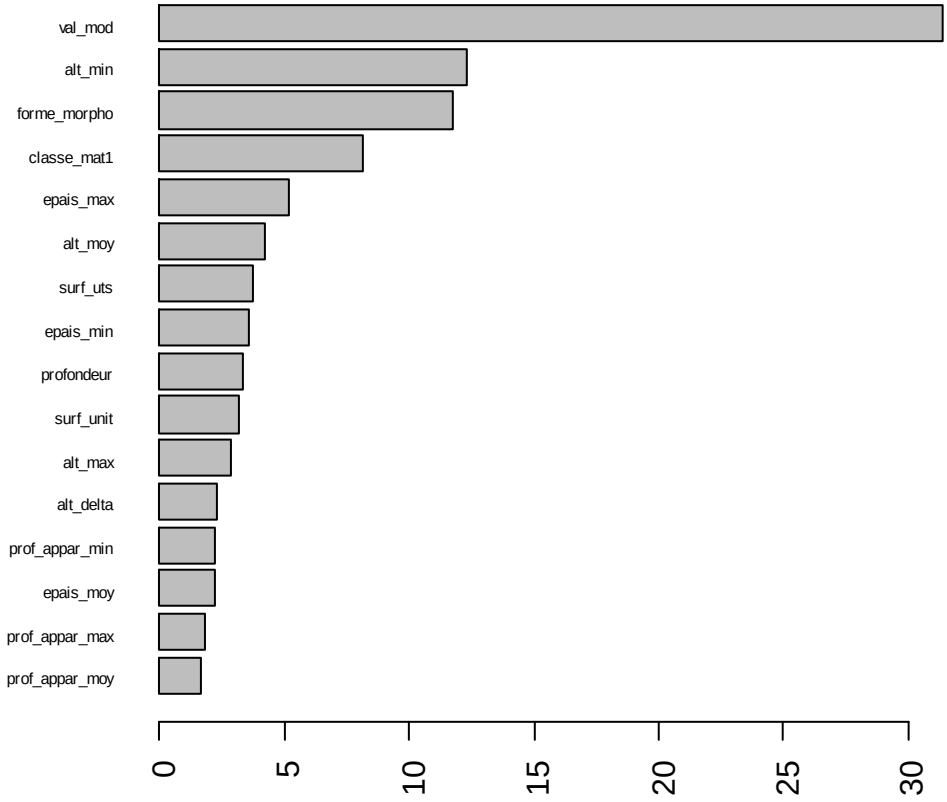
- Val_max, Cubist :



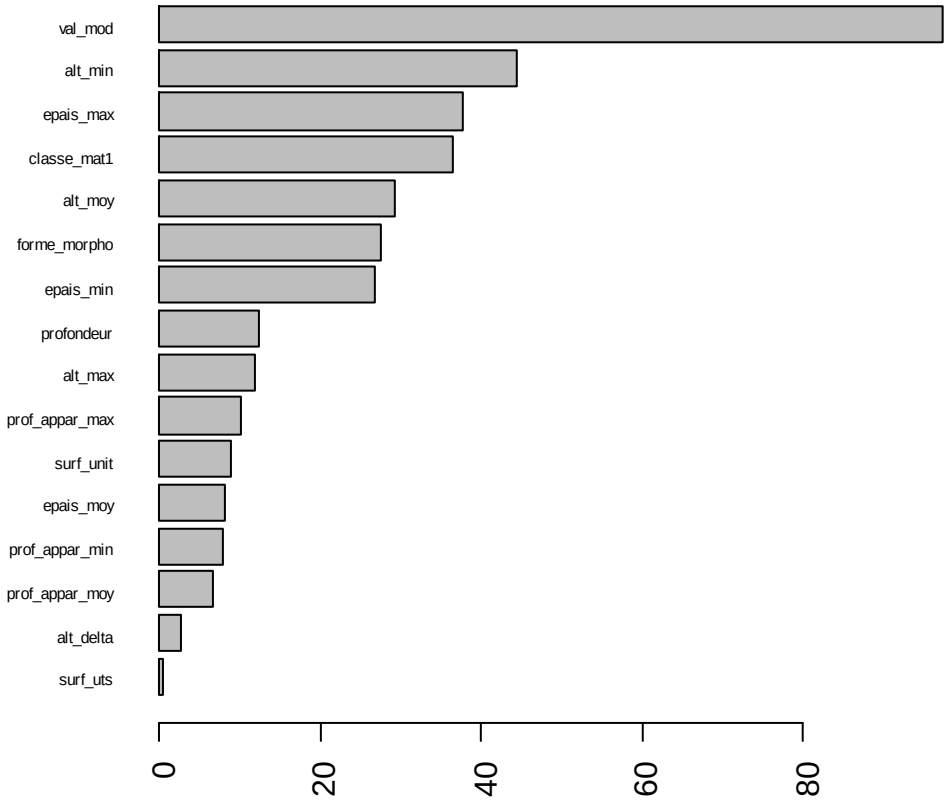
- Etendue_norm, Random Forest :



- Etendue_norm, GBM :



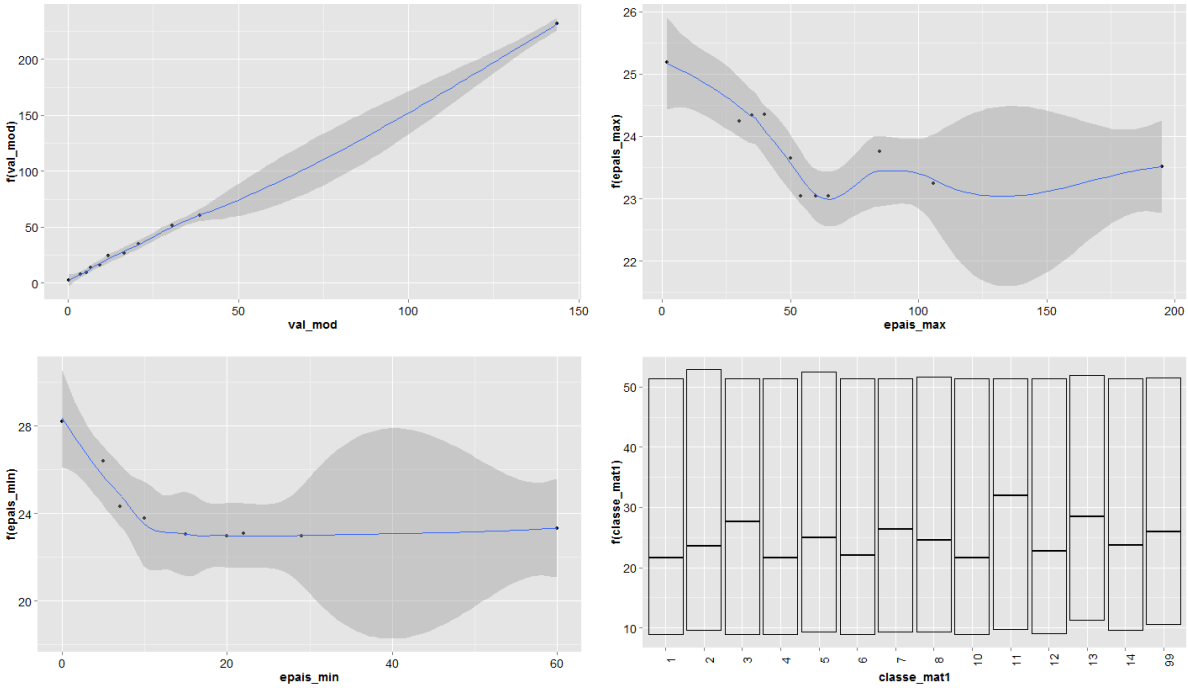
- Etendue_norm, Cubist :



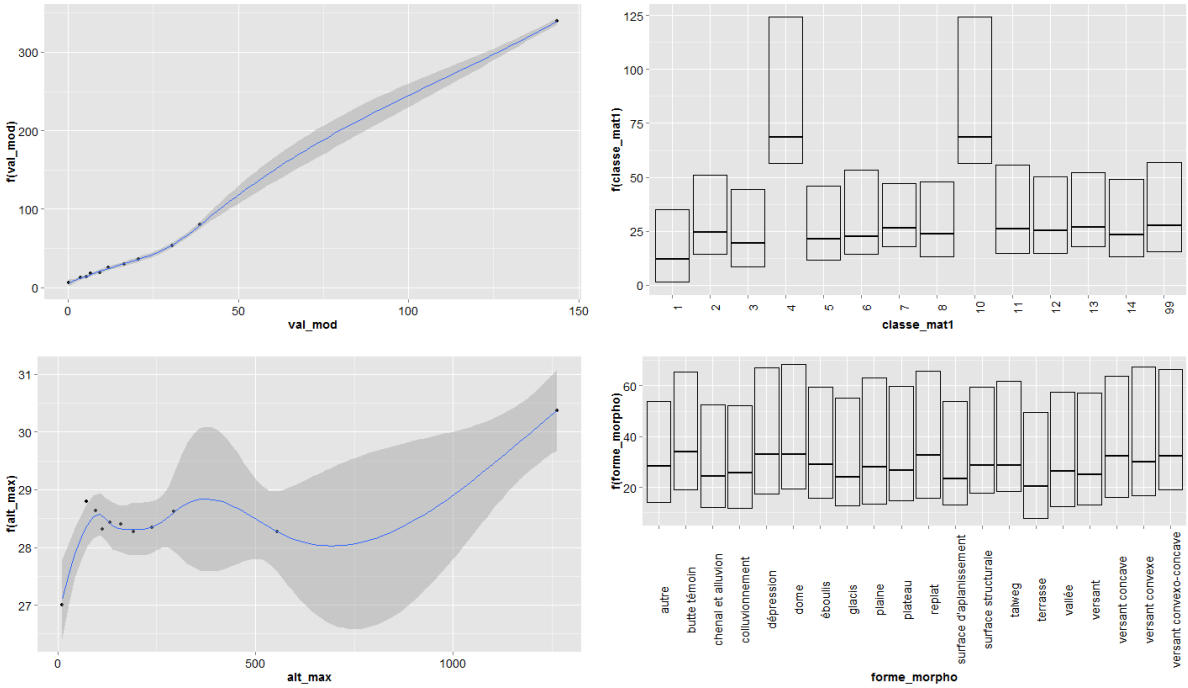
Annexe 6 : Représentation de l'effet des 4 variables les plus importantes sur la prédiction

Carbone :

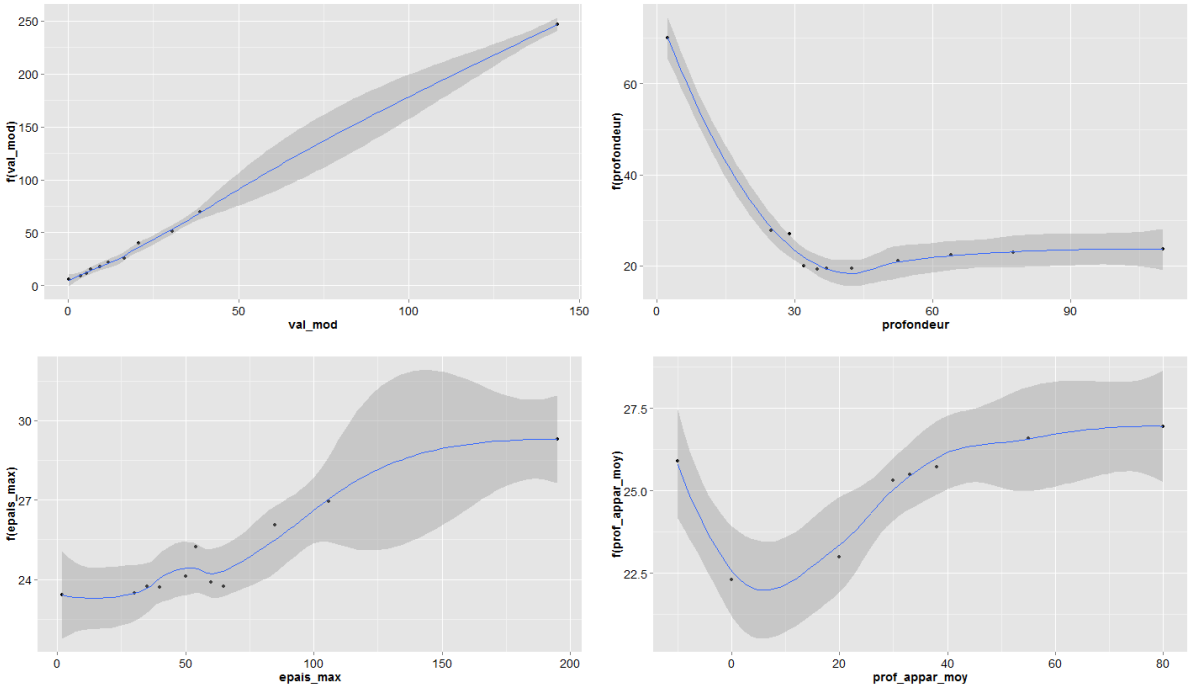
- Val_max, Random Forest :



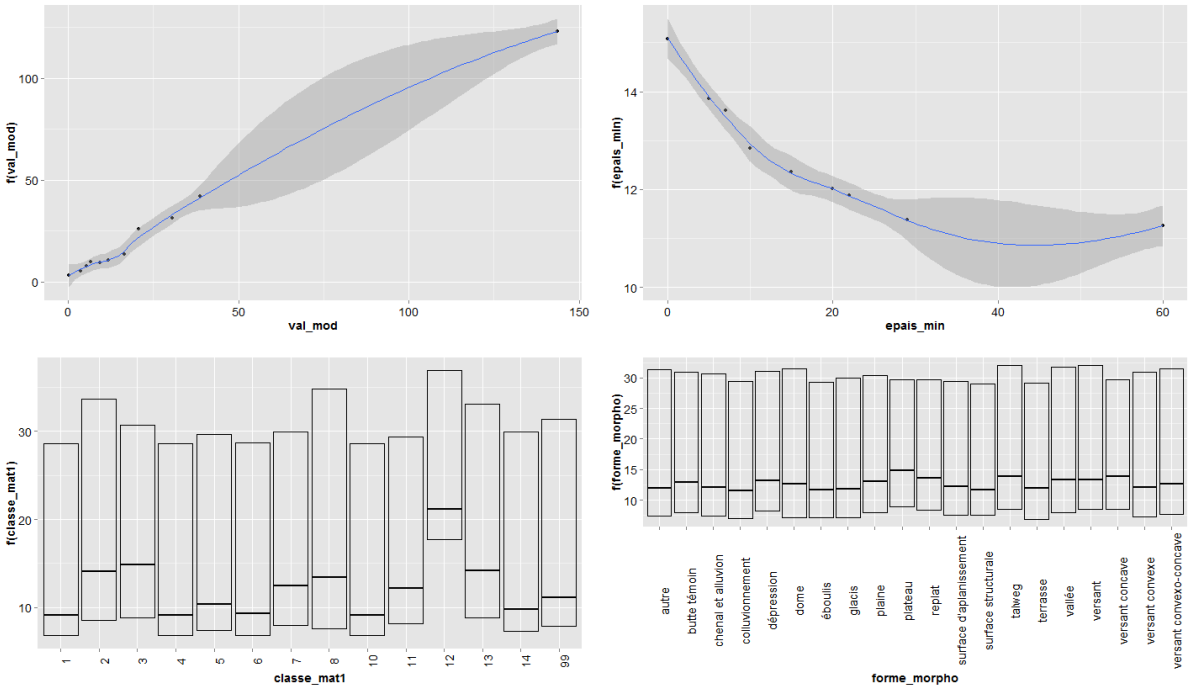
- Val_max, GBM :



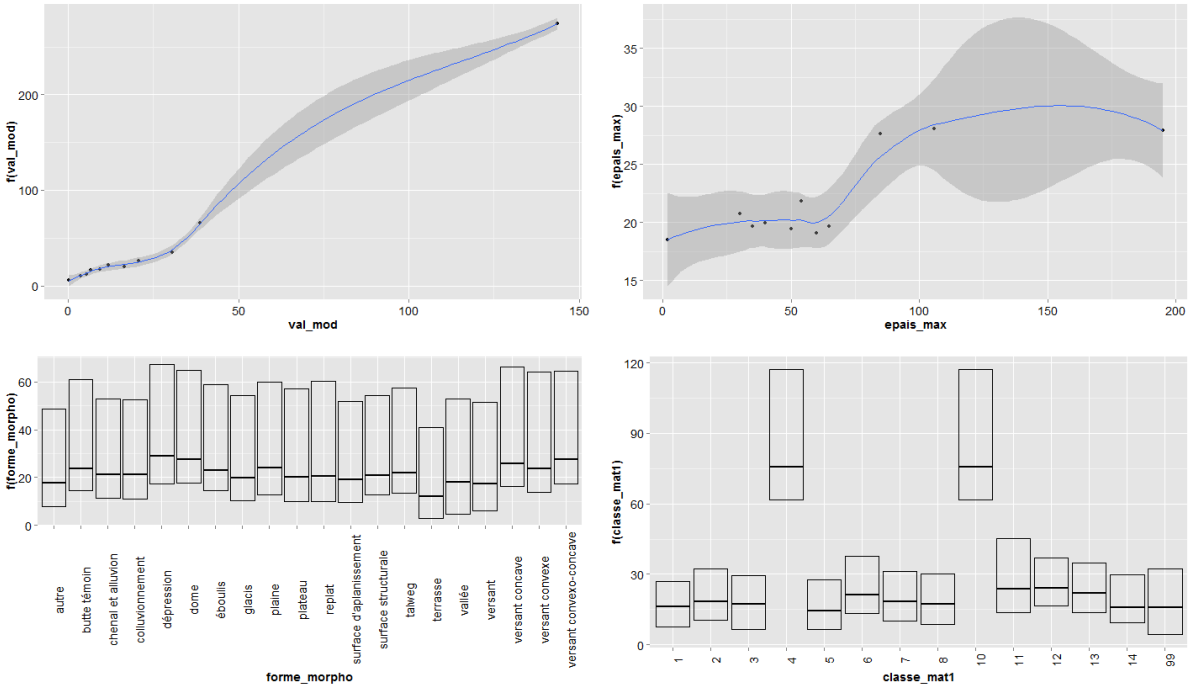
- Val_max, Cubist :



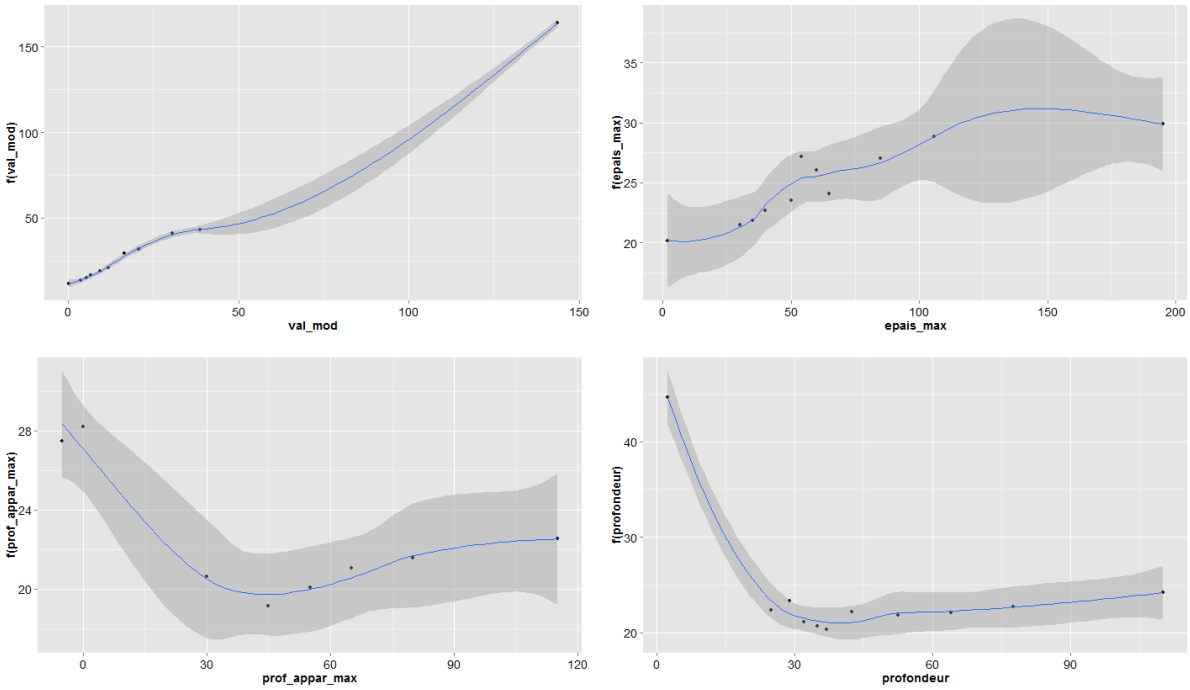
- Etendue, random Forest :



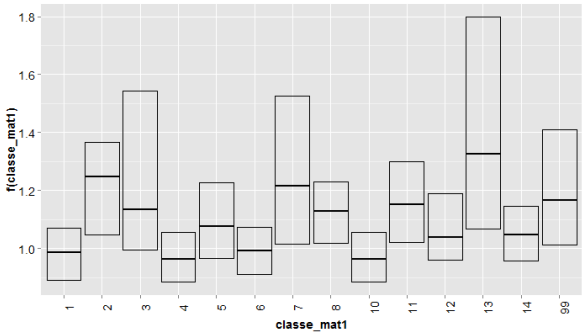
- Etendue, GBM :

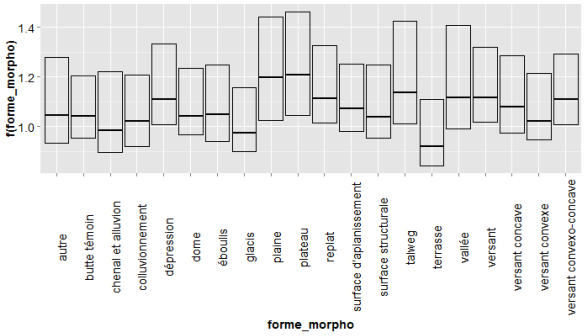


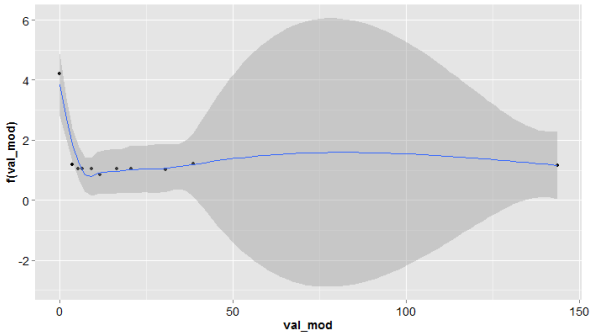
- Etendue, Cubist :

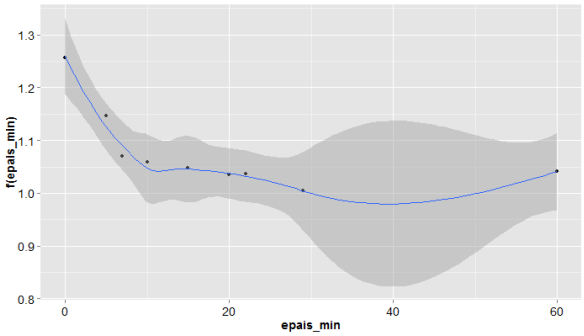


- Etendue_norm, Random Forest :

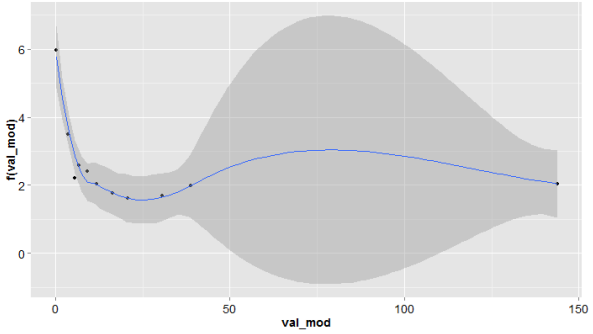


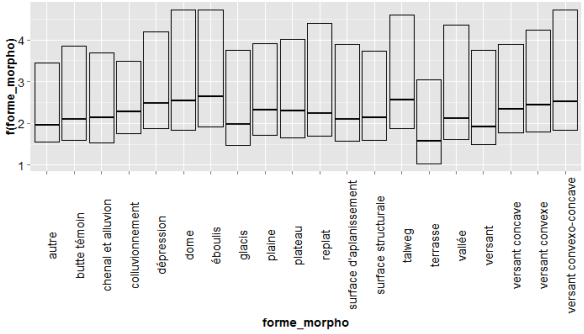


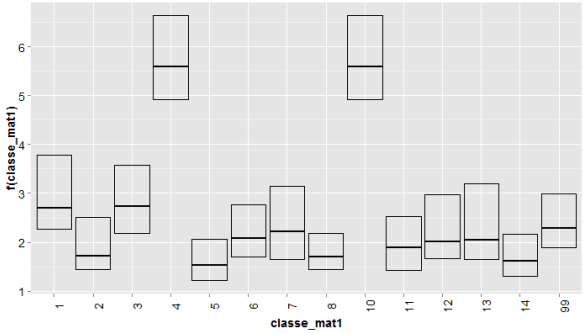


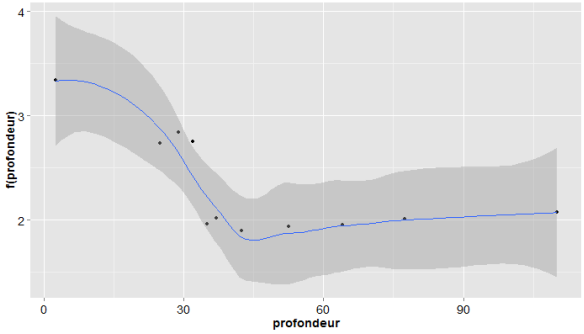


- Etendue_norm, GBM :

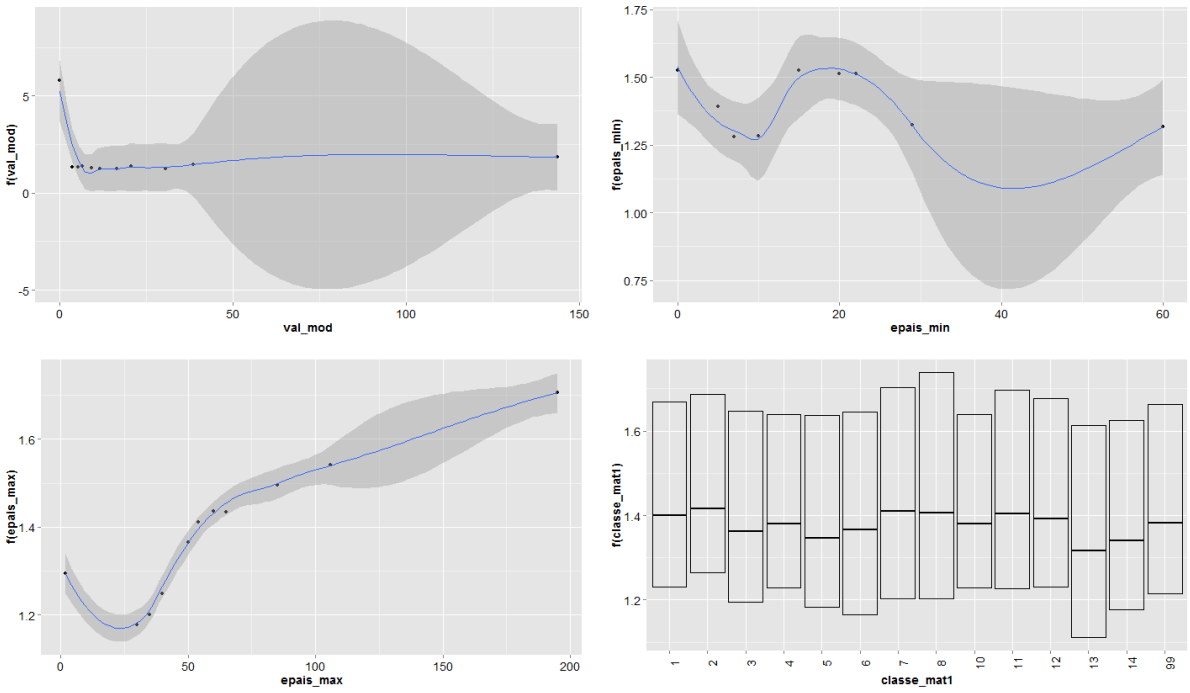






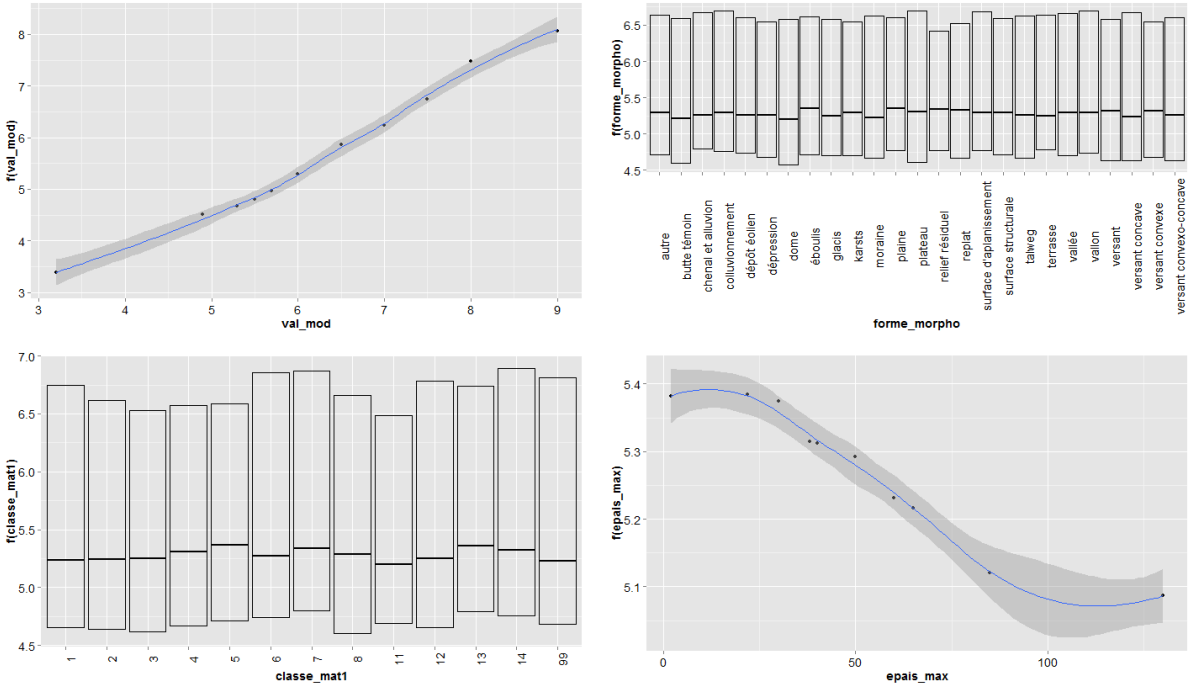


- Etendue_norm, Cubist :

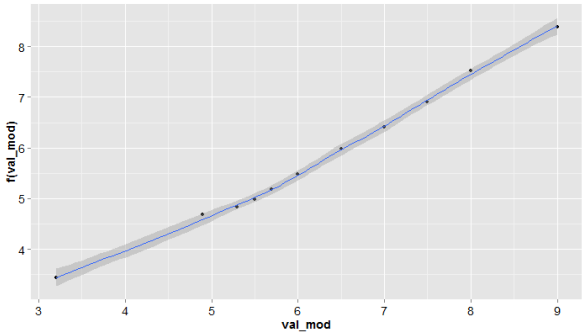


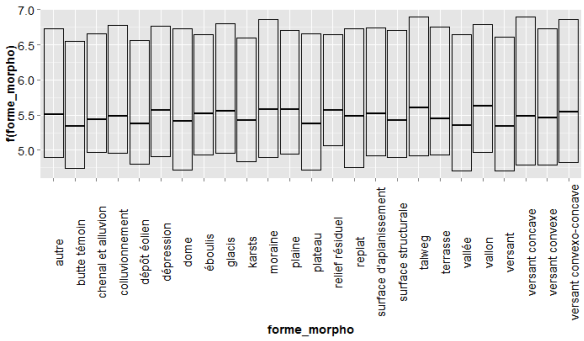
PH :

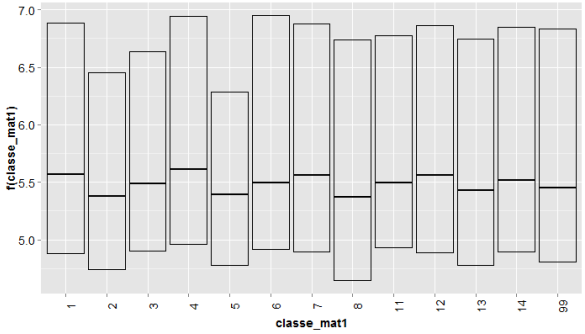
- Val_min, Random Forest :

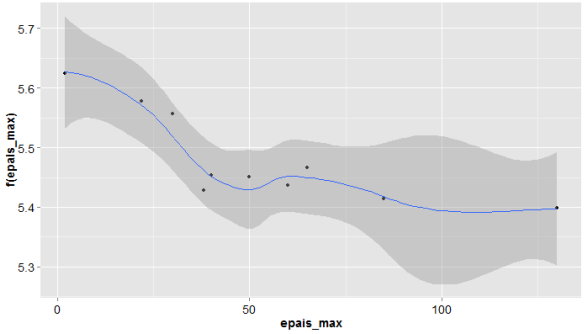


- Val_min, GBM :

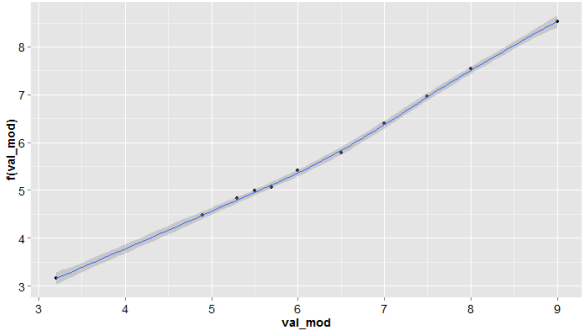


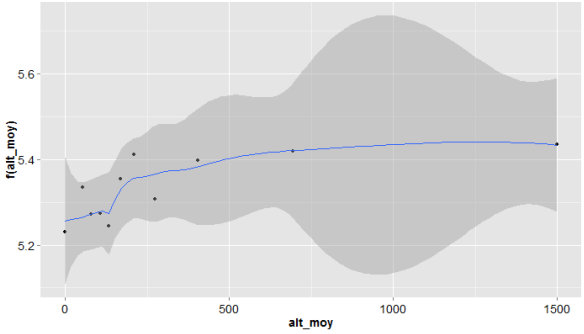


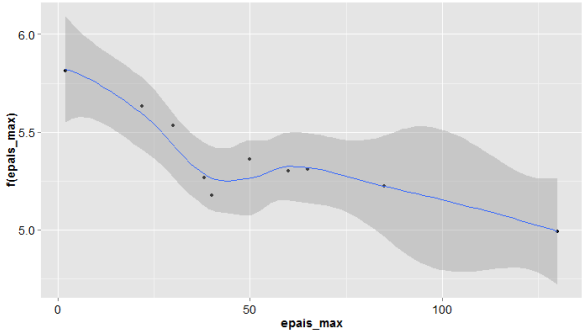


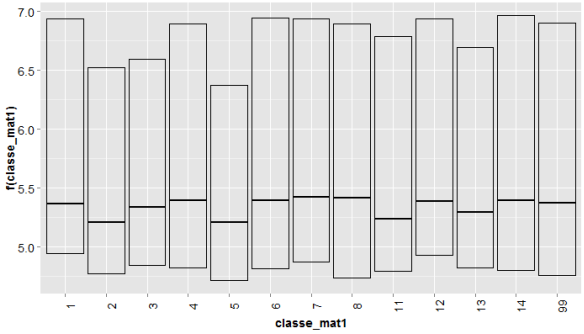


- Val_min, Cubist :

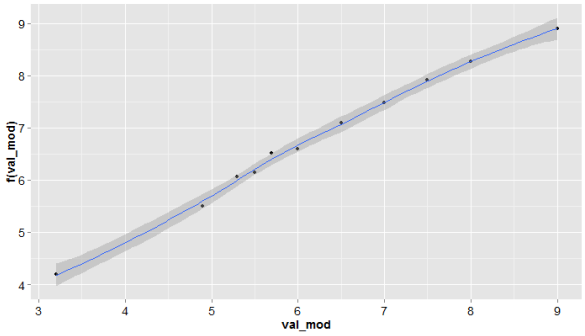


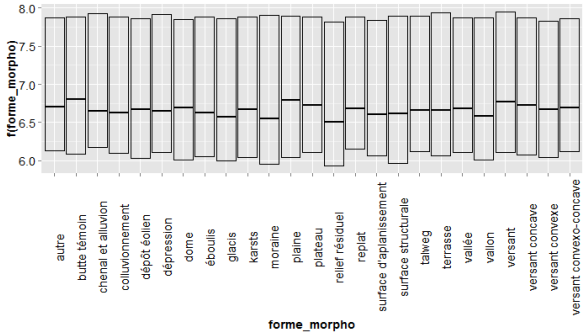


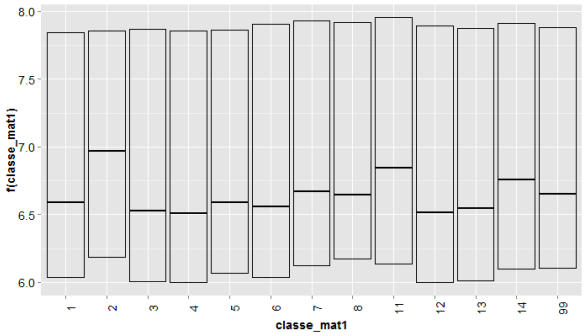


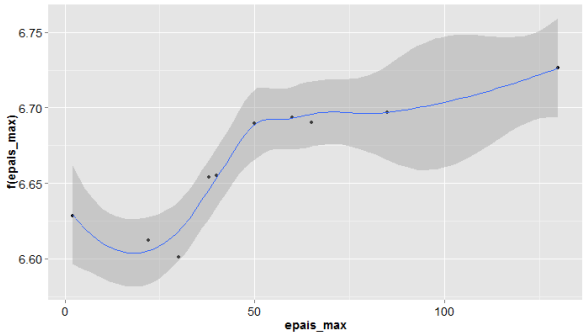


- Val_max, Random Forest :

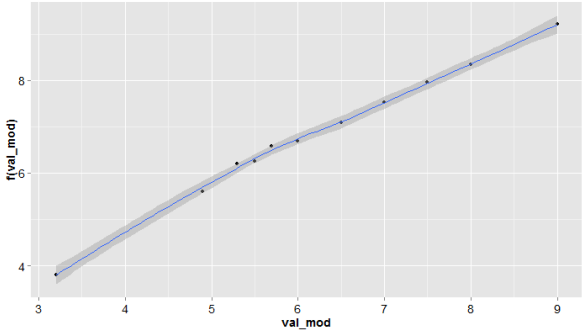


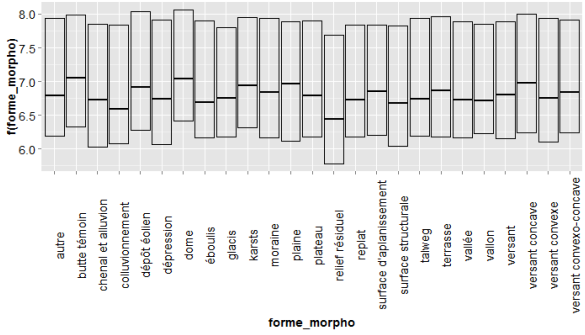


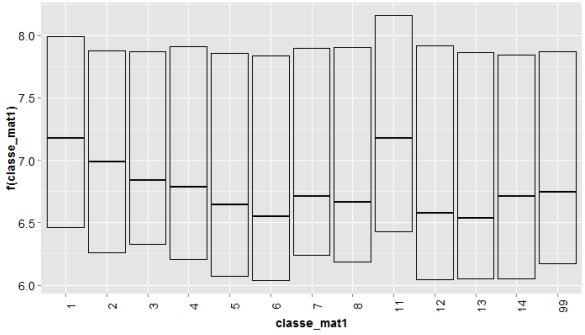


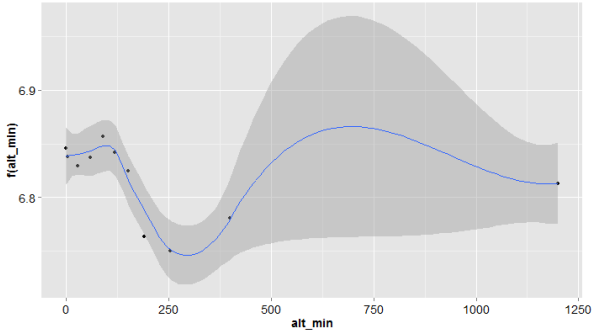


- Val_max, GBM :

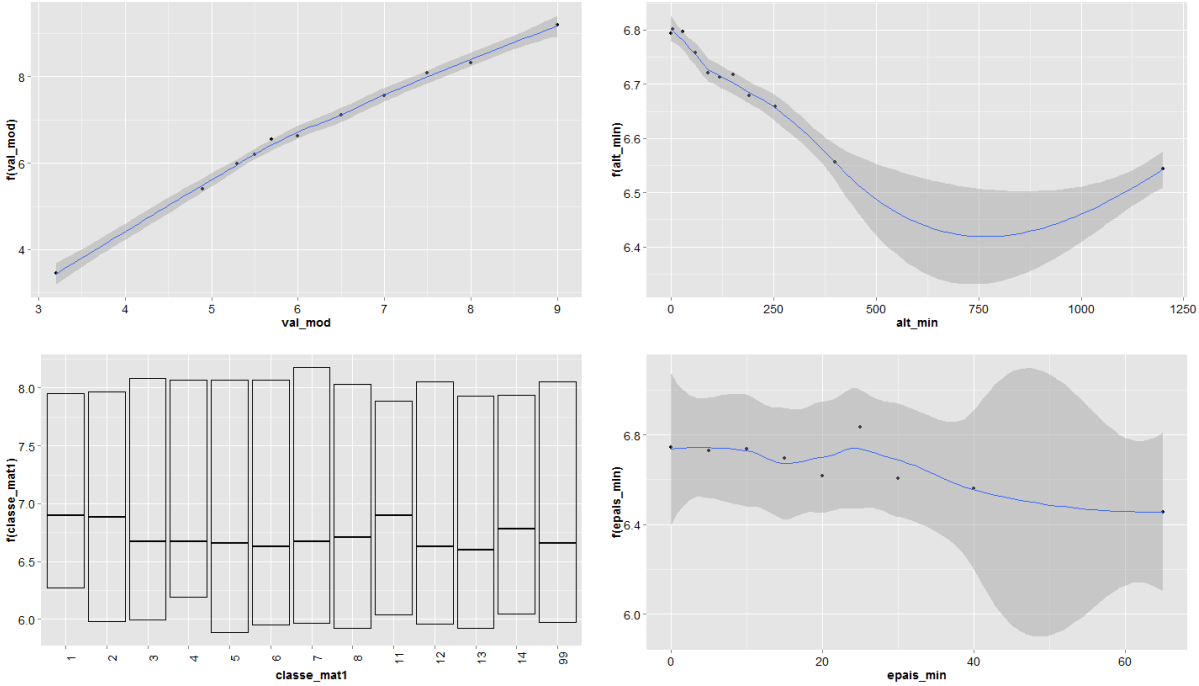




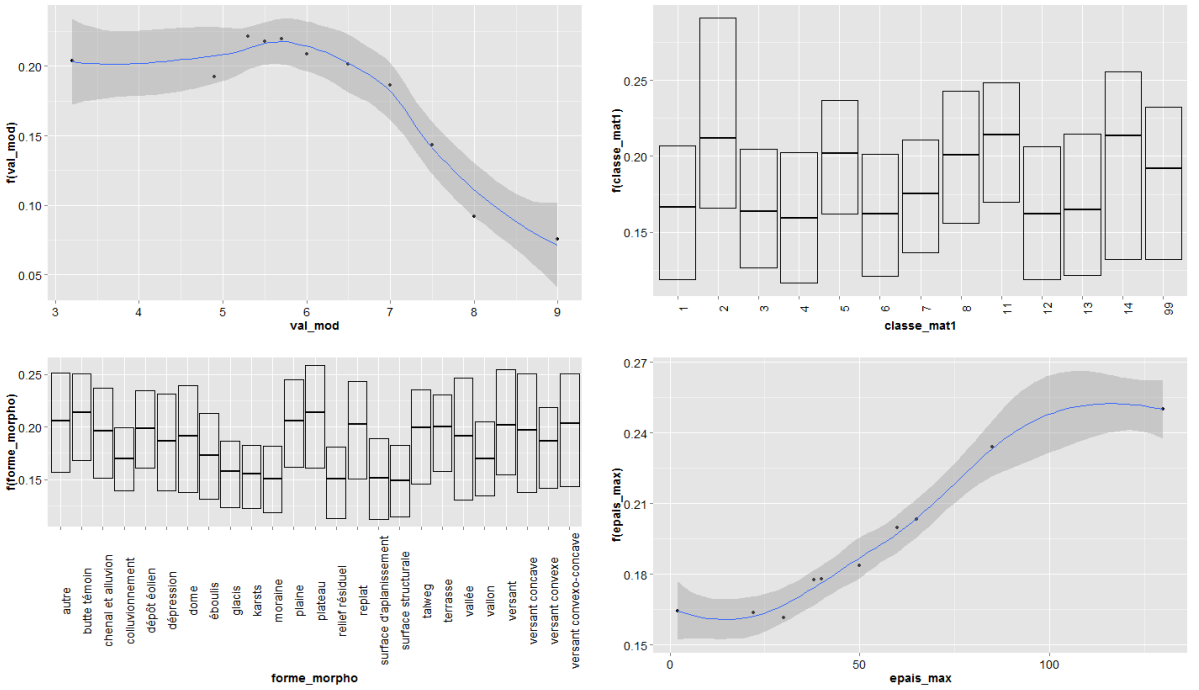




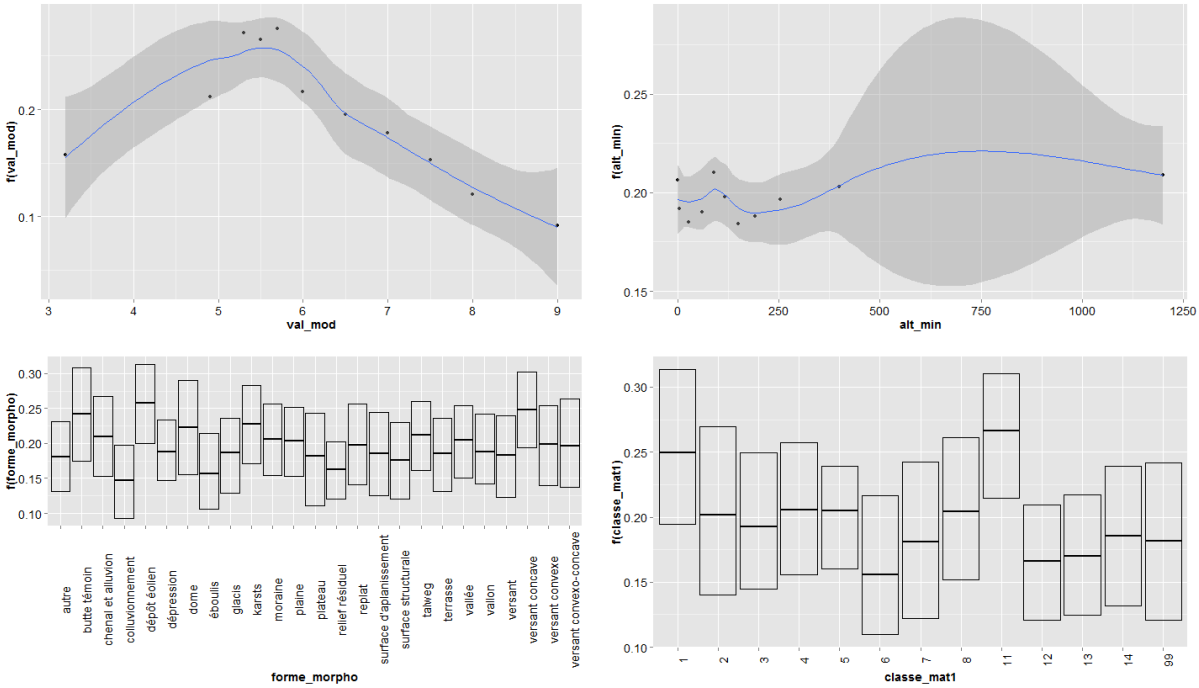
- Val_max, Cubist :



- Etendue_norm, Random Forest :



- Etendue_norm, GBM :



- Etendue_norm, Cubist :