Généralités sur le Machine Learning

Qu'est ce que le Machine Learning ?

- Le Machine Learning est une branche de l'intelligence artificielle (IA) qui a pour objectif d'analyser et d'interpréter des données et des modèles afin de permettre l'apprentissage, la prédiction et la prise de décision sans interaction humaine.
- Le succès du Machine Learning est dû à des avancées dans différents domaines :
  (1) Disponibilité des données : automatisation, digitalisation, applications web.
  (2) Puissance informatique : accroissement considérable (loi de Moore).
  (3) Algorithme d'apprentissage : avancées théoriques et méthodologiques pour l'apprentissage.

Le Machine Learning a été développé principalement par la communauté informatique.
Le principal langage de programmation est Python et la librairie `scikit-learn`.

Qu'est-ce que l'apprentissage statistique ?

- L'apprentissage statistique fait référence à un ensemble d'outis pour modéliser et comprendre les jeux de données complexes.
  - Hastie, Tibshirani & Friedman, *The Elements of Statistical Learning*, 2001, Springer.
- C'est une branche des statistiques qui s'est développée parallèlement au Machine Learning et qui met l'accent sur la modélisation, l'estimation d'erreur et les garanties théoriques.

L'apprentissage statistique a été développé par la communauté statistique, le principal langage de programmation est R, de nombreux packages ont été développés et notamment le package `caret` qui permet la comparaison de nombreuses méthodes.

Apprentissage supervisé et apprentissage non supervisé

Deux principales classes de problèmes en apprentissage statistique :

- Apprentissage supervisé :
  But : prédire une variable réponse Y (output) à partir de covariables observées X (input) en déterminant une relation $Y = f(X)$.
  Data : échantillon d'apprentissage $(X_i, Y_i)_{1 \leq i \leq n}$

- Apprentissage non supervisé :
  But : découvrir une structure dans le jeu de données (visualisation, clusters, réduction de dimension).
  Data : échantillon d'apprentissage $(X_i)_{1 \leq i \leq n}$

Le premier cas est mieux posé et ce cours porte principalement sur l'apprentissage supervisé et la prédiction.

Apprentissage supervisé : classification et régression

Deux classes principales de problèmes en apprentissage supervisé :

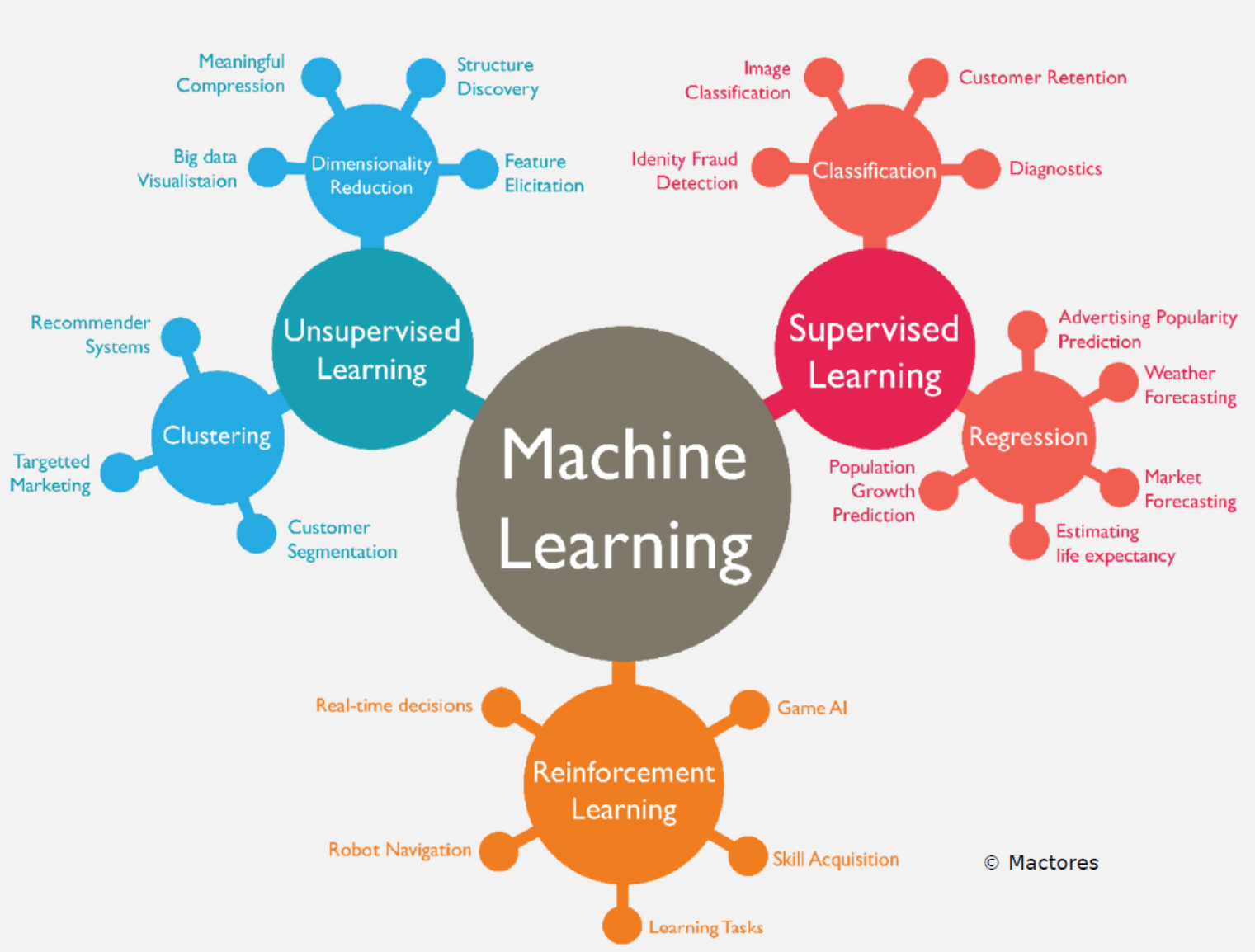
- Régression : la réponse Y est quantitative.
  - prédire le salaire Y d'un joueur de baseball en fonctions de ses caractéristiques X (age, club, points marqués à la dernière saison...)
  - prédire le prix Y d'une voiture en fonctions de ses caractéristiques X (puissance, marque ...)

- Classification : la réponse Y est qualitative.
  - prédire si un e-mail est du spam ou non (classification binaire) ;
  - prédire si une tumeur est bénine, maline de type I, II ou III (classification multitype)
  - reconnaître des chiffres écrits à la main de 0 à 9 (classification multitype)

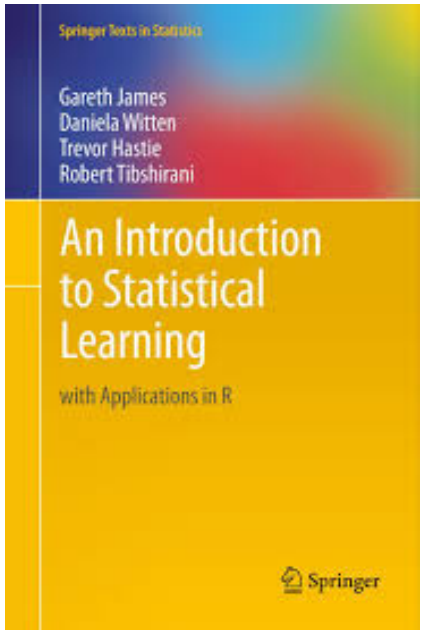
Terminologie – les termes utilisés diffèrent en statistique et Machine Learning :

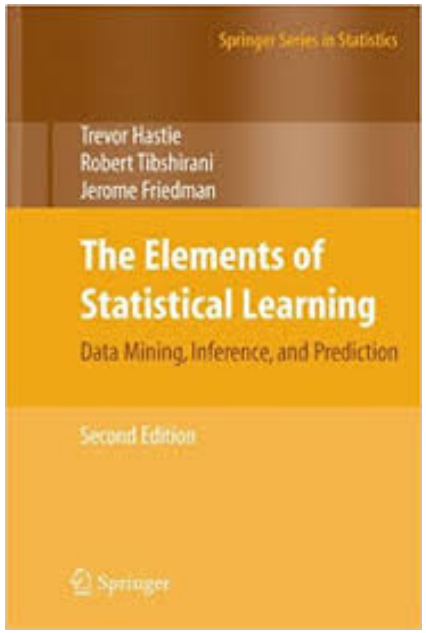
- Y : réponse, variable expliquée, output, label.
- X : covariable, variable explicative, prédicteur, input, feature.

Machine Learning : une vue d'ensemble



Références

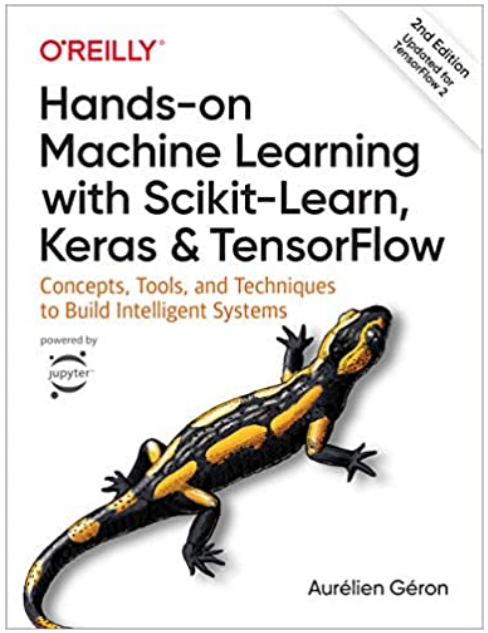




[ISL] James, Witten, Hastie & Tibshirani. *An Introduction to Statistical Learning*, Springer.
Version gratuite `https://www.statlearning.com/`

[ESL] Hastie, Tibshirani & Friedman. *The Elements of Statistical Learning*, Springer.
Version gratuite `http://web.stanford.edu/~hastie/ElemStatLearn/`

Bibliography



- Géron, *Hand-on machine learning with Scikit-Learn, Keras & TensorFlow*, O'Reilly. `https://github.com/ageron/handson-ml2`

Quelques éléments théoriques pour la prédiction en régression

Cadre théorique de la régression

- On s'intéresse à prédire une variable quantitative Y en fonction de p variables explicatives $X_1, \ldots, X_p$.

  On note $X = (X_1, \ldots, X_p) \in \mathbb{R}^p$.
  On note $(X, Y) \in \mathbb{R}^p \times \mathbb{R}$ une réalisation générique de ces variables.

- On observe les données $(x_1, y_1), \ldots, (x_n, y_n)$ sur un échantillon de taille n.

- L'objectif est d'utiliser les données pour construire une fonction de prédiction $\hat{f}_n$, appelée modèle, permettant de prédire Y lorsque l'on connait X.

  L'échantillon $(x_i, y_i)_{1 \leq i \leq n}$ est appelé échantillon d'apprentissage.
  On note souvent $\hat{y} = \hat{f}_n(x)$ la prédiction en x.

Erreur de prédiction

- L'erreur de prédiction est le plus souvent mesurée grâce à l'erreur quadratique

$$(y - \hat{y})^2 = (y - \hat{f}_n(x))^2.$$

où y est la réalisation et $\hat{y} = \hat{f}_n(x)$ la prédiction .

- L'erreur quadratique moyenne sur l'échantillon d'apprentissage s'écrit alors

$$\text{err}_{train} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{f}_n(X_i))^2.$$

- Il est courant que les algorithmes d'apprentissage cherchent à minimiser cette erreur pour construire $\hat{f}_n$.
(estimation des moindres carrés).

Exemple : régression linéaire

- Le modèle le plus simple est le modèle linéaire où la prédiction se fait par une fonction linéaire
$$f(x) = \alpha_0 + \alpha_1 x_1 + \cdots + \alpha_p x_p.$$
- Les coefficients de régressions $\hat{\alpha}_0, \ldots, \hat{\alpha}_n$ sont déterminés de manière à minimiser l'erreur quadratique
$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \alpha_0 - \alpha_1 X_{i1} - \ldots - \alpha_p X_{ip})^2$$
et on obtient
$$\hat{f}_n(x) = \hat{\alpha}_0 + \hat{\alpha}_1 x_1 + \cdots + \hat{\alpha}_p x_p.$$
- Lorsque $n > p$, il existe en général une unique solution $(\hat{\alpha}_0, \ldots, \hat{\alpha}_p)$ qui admet une forme mathématique simple.

Erreur de généralisation

- Une prédiction correcte sur l'échantillon d'apprentissage ne garantit pas une prédiction correcte sur d'autres observations que le modèle n'a pas vu.
  C'est la question de la capacité de généralisation du modèle au delà des données d'apprentissage.

- L'erreur de généralisation est définie theoriquement par

$$\mathbb{E}[(Y - \hat{f}_n(X))^2]$$

  où (X, Y) est un élément générique indépendant de l'échantillon d'apprentissage.

- Dans la pratique, cette erreur de généralisation est estimée sur de nouvelles données, appelées données test.

  On se donne un échantillon test noté $(x'_1, y'_1), \ldots, (x'_m, y'_m)$ et on calcule

$$\text{err}_{test} = \frac{1}{m} \sum_{i=1}^{n} (y'_i - \hat{f}_n(x'_i))^2.$$

Erreur d'apprentissage et erreur test

Erreur d'apprentissage

- Échantillon d'apprentissage $(x_i, y_i)_{1 \leq i \leq n}$.
- Ajustement du modèle $\hat{f}_n$ sur l'échantillon d'apprentissage.
- Calcul de l'erreur sur le même échantillon

$$\text{err}_{train} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}_n(x_i))^2.$$

$\rightsquigarrow$ L'erreur d'apprentissage est trop optimiste car elle ne prend pas en compte la généralisation à de nouvelles données.

Erreur test

- Échantillon d'apprentissage $(x_i, y_i)_{1 \leq i \leq n}$. et échantillon test $(x'_i, y'_i)_{1 \leq i \leq m}$.
- Ajustement du modèle $\hat{f}_n$ sur l'échantillon d'apprentissage.
- Calcul de l'erreur sur l'échantillon test

$$\text{err}_{test} = \frac{1}{m} \sum_{i=1}^{m} (y'_i - \hat{f}_n(x'_i))^2.$$

$\rightsquigarrow$ L'erreur test est une bonne mesure pour l'erreur de prédiction sur des nouvelles données.

MISE EN PRATIQUE : erreur d'apprentissage et erreur test en régression polynomiale (jupyter notebook)

Erreur d'apprentissage et erreur test

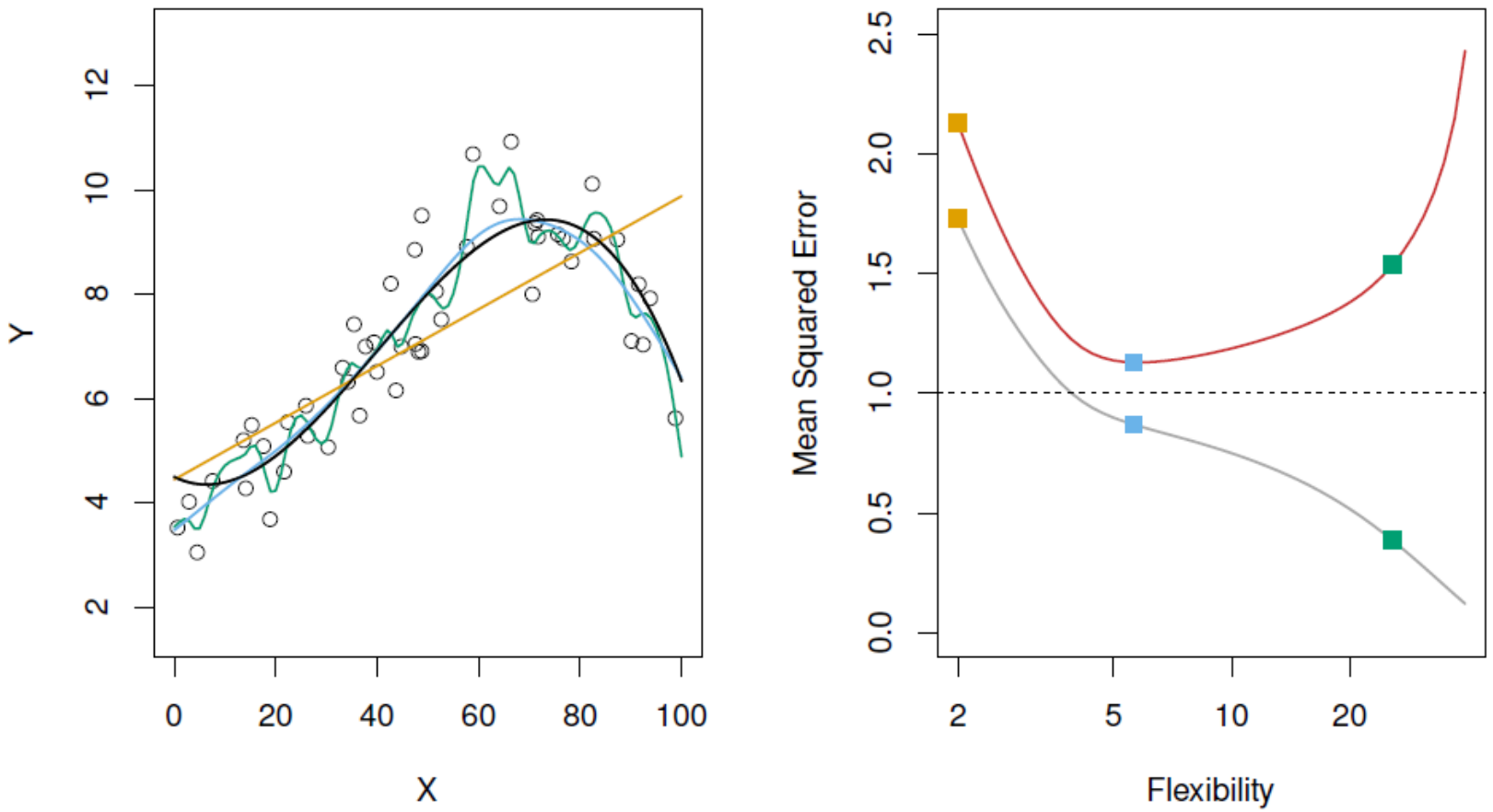


Figure – A gauche : données simulées selon le modèle $Y = f(X) + \varepsilon$ avec f représentée en noir, ainsi que trois estimateurs polynomiaux : ajustement linéaire (orange) et de degré plus élevé (bleu et vert).
A droite : erreur sur l'échantillon d'apprentissage (err_{train} en gris) et sur l'échantillon test (err_{test} en bleu) pour les trois modèles.

Décomposition biais/variance

- On suppose

$$Y = f(X) + \varepsilon$$

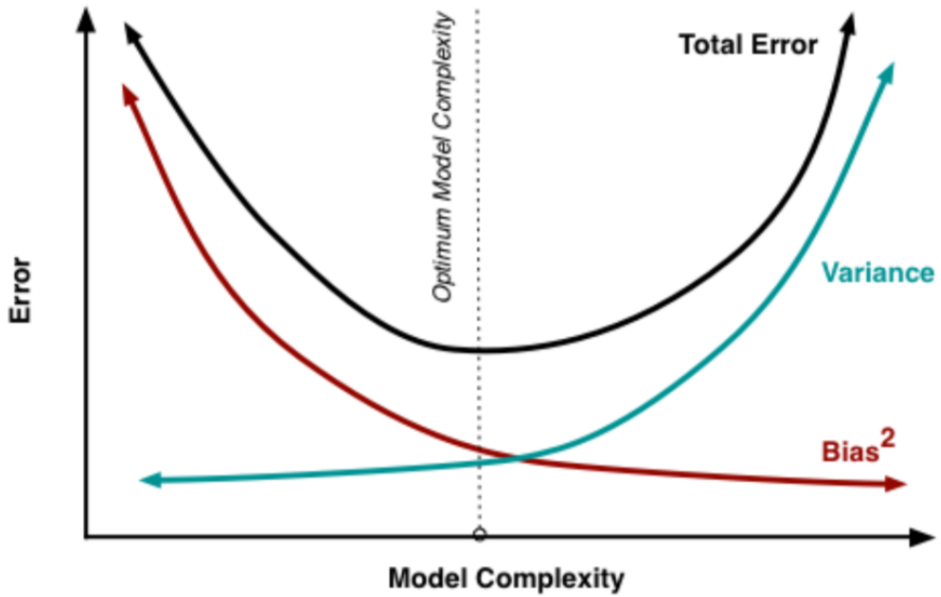
avec l'erreur ε indépendante de X vérifiant $\mathbb{E}[\varepsilon] = 0$ et $\mathrm{Var}[\varepsilon] = \sigma^2$.

- On peut montrer que l'erreur de prédiction se décompose en

$$\begin{aligned}\mathbb{E}[(Y - \hat{f}_n(x))^2 | X = x] &= (\mathbb{E}[\hat{f}_n(x)] - f(x))^2 + \mathrm{Var}[\hat{f}_n(x)] + \sigma^2 \\ &= \mathrm{Bias}^2(\hat{f}_n(x)) + \mathrm{Var}(\hat{f}_n(x)) + \sigma^2 \\ &= \text{biais carré + variance + erreur irréductible}\end{aligned}$$

  - L'erreur irréductible est due au bruit ε et ne peut pas être améliorée.
  - Le biais mesure la différence entre la prédiction moyenne $\mathbb{E}(\hat{f}_n(x))$ et la cible $f(x)$.
  - La variance mesure la variabilité de la prédiction $\hat{f}_n(x)$.

Le compromis biais/variance



- En général, lorsque la complexité du modèle augmente, le carré du biais diminue mais la variance augmente.
- La somme des deux à généralement un minimum, qui représente le bon compromis biais/variance.

Sur- et sous-apprentissage

Interprétation :

- Un modèle simple est généralement robuste et bien estimé (faible variance), mais il peut ne pas être suffisamment riche pour représenter la fonction de régression (biais élevé)
  $\rightsquigarrow$ **sous-apprentissage**

- Un modèle complexe peut bien approximer la vraie fonction de régression (faible biais) mais son estimation peut être peu robuste et trop coller aux données (variance élevée)
  $\rightsquigarrow$ **sur-apprentissage**

compromis biais/variance $\Leftrightarrow$ éviter sous/sur- apprentissage

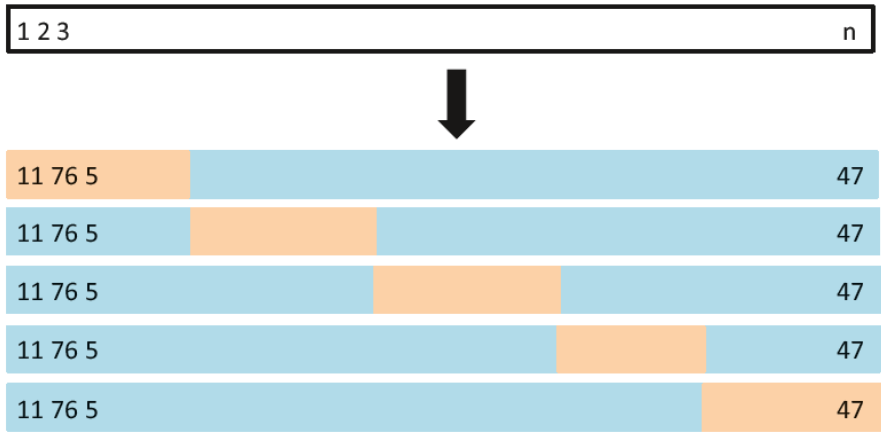
Validation simple

- La complexité du modèle dépend en général d'un hyper-paramètre qu'il faut choisir avec soin de manière à minimiser l'erreur de prédiction :
  - degré d'une régression polynomiale,
  - nombre de feuilles dans un arbre,
  - nombre d'itérations en boosting...
- Pour bien choisir l'hyper-paramètre :
  - on estime l'erreur de prédiction sur un échantillon test pour différentes valeurs de l'hyperparamètre,
  - on sélectionne la valeur minimisant l'erreur de prédiction.

  C'est la méthode de validation simple.
- Mais celle-ci a certains inconvénients :
  - beaucoup de données sont nécessaires car seulement une fraction est utilisée pour l'entrainement et une autre pour l'estimation de l'erreur ;
  - les résultats dépendent de la division en deux échantillons apprentissage/test ; une autre division donnera une autre estimation de l'erreur ;
  - la variabilité par rapport à l'échantillon d'apprentissage n'est pas vraiment considérée.

  $\rightsquigarrow$ La validation croisée (cross-validation) résoud ces inconvénients.

Validation croisée



- On divise l'échantillon en $K = 5$ (choix par défaut) groupes de tailles approximativement égales. Chaque groupe joue une fois le rôle d'échantillon test. Les groupes sont notés $C_1, \ldots, C_K$.
- Pour k variant de 1 à K, on traite C_k comme échantillon test et on ajuste le modèle sur les données restantes obtenant $\hat{f}_{-k}$; l'erreur de prédiction sur C_k est alors

$$\mathrm{err}_{C_k} = \frac{1}{\#C_k} \sum_{i \in C_k} (y_i - \hat{f}_{-k}(x_i))^2$$

- L'estimation d'erreur par validation croisée est alors

$$\mathrm{err}_{CV} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in C_k} (y_i - \hat{f}_{-k}(x_i))^2 = \sum_{k=1}^{K} \frac{\#C_k}{n} \mathrm{err}_{C_k}.$$

Utilisation de `scikit-learn`

Les fonctions suivantes `scikit-learn` sont utiles pour évaluer un modèle :

- fonctions scoring dans `sklearn.metrics` :
  `mean_squared_error`, `rooted_mean_squared_error`, `mean_absolute_error`, etc.
- validation simple et croisée (dans `sklearn.model_selection`) :
  `train_test_split`, `score`, `cross_val_score`, etc..
- automatisation de la recherche (dans `sklearn.model_selection`) :
  `GridSearchCV`

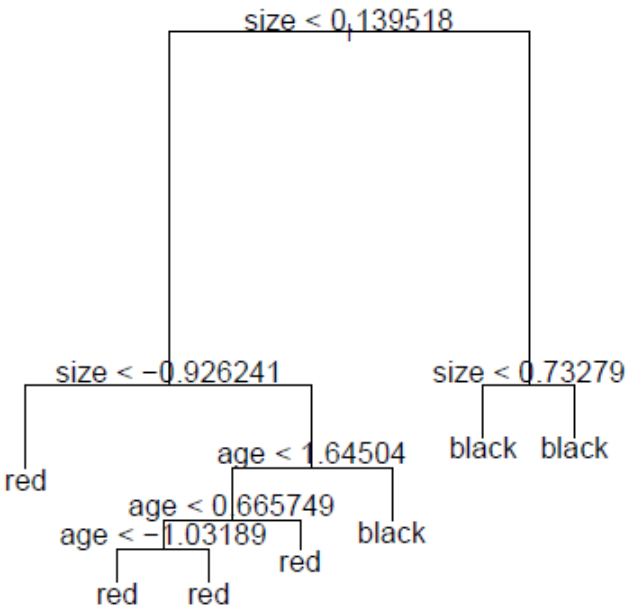
Plus de détail dans l'aide de Scikit-learn :
`https://scikit-learn.org/stable/model_selection.html`

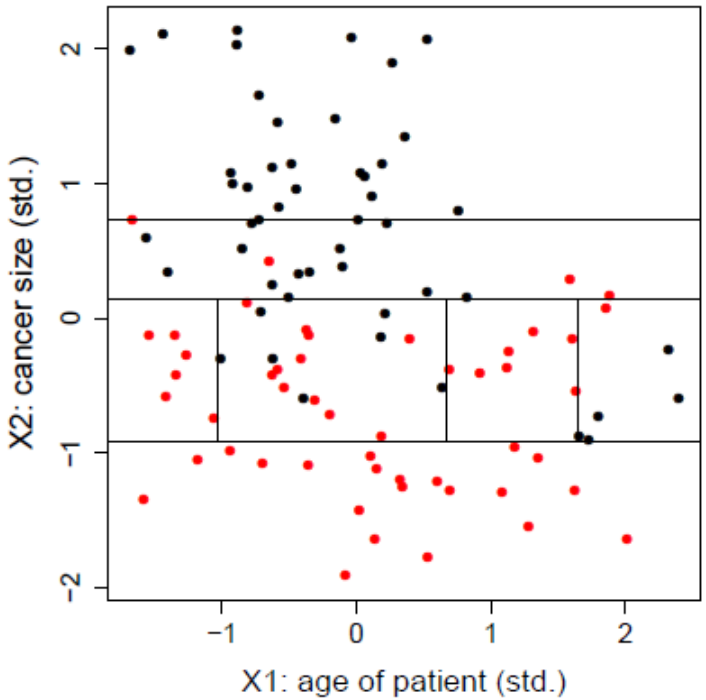
Arbres de régression

Arbres de décision

Classification And Regression Tree (CART, Breiman 1984)

- Les arbres CART peuvent être utilisés en classification ou en régression.
- En utilisant des test binaires successifs, le modèle définit une partition de l'échantillon d'apprentissage en différents groupes homogènes ainsi qu'une partition de l'espace des covariables en différentes régions.
- Pour un point $X = x$, on peut déterminer le groupe associé et, dans le cas de la régression, utiliser la moyenne de Y sur ce groupe pour faire la prédiction.

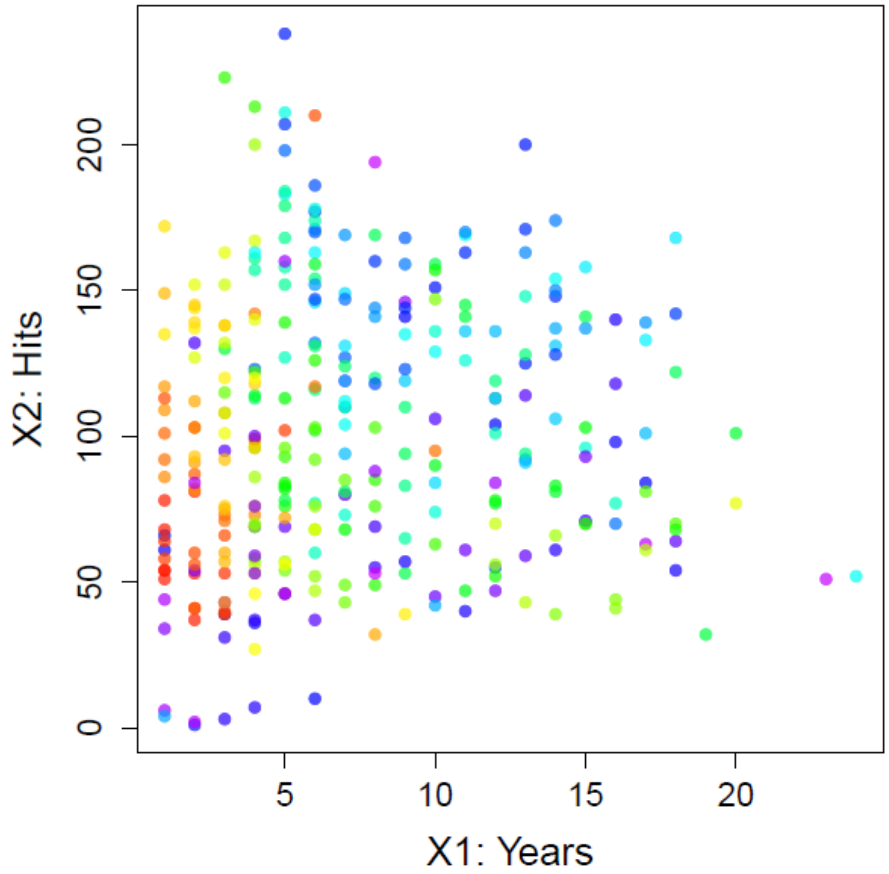




Exemple simple en régression (jeu de données Hitters)

Le jeu de données `Hitters` (livre et package ISLR) contient le salaire annuel `Salary` pour $n = 263$ joueurs de baseball et $p = 19$ covariables telles que l'ancienneté `Years` en première ligue et le nombres de points marqués `Hits` à la dernière saison.

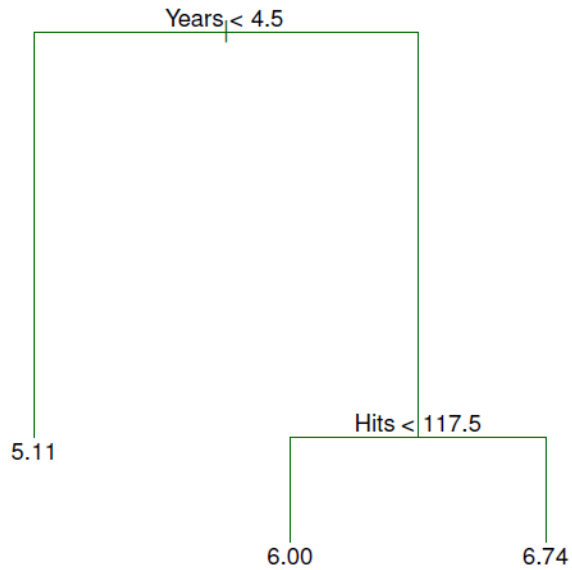
Code couleur pour les salaires : bas (jaune-rouge) à élevé (bleu-vert).



Exemple simple en régression (jeu de données Hitters)

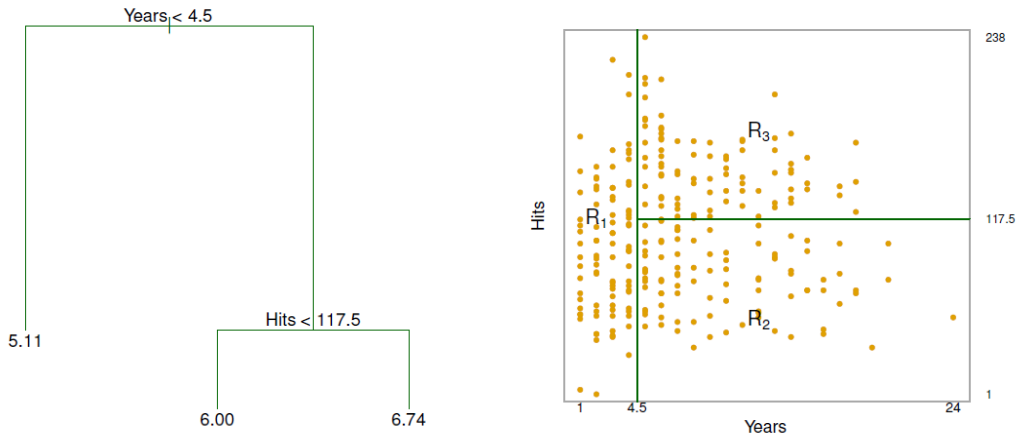
Arbre de régression pour prédire le logarithme du `Salary` d'un joueur basé sur les variables `Years` et `Hits` comme prédicteurs.

- Chaque nœud interne de l'arbre correspond à une division de l'échantillon.
- La division $(X_j < t)$ indique la branche gauche tandis que la branche droite correspond à $(X_j \geq t)$.
- Les nœuds terminaux sont appelés feuilles et indiquent une valeur prédite égale à la moyenne des observation de Y dans la feuille.



Un avantage des arbres est leur interprétabilité : la structure de la prédiction est simple avec des tests successifs et très facile à expliquer à des non spécialistes.

Exemple simple en régression (jeu de données Hitters)



- L'arbre fournit une partition de l'espace des covariables en trois regions correspondant aux feuilles de l'arbre.
- Chaque région est un rectangle : ici

$$R_1 = \{X \ : \ X_1 < 4.5\}, \ R_2 = \{X \ : \ X_1 \geq 4.5, X_2 < 117.5\}, R_3 = \{X \ : \ X_1 \geq 4.5, X_2 \geq 117.5\}$$

- Interprétation :
  - `Years` est la covariable avec le plus d'impact sur la variable `Salary` : les joueurs avec le plus d'expérience gagnent plus que d'autres moins expérimentés.
  - Pour un joueur moins expérimenté, le nombre de points `Hits` a peu d'influence sur le `Salary`.
  - Mais parmi les joueurs les plus expérimentes (plus de 5 ans en première ligue), le nombre de points `Hits` influence `Salary`.

Comment construire un arbre de régression ?

Comment construit-on l'arbre de régression à partir de l'échantillon $(x_i, y_i)_{1 \leq i \leq n}$?

- On veut partitionner l'espace des covariables $\mathbb{R}^p$ en J régions rectangulaires distinctes $R_1, \ldots, R_J$.
- Pour une nouvelle observation x tombant dans la région R_j, on prédit la moyenne de la classe R_j :

$$\hat{y} = \bar{y}_{R_j} = \frac{1}{n_j} \sum_{i : x_i \in R_j} y_i,$$

où n_j désigne le nombres d'observations dans R_j.

- Le modèle $x \mapsto \hat{f}_n(x) = \hat{y}$ est donc constant par morceaux et dépend des régions $R_1, \ldots, R_J$.
- Les régions $R_1, \ldots, R_J$ sont construites afin de minimiser l'erreur quadratique

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{f}_n(x_i))^2 = \sum_{j=1}^{J} \sum_{i : x_i \in R_j} (y_i - \bar{y}_{R_j})^2.$$

RSS = Residual Sum of Squares

Comment construire un arbre de régression ?

- Il est impossible d'optimiser la RSS sur l'ensemble des partitions en J régions (phénomène d'explosion combinatoire).
- On utilise un algorithme descendant (top-down) et glouton (greedy) qui fonctionne par partitionnement binaire récursif :
  - L'algorithme débute en haut de l'arbre et partitionne successivement l'espace des covariables, avec chaque noeud divisant en deux le groupe correspondant et produisant deux nouvelles branches de l'arbres.
  - L'algorithme est glouton car chaque pas est fait en optimisant le critère RSS à cette étape particulière, en ignorant ce qui se passera par la suite.
  - L'algorithme est récursif, car chaque étape réitère le même processus de partitionnement récursif.

Partitionnement binaire glouton

(1) Pour chaque covariable j et seuil t, on définit les demi-espaces gauche et droite

$$R_{left}(j,t) = \{x : x_j < t\} \quad \text{and} \quad R_{right}(j,t) = \{x : x_j \geq t\}.$$

(2) On cherche les valeurs j^* and t^* qui minimisent la RSS (residual sum of squares)

$$\text{RSS} = \sum_{i:x_i \in R_{left}(j,t)} (y_i - \bar{y}_{R_{left}(j,t)})^2 + \sum_{i:x_i \in R_{right}(j,t)} (y_i - \bar{y}_{R_{left}(j,t)})^2$$

puis on effectue la partition en $R_{left}(j^*, t^*)$ and $R_{right}(j^*, t^*)$.

(3) On réitère le processus sur chacune des régions obtenues, en cherchant à chaque étape la meilleur variable j^* et le meilleur seuil t^* pour la partition.

(4) On s'arrête lorsqu'un critère d'arrêt est atteint (lié au choix des hyperparamètres).

Les arbres en `scikit-learn`

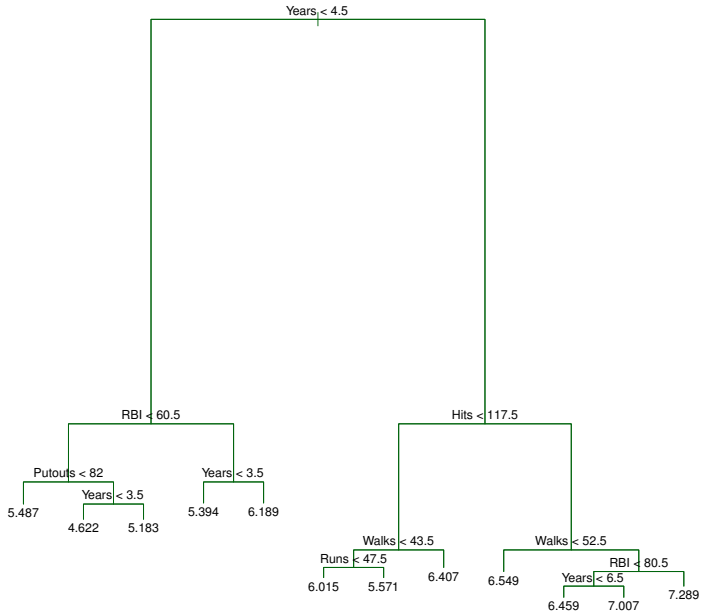
L'implémentation `scikit-learn` prévoit les fonctions `DecisionTreeRegressor()` et `DecisionTreeClassifier()` pour implémenter facilement les arbres de décision en régression ou classification. Les principaux hyperparamètres suivants :

- `max_depth`
- `min_samples_split`
- `min_samples_leaf`

Plus d'information dans l'aide de `scikit-learn` :
`https://scikit-learn.org/stable/modules/tree.html`

Exemple : jeu de données `Hitters`

Sur le jeu de données `Hitters`, arbre obtenu avec 12 feuilles (à gauche), ou arbre plus simple avec 3 feuilles (à droite).



Dans cette représentation, la longueur des branches est proportionnelle à la réduction du critère RSS due à la partition.

Compromis Biais-Variance

- Intuitivement, la complexité de l'arbre et du modèle prédictif associé est lié à son nombre de feuilles :
  - Un modèle complexe, avec de nombreuses feuilles, produit beaucoup de petites régions et est susceptible de produire de bonnes prédictions sur l'échantillon d'apprentissage mais avec un potentiel sur-apprentissage.
  - Inversement, si le nombre de feuilles est trop réduit, la règle de prédiction risque de mal représenter le signal d'où un sous-apprentissage potentiel.
- On résoud le compromis biais/vairance en sélectionnant "bien" les hyperparamètres réglant la complexité du modèle, c'est-à dire par validation simple (sur un échantillon test) ou par validation croisée.

Avantages et inconvénients des arbres

Avantages

- Simplicité : les arbres sont très simples à expliquer, même à des non spécialistes.
- Interprétabilité grâce aux graphiques, les variables les plus importantes ressortent dans les premières divisions.
- Souplesse d'utilisation : gestion des taches de classification et de régression, prise en compte des variables quantitatives et qualitatives ou des valeurs manquantes

Inconvénients

- Les performances prédictives sont généralement limitées.
- Instabilité / manque de robustesse : un petit changement des données peut provoquer un changement important dans l'arbre.

Forêts aléatoires

Bagging

- Le bagging, contraction de bootstrap aggregation, est un principe général permettant de réduire la variance d'une méthode d'apprentissage.
  Il s'applique donc à une modèle ayant un biais faible et une variance élevée.
- L'idée est d'utiliser la méthode de rééchantillonnage bootstrap pour dupliquer artificiellement l'échantillon d'apprentissage et de construire un grand nombre de modèles $\hat{f}_n^{*1}, \ldots, \hat{f}_n^{*B}$.
- Les modèles sont ensuite agrégés en prenant leur moyenne :

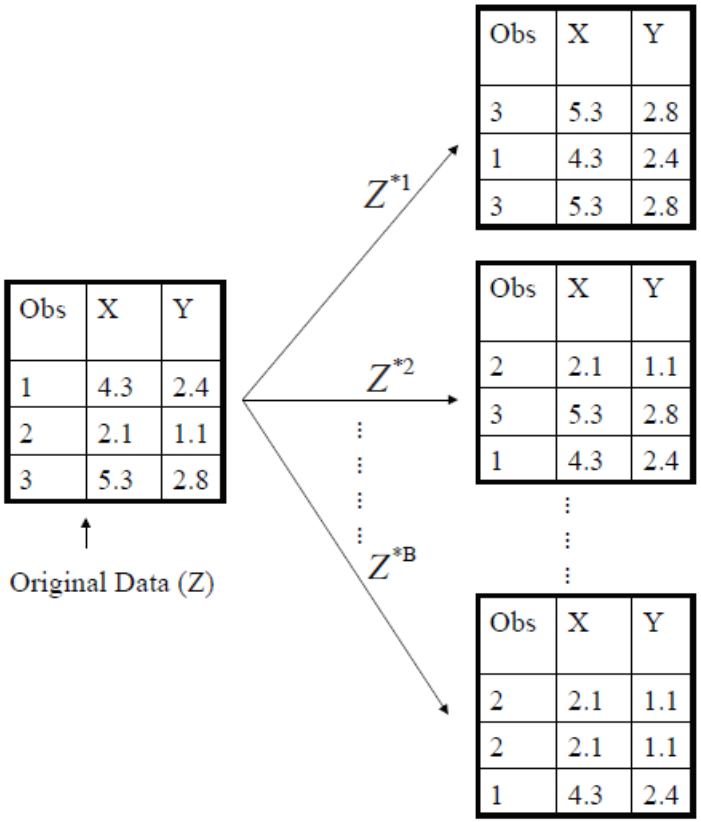
$$\hat{f}_n^{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_n^{*b}(x).$$

Le bootstrap

Soit $Z = \{Z_1, \ldots, Z_n\}$ l'échantillon d'apprentissage avec $Z_i = (X_i, Y_i)$.

On construit B échantillons bootstrap $Z^{*1}, \ldots, Z^{*B}$ par tirage aléatoire :

- tirage de n individus avec remise $(i_1, \ldots, i_n)$ parmi $(1, \ldots, n)$.
- échantillon bootstrap $Z^* = (Z_{i_1}, \ldots, Z_{i_n})$.



Bagging basé sur des arbres

- Les arbres touffus ont un faible biais et une grande variance et sont de bons candidats au bagging.
- Pour chaque échantillon boostrap, on construit un arbre touffu afin d'obtenir un modèle prédictif de faible biais (mais potentiellement grande variance et sur-apprentissage).
- On agrège tous les arbres, ce qui a pour effet de réduire la variance.
- La pratique a montré que les performances en prédiction sont grandement améliorées, mais au pris d'une interprétation plus difficile.
- Important : ici B n'est pas un hyperparamètre classique puisque $B \to \infty$ ne conduit pas au sur-apprentissage ; on a juste une stabilisation des fluctuations d'échantillonnage dues au bootstrap.
  Dans la pratique, le choix par défaut $B = 500$ est classique.

Bagged trees

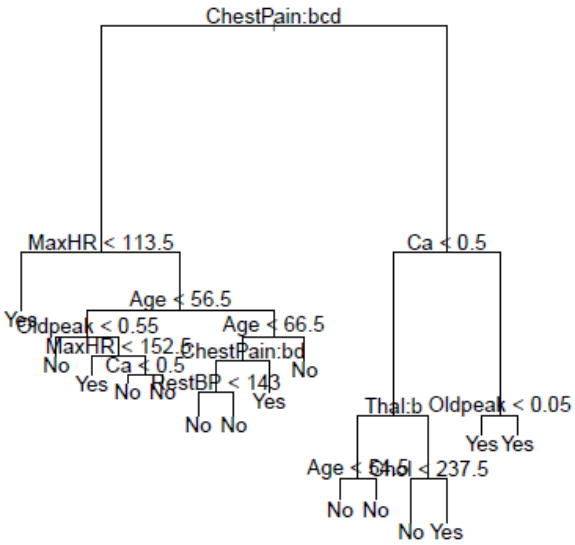
Algorithme Bagged Trees

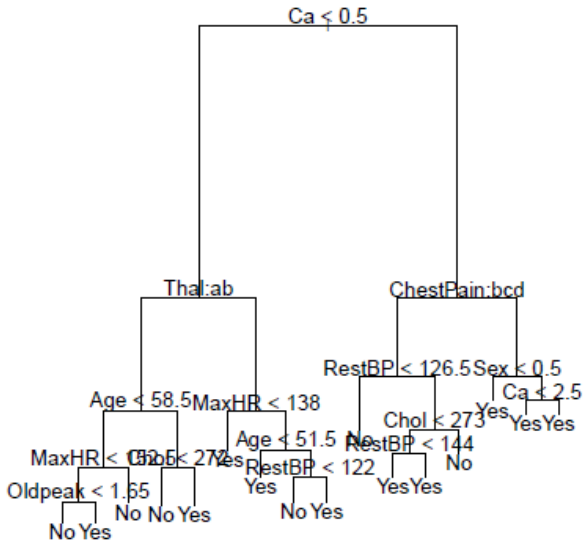
1) Construire B échantillons bootstrap.
2) Pour chaque échantillon boostrap, construire un arbre de régression donnant le prédicteur f^{*b}.
3) Faire la prédiction par aggrégation des B arbres (moyenne) :

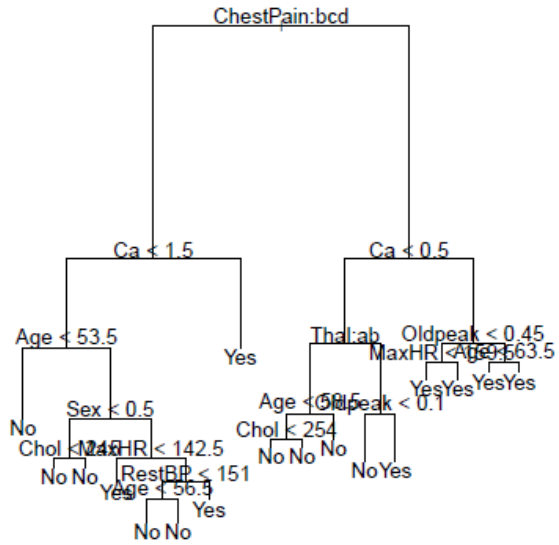
$$\hat{f}_n^{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_n^{*b}(x).$$

Bootstrap et arbres

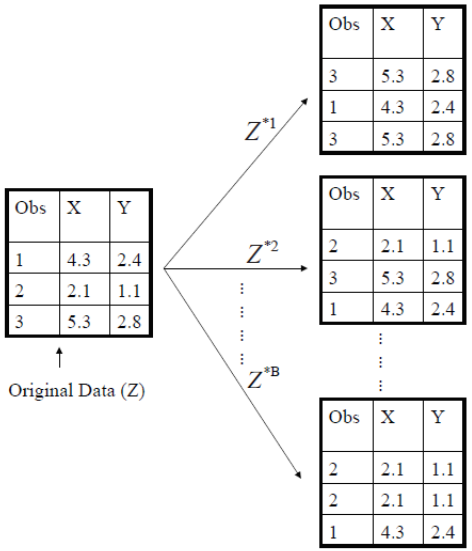
Sur le jeu de données `Heart`.







Observations OOB



- Dans un échantillon bootstrap, du fait du tirage avec remise, certaines observations sont répétées tandis que d'autres sont absentes.
- Les observations absentes sont dites out of bag ou OOB.
- En moyenne, dans un échantillon bootstrap, 63% des observations sont présentes (certaines plusieurs fois) et 37% sont absentes ou OOB.
  En effet, la probabilité de ne pas sélectionner un indice i fixé est

$$\left(1 - \frac{1}{n}\right)^n \longrightarrow e^{-1} \approx 37\%$$

Erreur de prédiction OOB

- La structure du bagging suggère d'utiliser une estimation de l'erreur de prédiction ad-hoc, basée sur les observations OOB.
- Une observation donnée i est OOB pour dans environ 37% des échantillons bootstrap. La prédiction OOB correspondante est

$$\hat{f}_{oob}(x_i) = \frac{1}{|B_i|} \sum_{b \in B_i} \hat{f}^{*b}(x_i) \quad \text{(régression)},$$

avec $B_i = \{b : Z_i \notin Z^{*b}\}$.

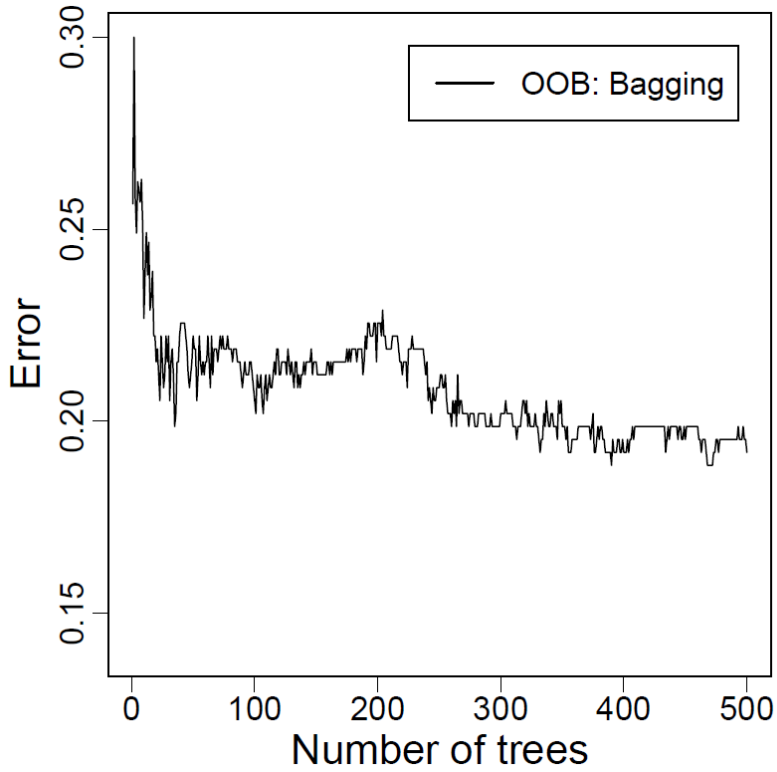
- L'erreur de prédiction OOB est

$$\text{err}_{oob}(\hat{f}_{bag}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}_{oob}(x_i))^2 \quad \text{(régression)}.$$

- L'idée est similaire à la validation croisée mais utilise la structure du bagging et évite de nombreux calculs !

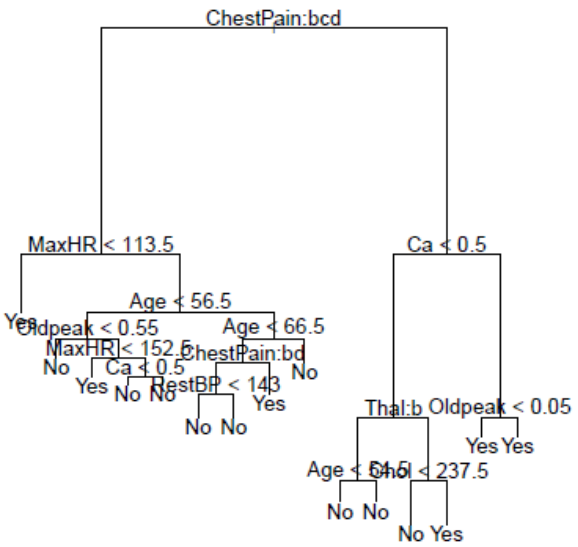
Bagging

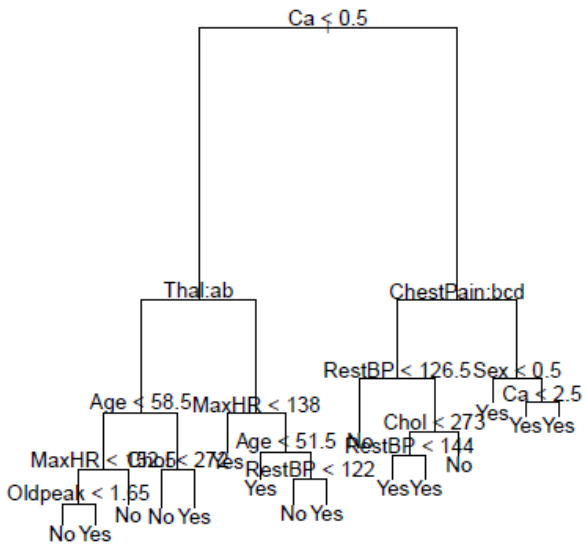
On peut utiliser le package `randomForest` pour implémenter le bagging basé sur des arbres. Le paramètre `mtry` est alors choisi égal à p.

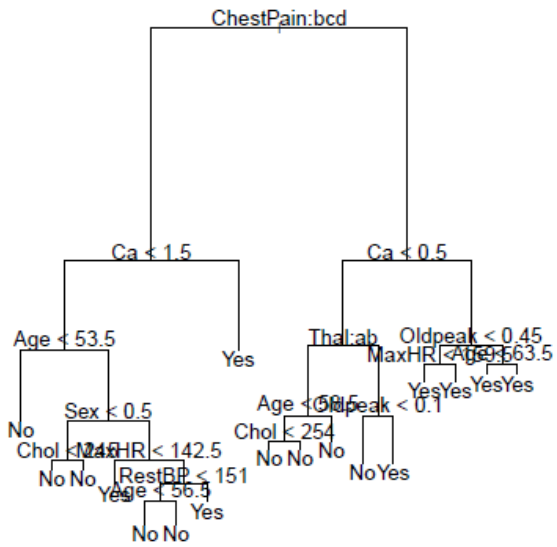


Bagging et corrélation

- Pour le bagging d'arbres, les échantillons bootstrap partagent de nombreuses observations et on observe que les arbres se ressemblent, au moins dans les premiers nœuds où les mêmes prédicteurs apparaissent.
- Intuitivement, cela induit de la corrélation entre les arbres et réduit l'efficacité du bagging.







- Random Forest (Breiman 2001) modifie le bagging d'arbre en ajoutant un aléa supplémentaire dans la construction des arbres afin de réduire la corrélation.

Random Forest

Algorithme Random Forest

1) Construire B échantillons bootstrap.
2) Pour chaque échantillon boostrap, construire un arbre randomisé ainsi :
   à chaque partitionnement binaire, tirer au hasard m parmi p variables puis considérer la recherche de la meilleure partition dans ce sous-ensemble seulement.
3) Faire la prédiction par aggrégation des B arbres (moyenne).

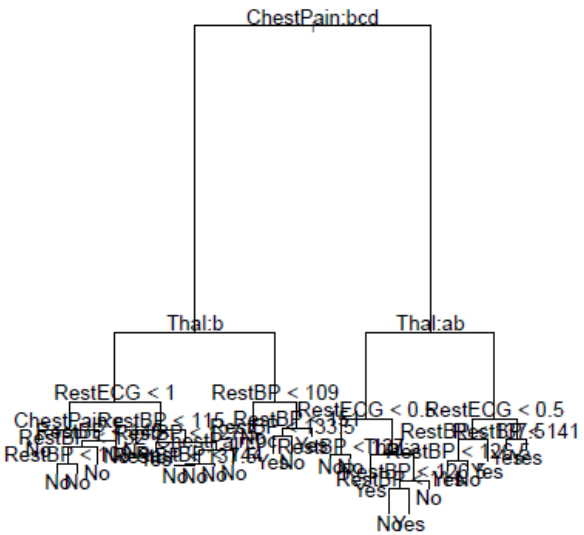
Remarque : les points 1) et 3) sont les mêmes que dans le bagging.

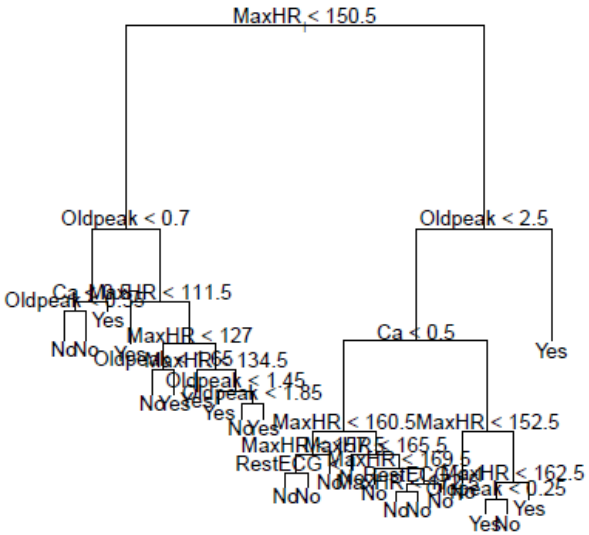
Remarque : dans le point 2), lorsque $m = p$ on retrouve les arbres CART habituels. Lorsque $m < p$, on a des arbres randomisés, plus rapides à calculer car on teste moins de variables pour chaque nœud.

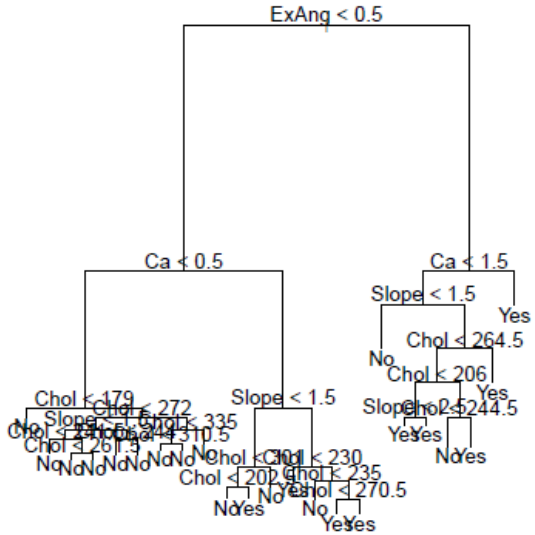
Choix de m : des études empiriques on montré que le choix par défaut $m \approx p/3$ en regression est raisonnable et donne souvent de meilleurs résultat que Bagged Trees. On peut aussi choisir m par validation croisée.

Random Forest

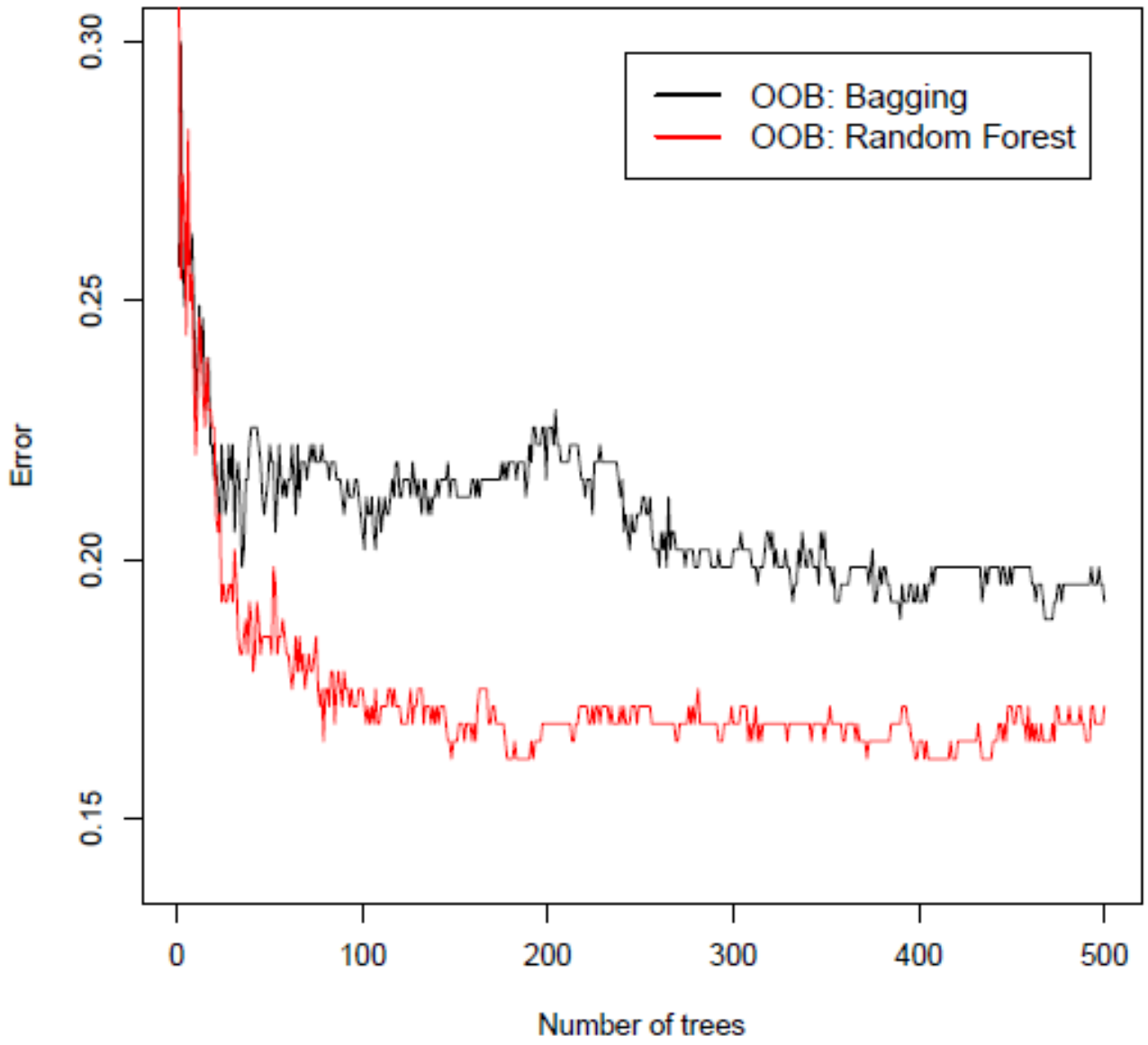
Intuition : le choix aléatoires d'un sous-ensemble de variables évite des arbres trop ressemblants et force la décorrélation. De plus, enlever des variables importantes force l'algorithme à extraire l'information contenue dans les variables moins importantes.





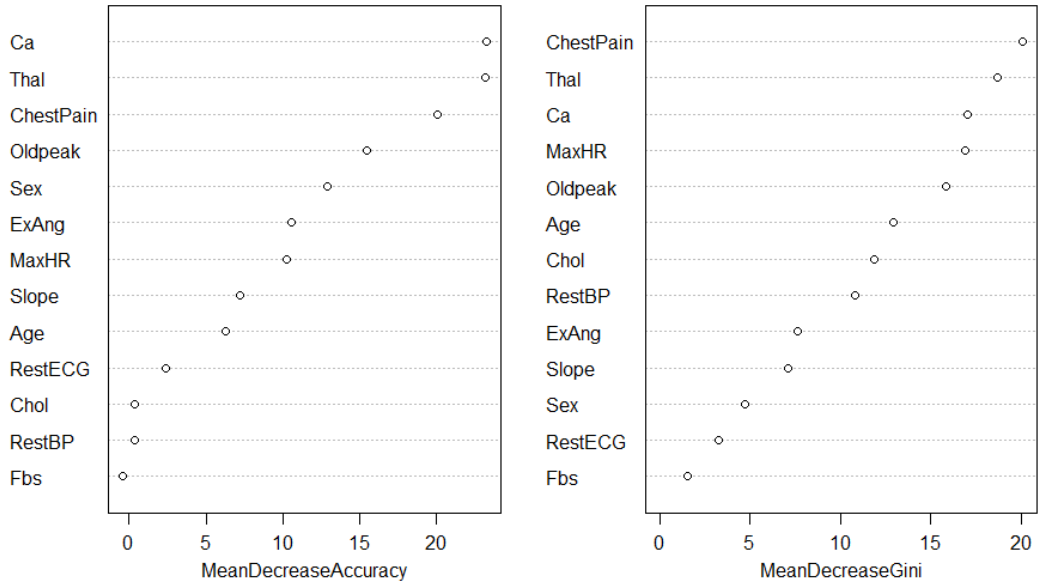


Jeu de données `Heart` : bagging et Random Forest



Mesure d'importance des variables

- BaggedTrees et RandomForest offrent de bien meilleures performance en prédiction qu'un arbre unique, mais au prix d'un modèle beaucoup plus difficile à interpréter. Les mesures d'importance des variables donnent une information simple et pertinente pour l'interprétation.
- Deux types de mesures, avec des résultats souvent grossièrement concordants :
  - Importance basée sur la réduction du critère RSS : pour chaque variable, on somme les contributions à la réduction du critère dûes aux nœuds impliquant la variable en question.
  - Importance basée sur la puissance prédictive : pour chaque variable, on évalue comment l'erreur de prédiction OOB augmente lorsque cette variable est perturbée par permutation.



RandomForest en `scikit-learn`

L'implémentation `scikit-learn` prévoit les fonctions `RandomForestRegressor()` et `RandomForestClassifier()` pour implémenter facilement les arbres de décision en régression ou classification. Les principaux hyperparamètres sont les suivants :

- `n_estimators` : nb d'arbres (100 par défaut, souvent à laisser) ;
- `max_features` : nb de variables testées à chaque split ;
- `oob_score` : True or False, pour le calcul de l'erreur oob (False par défaut) ;
- tous les paramètres hérités des arbres : `max_depth`, etc.

Plus d'information dans l'aide de `scikit-learn` :
`https://scikit-learn.org/stable/modules/ensemble.html`

Quelques éléments théoriques pour la prédiction en classification

Cadre théorique de la classification

- On veut prédire une variable qualitative Y à K modalités notée $\{0, 1, \ldots, K-1\}$ en fonction de p variables explicatives regroupées dans un vecteur $X = (X_1, \ldots, X_p)$.

  On note $(X, Y) \in \mathbb{R}^p \times \{0, \ldots, K-1\}$ une réalisation générique de ces variables.

- L'objectif est d'utiliser un échantillon d'apprentissage $(x_1, y_1), \ldots, (x_n, y_n)$ pour construire une fonction de prédiction $\hat{f}_n$ à valeurs dans $\{0, \ldots, K-1\}$ permettant de prédire Y lorsque l'on connait X.

- Une métrique importante est le taux d'erreur de classification

$$\text{err} = \frac{1}{n} \sum_{i=1}^{n} 1_{\{\hat{f}(x_i) \neq y_i\}}$$

  qui est évalué ici sur l'échantillon d'apprentissage mais sera évalué sur un échantillon test ou par validation croisée pour évaluer la capacité de généralisation du modèle.

- On se limite ici à la classification binaire ($K = 2$), mais des adaptations existent pour la classification multitype ($K \geq 3$).

Matrice de confusion

- La matrice de confusion s'obtient en comparant observations et prédictions :

	$Y = 1$	$Y = 0$
$\hat{Y} = 1$	TP = # true positives	FP= # false positives
$\hat{Y} = 0$	FN = # false negatives	TN = # true negatives

  Elle se calcule de préférence sur un échantillon test ou par validation croisée.

- Le taux d'erreur est

$$\text{err} = \frac{\text{FN+FP}}{\text{TN+FP+FN+FP}}$$

- La sensibilité (true positive rate) et la spécificité (true negative rate) sont définis respectivement par

$$\text{TPR} = \frac{\text{TP}}{\text{TP+FN}} \quad \text{and} \quad \text{TNR} = \frac{\text{TN}}{\text{TN+FP}}.$$

  Ce sont des estimations de $\mathbb{P}(\hat{Y} = 1 \mid Y = 1)$ et $\mathbb{P}(\hat{Y} = 0 \mid Y = 0)$ respectivement.

Autres métriques classiques

D'autres quantités se déduisent de la matrice de précision :

- Positive predictive value (precision) :

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP+FP}} \quad \text{estimateur de } \mathbb{P}(Y = 1 \mid \hat{Y} = 1).$$

- Negative predictive value :

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN+FN}} \quad \text{estimateur de } \mathbb{P}(Y = 0 \mid \hat{Y} = 0).$$

- False discovery rate :

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP+TP}} = 1 - \mathrm{PPV} \quad \text{estimateur de } \mathbb{P}(Y = 0 \mid \hat{Y} = 1).$$

- False omission rate :

$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN+TN}} = 1 - \mathrm{NPV} \quad \text{estimateur de } \mathbb{P}(Y = 1 \mid \hat{Y} = 0).$$

- F_1-score :

$$F_1 = 2\Big(\frac{1}{\mathrm{TPR}} + \frac{1}{\mathrm{PPV}}\Big)^{-1}$$

(moyenne harmonique de la sensibilité et de la précision).

Estimation de la probabilité conditionnelle

- Plutôt que de prédire uniquement une classe $\hat{Y} = 0$ ou 1, il est plus informatif de s'intéresser à la probabilité conditionnelle

$$p(x) = \mathbb{P}(Y = 1 \mid X = x)$$

et d'en construire un estimateur $\hat{p}_n(x)$.

- La théorie montre que pour minimiser l'erreur de classification, il faut poser

$$\hat{Y} = \begin{cases} 1 & \text{si } p(x) > 0.5, \\ 0 & \text{si } p(x) \leq 0.5. \end{cases}$$

(prédiction de la classe la plus probable)

- Si on estime la probabilité $\hat{p}_n(x)$, il est alors naturel de prédire

$$\hat{Y} = \begin{cases} 1 & \text{si } \hat{p}_n(x) > 0.5, \\ 0 & \text{si } \hat{p}_n(x) \leq 0.5. \end{cases}$$

Le choix du seuil 0.5 est cependant lié à la symétrie de l'erreur de classification et est souvent discutable dans la pratique.

Choix du seuil et courbe ROC

- La prédiction associée au seuil $u \in (0,1)$ est définie par

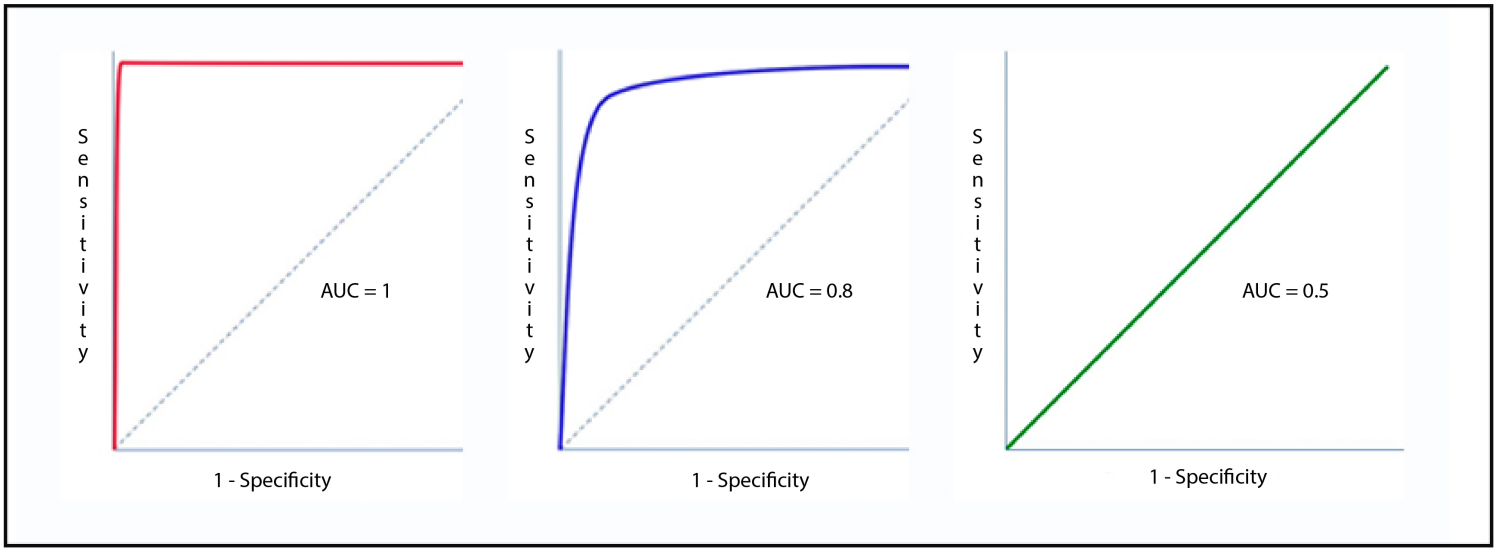
$$\hat{Y} = \begin{cases} 1 & \text{si } \hat{p}_n(x) > u \\ 0 & \text{si } \hat{p}_n(x) \leq u \end{cases}.$$

- Diminuer le seuil u a pour effet d'augmenter la sensibilité mais de diminuer la spécificité.
- La courbe ROC permet un diagnostic de ce dilemme entre sensibilité et spécificité.

Courbe ROC (Receiving Operator Characteristics, terminologie de traitement du signal)

La courbe ROC représente le taux de vrais positifs (sensibilité) en fonction du taux de faux positifs (1-spécificité) lorsque le seuil u varie (courbe paramétrée)

Courbe ROC



- La diagonale correspond à une prédiction aléatoire (pile ou face).
- Le point $(0, 1)$ représente un classifieur ideal :
  - $\mathrm{FPR} = 0$ correspond à l'absence de faux positifs ;
  - $\mathrm{TPR} = 1$ implique l'absence de faux négatif ;

  $\rightsquigarrow$ alors le taux d'erreur est égal à 0.
- Un classifieur est meilleur qu'un autre si sa courbe ROC est toujours "au dessus" ;
- L'aire sous la courbe résume la qualité de la courbe ROC.
  AUC = Area Under Curve

Fonction de perte pour estimer une probabilité

- Pour estimer la probabilité conditionnelle, on utilise de manière privilégiée la fonction de perte définie par

$$L(y, f(x)) = -y \log f(x) - (1-y) \log(1-f(x)), \quad y \in \{0,1\}, f(x) \in [0,1].$$

liée à l'opposé de la log-vraisemblance (déviance).

Proposition

A une fonction de prédiction $f : \mathbb{R}^p \to [0,1]$, on associe la perte moyenne $\mathbb{E}[L(Y, f(X))]$. Cette perte moyenne est minimale lorsque $f(x) = \mathbb{P}(Y = 1 | X = x)$ pour tout x.

- Pour s'affranchir de la contrainte de valeurs entre 0 et 1, on utilise souvent la transformation logistique consistant à estimer le `log-odds`

$$\log \frac{\mathbb{P}(Y = 1 | X = x)}{\mathbb{P}(Y = 0 | X = x)}.$$

Arbres de classification

- Les arbres de classification sont construits de manière analogue aux arbres de régression avec un algorithme récursif de partition binaire.
- La principale différence est que le split (j, t) de la région R en

$$R_{left} = \{x \in R \colon x_j \leq t\} \quad \text{et} \quad R_{right} = \{x \in R \colon x_j \leq t\}$$

est recherché de manière à minimiser la déviance

$$\sum_{i \colon x_i \in R_{left}} L(y_i, \hat{p}_{R_{left}}) + \sum_{i \colon x_i \in R_{right}} L(y_i, \hat{p}_{R_{right}})$$

où $\hat{p}_{R_{left}}$, $\hat{p}_{R_{right}}$ désignent la proportion de 1 dans la feuille de gauche et dans la feuille de droite respectivement.

- Alternativement, le critère d'impureté de Gini peut être minimisé

$$\text{Gini} = n_{R_{left}} \hat{p}_{R_{left}} (1 - \hat{p}_{R_{left}}) + n_{R_{right}} \hat{p}_{R_{right}}) (1 - \hat{p}_{R_{right}})$$

qui est assez proche d'un critère de variance intraclasse utilisé en régression.

Les méthodes de type boosting

et l'algorithme XGBoost

Méthodes d'ensemble : bagging et boosting

Une méthode d'apprentissage est dite méthode d'ensemble, lorsqu'elle combine de multiples modèles dits modèles basiques.

- Rappel : le bagging est une méthode d'ensemble utilisant l'agrégation bootstrap pour réduire la variance d'un modèle basique de faible biais et variance élevée.

  L'apprentissage se fait en parallèle sur les différents échantillons bootstrap.

  Dans le cas des arbres, on utilise de nombreux arbres profonds.

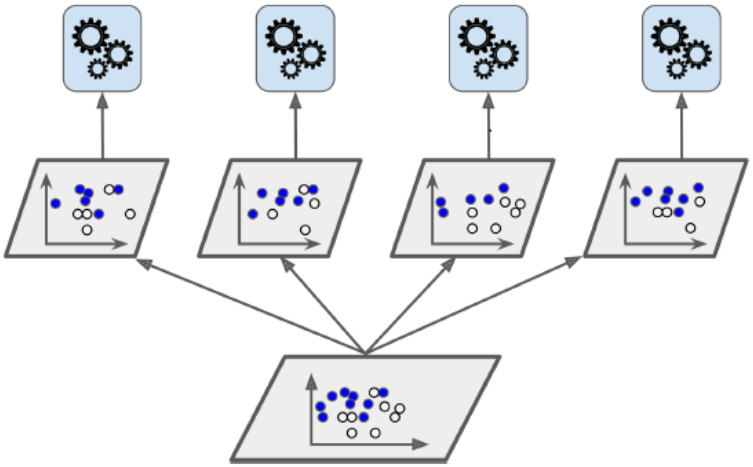
- **Nouveau :** le boosting est une méthode d'ensemble utilisant l'agrégation séquentielle pour réduire le biais d'un modèle basique de faible biais et variance élevée.

  L'apprentissage se fait en série : chaque modèle utilise l'information des modèles précédents et cherche à améliorer la prédiction.

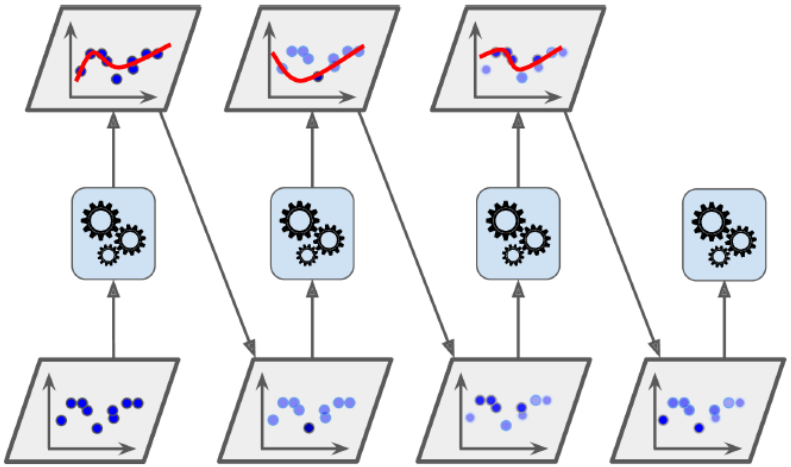
  Dans le cas des arbres, on utilise de nombreux arbres peu profonds.

Bagging VS Boosting

Bagging
apprentissage en parallèle



Boosting
apprentissage séquentiel



(images A. Géron, Hands on Machine Learning with Scikit-Learn)

Le "residual learning"

A titre pédagogique, un premier algorithme de type boosting en régression est l'algorithme suivant de type "residual learning".

Algorithme "residual learning" avec des arbres de régression

Input : échantillon d'apprentissage $(x_i, y_i)_{1 \leq i \leq n}$.
Paramètres : nombre total d'arbres $B \geq 1$; profondeur $d \geq 1$ de chaque arbre.
Procédure :

1) Initialisation $\hat{f}(x) = \bar{y}_n$;
2) Apprentissage séquentiel : pour $b = 1, \ldots, B$,
   i) Calculer les résidus $r_i = y_i - \hat{f}(x_i)$, $1 \leq i \leq n$;
   ii) Ajuster un arbre de régression $\hat{T}_b$ de profondeur d sur l'échantillon $(x_i, r_i)_{1 \leq i \leq n}$;
   iii) Mettre à jour le modèle en ajoutant le nouvel arbre $\hat{f}(x) \leftarrow \hat{f}(x) + \hat{T}_b(x)$;
3) Renvoyer le modèle

$$\hat{f}(x) = \bar{y}_n + \sum_{b=1}^{B} \hat{T}_b(x).$$

Mise en pratique : notebook pédagogique

Taux d'apprentissage

L'algorithme gradient boosting en régression s'obtient en ralentissant l'apprentissage grâce à l'introduction d'un taux d'apprentissage (learning rate).

Algorithme de gradient boosting en régression

Input : $(x_i, y_i)_{1 \leq i \leq n}$.
Paramètres : $B \geq 1$; $d \geq 1$; learning rate $\lambda \in (0, 1)$
Procédure :

1) Initialisation $\hat{f}(x) = \bar{y}_n$;
2) Apprentissage séquentiel : pour $b = 1, \ldots, B$,
   i) Calculer les résidus $r_i = y_i - \hat{f}(x_i)$, $1 \leq i \leq n$;
   ii) Ajuster un arbre de régression $\hat{T}^b$ de profondeur d sur l'échantillon $(x_i, r_i)_{1 \leq i \leq n}$;
   iii) Mettre à jour le modèle en ajoutant le nouvel arbre avec un facteur λ :

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x);$$

3) Renvoyer le modèle

$$\hat{f}_{boost}(x) = \hat{f}(x) = \bar{y}_n + \lambda \sum_{b=1}^{B} \hat{T}^b(x).$$

Commentaires : rôle de B

- L'apprentissage est ralenti grace au learning rate λ.
  Typiquement $\lambda = 0.01$ ou 0.001 est fixé.
  Des valeurs plus petites nécessitent un nombre d'arbres trop élevé.

- Contrairement au bagging, en boosting le choix du nombre d'arbres B est crucial dans le dilemme biais/variance :
  - B trop petit conduit au sous-apprentissage ;
  - B trop grand conduit au sur-apprentissage.

$\rightsquigarrow$ Le choix de B se fait par validation sur un échantillon test ou validation croisée.

Commentaires : rôle de d

- La profondeur d des arbres règle la complexité du modèle.
  En général, on utilise des arbres peu profonds ($d \leq 5$) avec une variance petite mais un biais élevé.

- Le cas $d = 1$ où chaque arbre a un seul noeud donne un modèle additif de la forme

$$\hat{f}_{boost}(x) = \sum_{j=1}^{p} \hat{f}_j(x_j).$$

  Il est possible de représenter ces fonctions ce qui permet de visualiser le modèle.

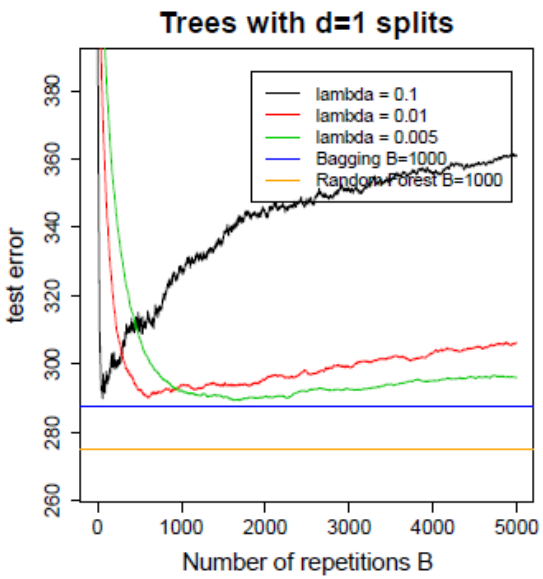
- Pour $d = 2$:

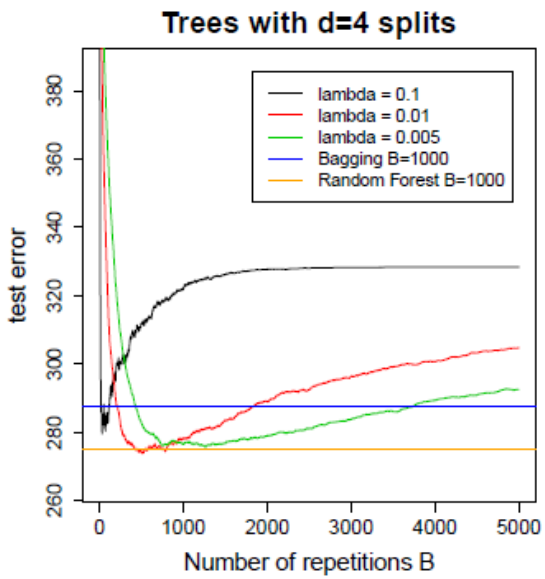
$$\hat{f}_{boost}(x) = \sum_{j=1}^{p} \hat{f}_j(x_j) + \sum_{1 \leq j_1 < j_2 < p} \hat{f}_{j_1,j_2}(x_{j_1}, x_{j_2}).$$

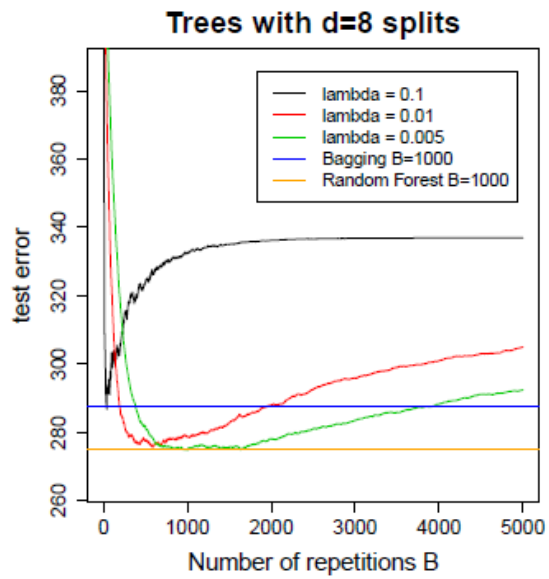
  Plus généralement, d peut être interprété comme l'ordre des d'interactions entre covariables autorisées par le modèle.

- Le choix de d se fait par validation sur un échantillon test ou validation croisée.

Boosting sur le jeu de données `Hitters`







Pourquoi le nom *Gradient boosting* ?

- D'un point de vue mathématique, on peut interpréter le gradient boosting comme une descente de gradient sur un espace fonctionnel de dimension infinie pour minimiser l'erreur quadratique moyenne

$$L_n(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2.$$

- En introduisant l'erreur quadratique $L(y,z) = \frac{1}{2}(y-z)^2$ entre l'observation y et la prédiction z (au facteur 1/2 près), le résidu $r_i = y_i - f(x_i)$ s'écrit comme le gradient

$$r_i = -\frac{\partial L}{\partial f(x_i)}(y_i, f(x_i))$$

- Heuristique : l'arbre de régression $\hat{T}$ entrainé sur les résidus apprend la direction minimisant L_n et la mise à jour $\hat{f} \leftarrow \hat{f} + \lambda\hat{T}$ correspond à un pas de descente de gradient sur l'espace des fonctions.

- Ces idées permettent d'adapter l'algorithme pour minimiser d'autres types d'erreur liées à d'autres tâches statistiques.

Exemples d'autres fonctionnelles de perte

- En régression robuste, on cherche à prédire la médiane conditionnelle $\mathrm{Med}(Y \mid X = x)$ et on utilise la fonctionnelle de perte MAE (Mean Absolute Error)

$$L_n(f) = \frac{1}{n} \sum_{i=1}^{n} |y_i - f(x_i)|.$$

  Le résidu s'écrit $r_i = \mathrm{sign}(y_i - f(x_i))$ (valeur ± 1).

- En classification binaire, pour prédire le log-odd $\log \frac{\mathbb{P}(Y=1|X=x)}{1-\mathbb{P}(Y=1|X=x)}$ on peut utiliser la fonction de perte

$$L(y, f(x)) = -yf(x) + \log(1 + e^{f(x)})$$

  Le résidu s'écrit $r_i = y_i - \frac{e^{f(x_i)}}{1+e^{f(x_i)}}$.

L'algorithme XGBoost

- XGBoost (eXtreme Gradient Boosting), conçu par Tianqi Chen (2016) est une implémentation optimisée du gradient boosting.
- Algorithme très efficace et populaire en prédiction :

  XGBoost : The super star of algorithms in ML competition – Benoit Cayla.

  Why does XGBoost win "every" machine learning competition ? – Nielsen Didrik.
- Avantages :
  - algorithme versatile et flexible s'adaptant à de multiples tâches, aux données catégorielles, aux données manquantes ;
  - implémentation optimisée et distribuée permettant la gestion des gros jeux de données (plusieurs milliards de lignes)
  - portabilité multi-environnement (C++, Python, R, Linux, Windows...)

XGBoost : principes de l'algorithme

- XGBoost cherche à minimiser une fonctionnelle de perte calculée sur l'échantillon d'apprentissage

$$L_n(f) = \sum_{i=1}^{n} L(y_i, f(x_i))$$

où $f : x \mapsto \hat{y} = f(x)$ est la fonction de prédiction à estimer.

Le choix de L dépend de la tâche statistique à effectuer.

- L'apprentissage est séquentiel avec un algorithme de descente de type Newton-Raphson utilisant les dérivées du second ordre.
- Un learning rate régule la vitesse d'apprentissage et plusieurs hyperparamètres permettent la régularisation du modèle.
- L'algorithme est très flexible permettant de bonnes performances mais nécessite un travail de réglage des différents hyperparamètres.

XGBoost - algorithme simplifié (source wikipedia)

Input: training set $\{(x_i, y_i)\}_{i=1}^{N}$, a differentiable loss function $L(y, F(x))$, a number of weak learners M and a learning rate α.

Algorithm:

1. Initialize model with a constant value:

$$\hat{f}_{(0)}(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta).$$

2. For m = 1 to M:
   1. Compute the 'gradients' and 'hessians':

$$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

   2. Fit a base learner (or weak learner, e.g. tree) using the training set $\left\{x_i, \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right\}_{i=1}^{N}$ by solving the optimization problem below:

$$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2}\hat{h}_m(x_i)\left[\phi(x_i) - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right]^2.$$

$$\hat{f}_m(x) = \alpha\hat{\phi}_m(x).$$

   3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$

3. Output $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x).$

Outils pour l'inspection et l'interprétation

Outils pour l'inspection et l'interprétation

- Les méthodes d'ensemble (Random Forest, Gradient Boosting) combinant de nombreux arbres, la simplicité et l'interprétabilité d'un unique arbre est difficile et ces méthodes agissent comme des "boîtes noires".
- Les outils pour l'inspection et l'interprétation permettent cependant :
  - d'identifier les variables ayant le plus d'impact sur la prédiction
    $\rightsquigarrow$ mesures d'importance des variables
  - d'analyser l'effet des variables les plus importantes sur la prédiction
    $\rightsquigarrow$ partial dependence plot, individual conditionnal expectation (ICE)
- Ces méthodes ne sont pas spécifiques à Random Forest et Gradient Boosting mais valables et pertinents pour tous les modèles de type boîte noire (black box models), y compris les modèles de réseaux de neurones.

Mesures d'importance des variables

- Mesures d'importance basée sur la permutation des variables
  Sur un échantillon test, on quantifie comment la permutation d'une variable affecte la prédiction .
  Une mesure d'importance élevée indique que la permutation de la variable dégrade fortement la qualité de la prédiction et donc que la variable est importante.

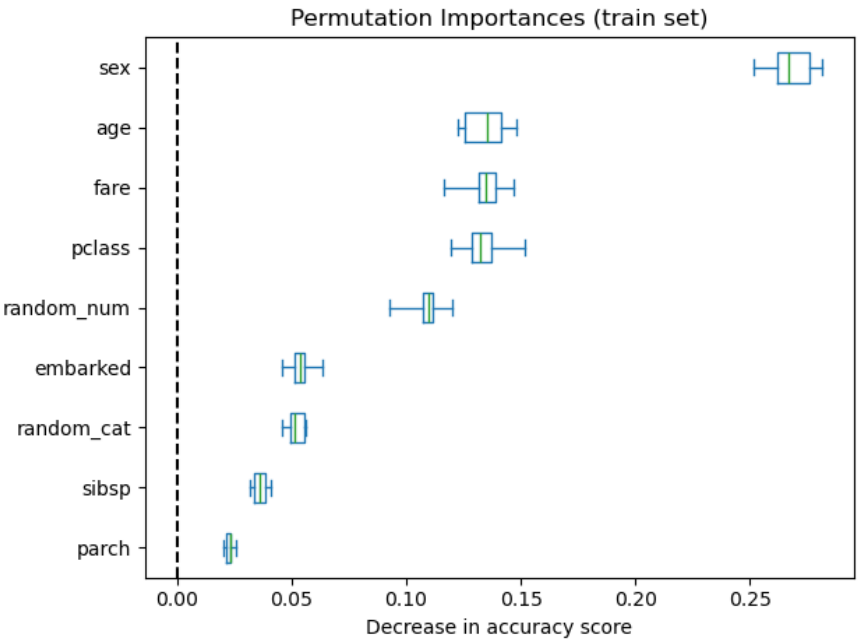
  $\rightsquigarrow$ identification des variables les plus influentes
  $\rightsquigarrow$ identification de variables non pertinentes qui pourraient éventuellement être supprimées du modèle.

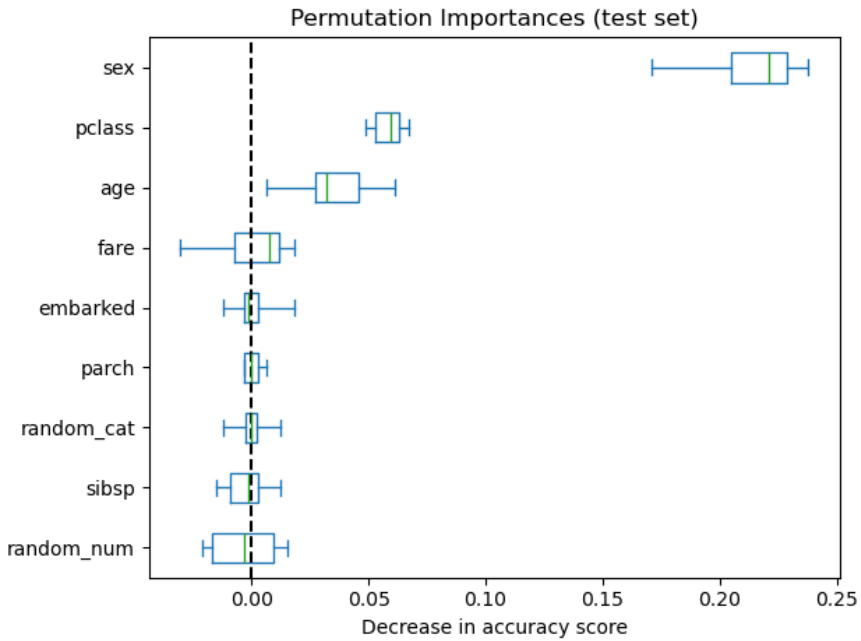
- Rappel - Dans le cas des arbres, des importances peuvent être calculer pour chaque variable selon la réduction du critère (MSE, deviance, Gini) lors des splits.
  Ce sont les mesures d'importance MDI ("mean decrease in impurity").
  Ces mesures sont moins spécifiques à la question de la prédiction et valables uniquement pour les arbres.

- Vignettes scikit-learn :
  exemple sur données simulées
  exemple sur données réelles

Exemple

Exemple basé sur le jeu de données `titanic`
lien vers la vignette scitkit-learn





Les variables `rand_num` et `rand_cat` ont été générées artificiellement et sont sans lien avec la réponse.

Partial depencence plot et Individual conditional expectation

- Pour les variables les plus importantes identifiées, on s'intéresse à l'impact de la variable sur la réponse :
  Une variable explicative a-t-elle un effect positif sur la réponse ? un effet négatif ? un effet non monotone ? Il y a-t-il un effet plutot linéaire ou bien non linéaire ?

- Le partial dependence plot pour une variable X_j (ICE) calcule sur l'échantillon d'apprentissage la réponse moyenne lorsque la variable X_j est arbitrairement remplacée par une valeur x que l'on fait varier. Ce partial dependence plot peut

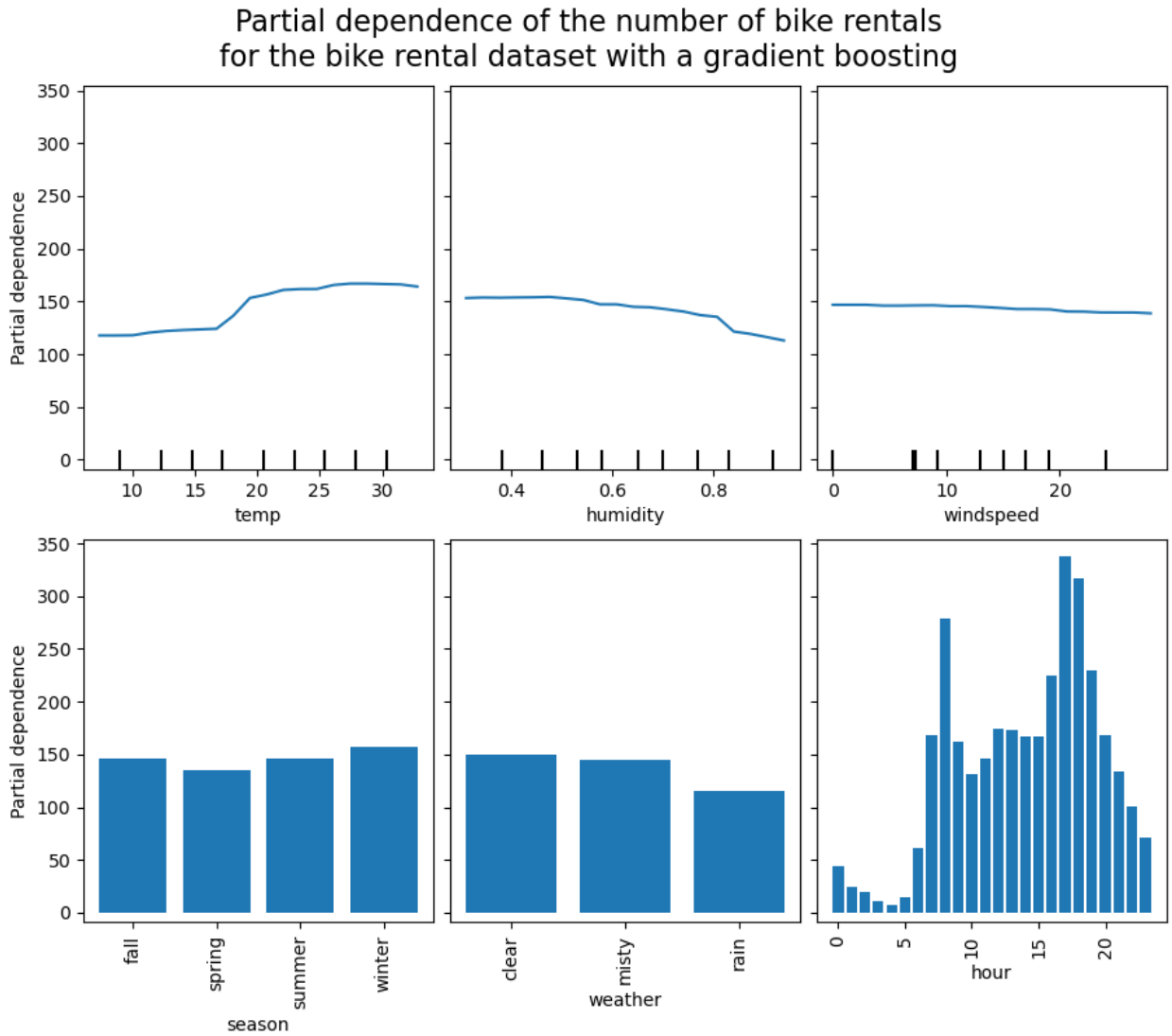
  être réalisé pour 2 variables X_{j_1}, X_{j_2} afin d'investiguer l'effet d'interaction des deux variables.

- Le plot ICE (Individual Conditional Expectation) représente les différentes courbes obtenues pour tous les individus de l'apprentissage (sans prendre la moyenne).
- Vignettes scikit-learn :
  exemple sur données simulées

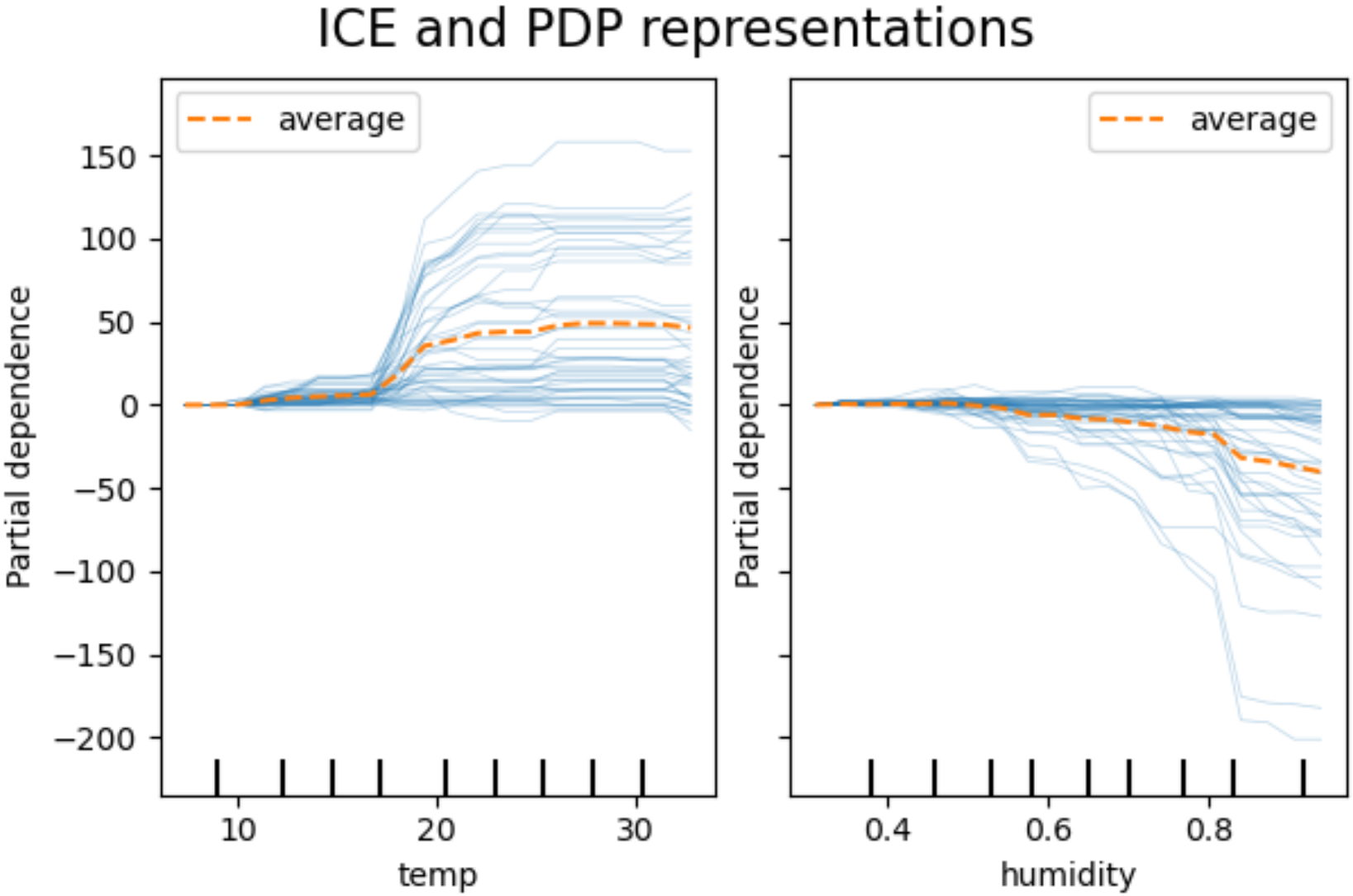
Exemple

Prédiction du nombre de locations de vélo sur le jeu de données `bike sharing`

lien vers la vignette scitkit-learn

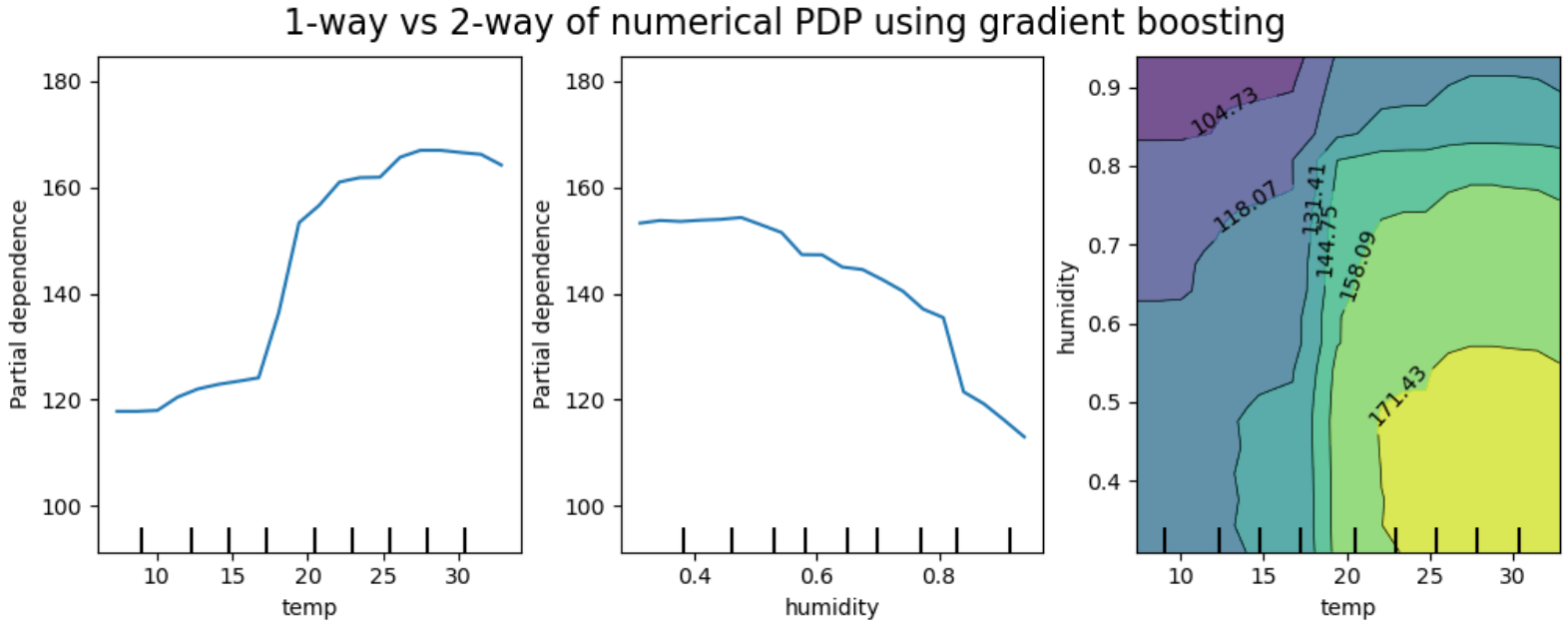


Exemple



Interprétation : la hausse des températures a globalement un effet positif sur le nombre de vélos loués ; celle de l'humidité a globalement un effet négatif. L'effet est plus marqué sur certaines courbes.

Exemple



Interprétation : effet d'interaction température / humidité.

- Température élevée et humidité faible ont un effet positif +++ sur le nombre de vélos loués ;
- Température faible et humidité élevée ont un effet négatif — sur le nombre de vélos loués.

Pour aller plus loin

Pour aller plus loin

Nous avons présenté les algorithmes classiques dans le cadre simple de la régression et de la classification binaire. Mais :

- de nombreuses variantes des algorithmes afin de
  - accélerer les calculs,
  - gérer les features de type facteur (variable qualitatives),
  - gérer les valeurs manquantes en prédiction.

- ces algorithmes de bases peuvent être adaptés pour effectuer d'autres tâches plus avancées :
  - imputation de valeurs manquantes avec l'algorithme MissForest,
  - prédiction de durée de survie (utile en médecine),
  - prédiction probabiliste pour analyser l'incertitude,

Sans rentrer dans les détails, nous survolons quelques thèmes qui pourraient être approfondis ou pour lesquels des packages peuvent être utilisés.

Utilisation des histogrammes

- Dans le cadre des données massives (big data), l'algorithme LightGBM a introduit l'utilisation d'histogramme pour discrétiser les variables explicatives.

  Ke et al. (2017), "LightGBM : A Highly Efficient Gradient Boosting Decision Tree"

- En phase préparatoire à l'algorithme, les variables explicatives sont discrétisées en "bins", ce qui permet une accélération du temps de calcul assez marquée pour les grands échantillons (à partir de plusieurs milliers).

- Scikitlearn propose l'utilisations des histogrammes en boosting via les fonctions `HistGradientBoostingClassifier` et `HistGradientBoostingRegressor`.

- Un autre avantage de ces fonctions est la gestion des variables catégorielles et des données manquantes de manière native.

Taches statistiques avancées

La modification de la fonction de coût permet d'envisager des tâches statistiques avancées comme par exemple :

- régression pour des données de comptage ;
  $\rightsquigarrow$ option `loss='poisson'` dans un arbre de régression ;
- régression quantile pour estimer la médiane (ou les quartiles) conditionnels ou encore pour des intervalles de confiance en prédiction ;
  $\rightsquigarrow$ option `loss='quantile'` dans `GradientBoostingRegressor` ;
- régression distributionnelle ;
  $\rightsquigarrow$ package `DRF` Distributional Random Forest sous R.
- analyse de survie avec des algorithmes de type Random Forest ou Gradient Boosting spécifiques ;
  $\rightsquigarrow$ librairie `scikit survival` sous Python.

En revanche, les modèles de réseaux de neurones sont généralement préférables pour les taches statistiques avec des données très structurées (images, signaux temporels, données textuelles).

Imputation de données manquantes

Mentionnons enfin un algorithme populaire pour l'imputation des données manquantes basé sur RandomForest.

Stekhoven, D.J. and Buehlmann, P. (2012), MissForest - nonparametric missing value imputation for mixed-type data, *Bioinformatics*, 28(1) 2012.

Algorithme MissForest

1) Initialiser l'imputation des valeurs manquantes par une imputation simple (moyenne ou mode).
2) Considérer chaque variable avec des valeurs manquantes :
   - sur l'échantillon où la variable est connue, entrainer RandomForest pour prédire la variable en question en fonction des autres,
   - puis sur l'échantillon où la variable est manquante imputer la valeur prédite par RandomForest.
3) Itérer jusqu'à la convergence ou l'atteinte d'un critère d'arrêt.

Une implémentation Python de cet algorithme est fourni par la librairie `MissForest`.