Université de Mons

Faculté des Sciences

Le phénomène de double descente au sein de l'Apprentissage automatique

Raphael Palmeri

Sous la direction de Souhaib Ben Taieb

Master en Sciences informatiques

Je soussigné, Palmeri Raphael, atteste avoir respecté les règles éthiques en vigueur

Année universitaire 2020-2021

Table des matières

1 Introduction

Dans le cadre du Master en Sciences Informatiques en horaire décalé, il est demandé de réaliser un mémoire sur un sujet proposé par les enseignants de l'UMons de la section Sciences Informatiques. Mon choix de sujet s'est porté sur "Le phénomène de double descente au sein de l'Apprentissage Machine (*'The double descent phenomenon in machine learning'*)".

Dans un premier temps, je vais présenter ce qu'est l'apprentissage Machine ainsi qu'expliquer brièvement les différentes sous-catégories existantes au sein de celui-ci.

Dans une deuxième partie, je vais expliquer les notions de biais et de variance, ainsi que le phénomène de compromis entre ces deux notions au sein de l'apprentissage machine.

Dans la troisième partie, je vais démontrer mathématiquement le compromis biais-variance et expliquer cette démonstration plus en détails.

Dans la quatrième partie, à l'aide de simulations, nous montrerons l'effet de ce phénomène ainsi que son comportement en face aux modifications des différents paramètres de la simulation.

2 Apprentissage automatique ("Machine Learning")

L'apprentissage automatique est la capacité d'un ordinateur à "apprendre" en se basant sur des données mises à sa disposition. Le terme "apprendre" désigne la capacité à détecter/trouver des répétitions (*'patterns'*) dans ces données. Ces répétitions permettent à la machine de donner une expertise par rapport à un problème donné ou une réponse à une certaine question à l'aide d'un modèle qu'il aura construit lors de cette apprentissage.[2][5]

L'apprentissage automatique est notamment utilisé dans différents domaines tels que l'automobile avec ses voitures sans conducteurs, les finances afin de notamment détecter les fraudes, mais aussi le domaine de la santé avec la possibilité d'essayer de produire un diagnostic sur base des informations disponibles à propos d'un patient, etc.

Afin de réaliser de l'apprentissage automatique, il est nécessaire d'avoir deux choses : un algorithme d'apprentissage (voir sous-section 2.1) et des données (voir sous-section 2.3) que nous allons explorer dans les prochaines sections.

2.1 Algorithme d'apprentissage

Il existe différentes catégories d'apprentissages en machine learning :

2.1.1 Apprentissage supervisés

Dans ce type d'apprentissage, la machine reçoit un ensemble de données avec les classes de tout les exemples existants. Par exemple, un expert aura déjà défini les différentes classes possibles pour la reconnaissance d'objets via des images ('Chaise', 'Table', 'Chien', 'Chat', ...). Les algorithmes de cette famille vont dès lors se baser sur ces classes déjà définies afin de pouvoir attribuer une classe (que l'on espère correcte) à une nouvelle donnée encore inconnue. Dans l'exemple ci-dessus, on parlera de **"Classification"**.
Il existe aussi des problèmes de **"Régression"**, ceux-ci tentent de lier une nouvelle donnée à un nombre réel. Par exemple, dans le cadre de l'estimation de prix de maison.

Étant donné le fait que le compromis Biais-Variance (voir section 3) n'existent qu'au sein de cette catégorie d'algorithmes, il est logique que celle-ci soit la famille que nous étudierons le plus dans ce rapport.

2.1.2 Apprentissage non-supervisés

Dans le cas d'algorithme non-supervisé, les données d'entrainement et de test sont mélangés. Le modèle n'aura aucun exemple pour s'aider à détecter un pattern, il devra le faire par lui même en étudiant les similarités entre les différentes données. Ces dernières seront ensuite rangées par groupes afin qu'un expert puisse par exemple les utiliser pour une recherche scientifique. Dans le cadre de l'utilisation de cette famille, on parlera de **"Clustering"**.

2.1.3 Apprentissage semi-supervisés et Apprentissage actif

Dans la plupart des situations, il est impossible de classifier l'ensemble des données d'apprentissage. Dans ce genre de cas, la machine doit dès lors apprendre des classes qui lui sont fournies, mais aussi des données non labellisées, c'est ce que l'on appelle l'apprentissage semi-supervisé. Dans le cadre où ce n'est pas un expert qui donne les classes mais bien la machine qui tente de leur donner un label, on se trouve dans le cas de l'apprentissage actif.

2.1.4 Apprentissage par transfert et Apprentissage multitâche

L'idée principale derrière l'apprentissage par transfert est d'aider le modèle à s'adapter à des situations qu'il n'as pas rencontrés précédemment. Cette forme d'apprentissage s'appuie sur le fait d'effectuer des réglages précis sur un modèle générale qui lui permettra de travailler dans un nouveau domaine.

2.1.5 Apprentissage par renforcement

L'apprentissage par renforcement se base sur l'idée de maximiser une récompense selon une ou plusieurs actions. On va dés lors définir en fonction des actions, si elles sont encouragées, ou au contraire, découragées.

2.1.6 Exemples d'algorithmes

— Prédicteurs linéaires (*'Linear Predictors'*) : tels que la régression linéaire, perceptron, etc.
— Boosting
— Support Vector Machines
— Arbres de décision (*'Decision Trees'*)
— Voisin le plus proche (*'Nearest Neighbor'*)
— Réseau de neurones (*'Neural Networks'*)
— ...

2.2 Notation mathématiques

Dans cette sous-section, je vais expliquer les différentes notations mathématiques nécessaires à la bonne compréhension des prochains chapitres.

Dans le cadre d'un apprentissage automatique supervisé, on cherche à prédire un résultat $y \in \boldsymbol{Y}$ à partir d'une donnée $x \in \boldsymbol{X}$ où les paires (x, y) proviennent d'une distribution inconnue $\boldsymbol{D}$.

Le problème d'apprentissage automatique consiste à apprendre une fonction $f' : \boldsymbol{X} \to \boldsymbol{Y}$ à partir d'un ensemble de données d'entrainement fini $\boldsymbol{S}$ contenant m variables indépendantes et identiquement distribuées ($i.i.d$) provenant de $\boldsymbol{D}$.

f' peut aussi être vue comme étant une hypothèse $h \in \boldsymbol{H}$, choisie à partir d'une classe d'hypothèses $\boldsymbol{H}$ contenant des fonctions possibles pour le modèle.

Dans un cadre idéal, la fonction f' serait équivalente à la fonction f, la 'vrai' fonction qui régit $\boldsymbol{X} \to \boldsymbol{Y}$.

Pour une fonction de perte $l : \boldsymbol{Y} \times \boldsymbol{Y}$, la qualité d'un prédicateur h peut être quantifié par le *risque* (ou *l'erreur attendue*) :

$$R(h) = \boldsymbol{E}_{(x,y)\sim D}\boldsymbol{l}(h(x), y)$$

Le but de l'apprentissage supervisé est de trouver $\min_{h\in\boldsymbol{H}} R(h)$, c'est-à-dire la valeur minimal de l'erreur attendue. Malheureusement, il nous est impossible de calculer réellement cette valeur car nous ne connaissons pas $\boldsymbol{D}$. Ce que nous connaissons, par contre, c'est *l'erreur d'entrainement* :

$$\hat{R}(h) = \boldsymbol{E}_{(x,y)\sim S}\boldsymbol{l}(h(x), y)$$

Cette erreur est issue de $\boldsymbol{S}$ qui est un ensemble provenant de $\boldsymbol{D}$, nous pouvons dés lors tenter de trouver la valeur minimal de celle-ci comme étant un substitut de celle de $R(h)$.

2.2.1 La fonction de perte quadratique

Elle s'exprime comme suit :

$$(y - y')^2$$

où **y** représente la valeur véritable de la vrai fonction et **y'** représente la valeur estimée par le modèle.

L'erreur quadratique moyenne quant à elle n'est que la moyenne des erreurs sur l'ensemble des données :

$$MSE = \frac{1}{n}\sum_{(x,y)\in D}(y - y')^2$$

2.2.2 Espérance mathématique d'une variable aléatoire

Dans les preuves de la décomposition du biais-variance, on peut y trouver une notation statistique appelée l'espérance mathématique qui se note $E(x)$ pour une variable aléatoire x.

L'espérance mathématique représente la moyenne pondérée des valeurs que peut prendre cette variable.

2.3 Données

Afin de permettre le bon fonctionnement de l'apprentissage automatique, il est nécessaire d'avoir des données. Celles-ci doivent être présentes en quantité et, dans le meilleur des cas, elles doivent êtres "nettoyées" c'est-à-dire qu'il faut parfois retirer des attributs inutiles, en modifier certains pour qu'il soit compréhensibles pour l'algorithme. En effet, il arrive parfois qu'un attribut défini dans les données ne soit pas exploitables par l'algorithme sans être modifiés ou précisés.

Ces données peuvent être distinguées en 2 catégories :

2.3.1 Données d'apprentissage

Ces données sont des exemples déjà traités par un expert dans le domaine. Elles peuvent être utilisés comme exemple d'apprentissage pour les algorithmes supervisés. Grâce à celles-ci, l'algorithme pourra générer un modèle qui pourra estimer la valeur (ou la classe) en fonction d'une donnée inconnue.

2.3.2 Données de test

Ces données sont destinés à valider le modèle créé par l'algorithme d'apprentissage. l'idée est de fournir des données inconnues du modèle, afin de vérifier et valider son comportement. Si le modèle produit des résultats extrêmement éloignés de la vérité, c'est qu'il n'est pas encore prêt. Il faut donc repasser par une phase d'apprentissage en fournissant potentiellement plus de données d'apprentissage et/ou en les rendant plus précises afin que la machine établisse un nouveau modèle dont les réponses seront plus correctes.

3 Le compromis biais - variance

Afin de mieux comprendre la notion du compromis de Biais-Variance, il est important de comprendre sa version plus générale qui est le compromis Approximation-Estimation.

3.1 Le compromis approximation - estimation

Le compromis approximation-estimation influencera l'erreur que réalisera un modèle sur ses prédictions. C'est un composant de l'erreur totale au sein de l'apprentissage machine. C'est aussi le seul élément que l'on peut tenter de minimiser afin de limiter celle-ci.

L'erreur totale est constitué de 3 choses :

$$error_{total} = error_{generalization} + error_{training} + error_{irreductible}$$

1. l'erreur de généralisation ou d'approximation (*'generalization error'*) : cette erreur est la conséquence de la sélection d'un sous-ensemble que l'on considère comme étant représentatif. De part cette sélection, on induit une possible erreur.

2. l'erreur d'entrainement ou d'estimation (*'training error'*) : cette erreur est la conséquence de l'apprentissage. Dans nos données sélectionnées pour l'apprentissage de la machine, on peut avoir des cas spécifiques qui ne se présentent que dans notre ensemble d'apprentissage, ce qui mènera le modèle à un 'biais d'apprentissage' et peut diminuer la précision de celui-ci lors de l'utilisation de données de test.

3. l'erreur irréductible (*'irreductible error'*) : cette erreur est la conséquence d'un traitement peu efficace des données en amont de l'apprentissage. Les données qui seront utilisées par l'algorithme doivent êtres nettoyées avant d'être utilisées. Cette erreur ne dépends donc pas de l'algorithme directement.
"Garbage In, Garbage Out", ce qui signifie que si les données en entrée ne sont pas correctes, les résultats ne sauraient l'être.

3.2 Le compromis biais - variance

Le compromis biais-variance permet de quantifier le compromis approximation-estimation lorsque l'on utilise la fonction de perte quadratique (ou perte *L2*) et plus particulièrement, l'erreur quadratique moyenne (*MSE*).

3.2.1 Le biais

Le biais est la mesure qui montre à quel point le modèle établi par l'algorithme d'apprentissage supervisé est proche de la 'vrai' fonction d'un problème donné.

La figure 1 montre une représentation graphique du biais. f représente la 'vrai' fonction d'un problème donné, H représente l'ensemble des hypothèses choisies et le point noir représente une hypothèse sélectionnée parmi H. Plus H est grand, plus on a de chances que f soit compris dedans et donc d'avoir un modèle qui est équivalent à la vraie fonction.

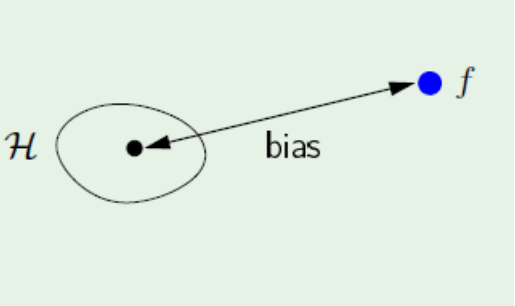

FIGURE 1 – Représentation du Biais.
[4]

3.2.2 La variance

La variance est la variation entre la valeur d'un ensemble de données de test par rapport à la valeur donnée par le modèle choisi.

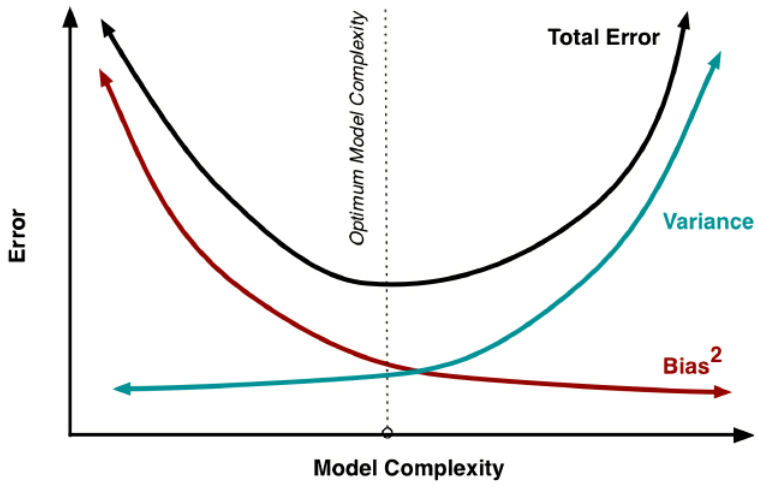

FIGURE 2 – Représentation de l'erreur en fonction de la complexité du modèle [4]

La figure 2 montre l'impact du biais et de la variance sur l'erreur d'un modèle et ce en fonction de sa complexité. Elle montre aussi très bien le point d'optimisation de la complexité du modèle lié au compromis entre le biais et la variance.

Un exemple concret du compromis biais-variance est celui du tir sur cible :

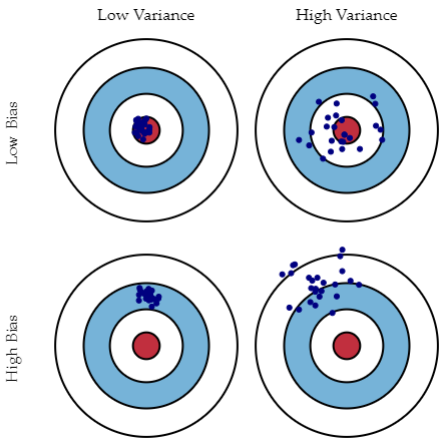

FIGURE 3 – Exemple concret du Compromis Biais-Variance.
[6]

Dans la figure 3, on a quatre cibles selon deux axes différents, le biais et la variance, chacun de ces axes peut-être soit faible, soit élevé.

Le premier cas (biais faible et variance faible) représente un excellent tireur, il vise toujours le centre et ses tirs sont fortement groupés.

Le second cas (biais faible et variance élevée) représente un 'bon' tireur, il vise le centre mais ses tirs sont assez dispersés.

Le troisième cas (biais élevé et variance faible) représente un tireur moyen, il ne vise pas le centre mais n'est pas non plus hors de la cible ou au bord de celle-ci et ses tirs sont fortement groupés.

Le quatrième cas (biais élevé et variance élevée) représente un mauvais tireur, il ne vise pas le centre et ses tirs sont très dispersés.

3.3 Décomposition du biais-variance

Considérons le modèle suivant :

$$y = f(x) + \epsilon \tag{1}$$

où :
— x ∼ p(x)
— f est une fonction fixée inconnue
— ϵ est du bruit aléatoire tel que :
  — $E[\epsilon|x] = 0$
  — $Var(\epsilon|x) = \sigma^2$

étant donné un set de données $D = (x_i, y_i)_{i=1}^n$ où (x_i, y_i) est un échantillon provenant de (1), et un ensemble d'hypothèses H, on calcule :

$$g^{(D)} = argmin_{h \in H} E_{in}(h) := \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i))$$

étant donné D, l'erreur au carré hors-échantillon de $g^{(D)}$ est :

$$E_{out}(g^{(D)}) = E_{x,y}[(y - g^{(D)}(x))^2]$$

considérons

$$E_D[E_{out}(g^{(D)})] = E_{x,y,D}[(y - g^{(D)}(x))^2] \tag{2}$$

représentant l'espérance moyenne sur les variables x, y et D.

En prenant $\bar{g} = E_D[g^{(D)}(x)]$, on peut décomposer (2) comme suit

$$\underbrace{E_x[(f(x) - \bar{g}(x))^2]}_{Biais} + \underbrace{E_{x,D}[(\bar{g}(x) - g^{(D)}(x))^2]}_{Variance} + \underbrace{\sigma^2}_{Variance irreductible}$$

On sait que le compromis biais-variance s'exprime comme suit :

$$E_{x,y,D}[(y - g^{(D)}(x))^2]$$

En sachant que le g moyen est :

$$\bar{g}(x) = E_D[g^{(D)}(x)] \tag{3}$$

Prouvons qu'il est égal à :

$$E_x[(f(x) - \bar{g}(x))^2] + E_{x,D}[(\bar{g}(x) - g^{(D)}(x))^2] + \sigma^2$$

(Voir Preuve en annexe A)

4 Le phénomène de double descente

Ce phénomène se présente comme suit : tout d'abord on remarque que l'erreur totale d'un modèle évolue en fonction du compromis biais-variance comme attendu mais lorsque la complexité augmente encore, on atteint un point d'interpolation à partir duquel l'erreur total décroit afin d'atteindre un niveau minimal plus faible que le meilleur choix pour le compromis.

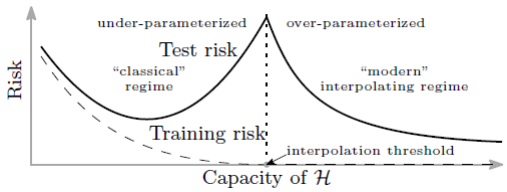

FIGURE 4 – Représentation du phénomène de double descente en fonction de la capacité de H

[4]

5 Simulation

Afin de montrer le comportement du biais et de la variance, j'ai réalise une série de simulations.

5.1 Simulation sur le nombre d'échantillons

Dans cette sous-section, les simulations ont été réalisées sur la base d'un changement de nombre d'échantillons utilisés, sélectionnés aléatoirement, comme données d'entrainement par l'algorithme pour définir un modèle. Afin d'obtenir une moyenne d'erreur correcte, chaque algorithme crée 20 modèles différents.

Les différentes tailles d'échantillons utilisées sont : $100, 500, 1000, 2000, 4000, 8000, 10000$.

Les données sont basés sur un ensemble de données représentant les propriétés physico-chimiques de la structure tertiaire des protéines (disponible à l'adresse suivante : `https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure`). Aucune traitement spécifiques n'a été fait sur les données, elle sont utilisés telles quelles.

Les données de test sont un échantillon de 1000 enregistrements choisi aléatoirement sur les 45730 disponibles dans l'ensemble.

Le code utilisé est présenté dans l'annexe **??**.

5.1.1 Régression Linéaire

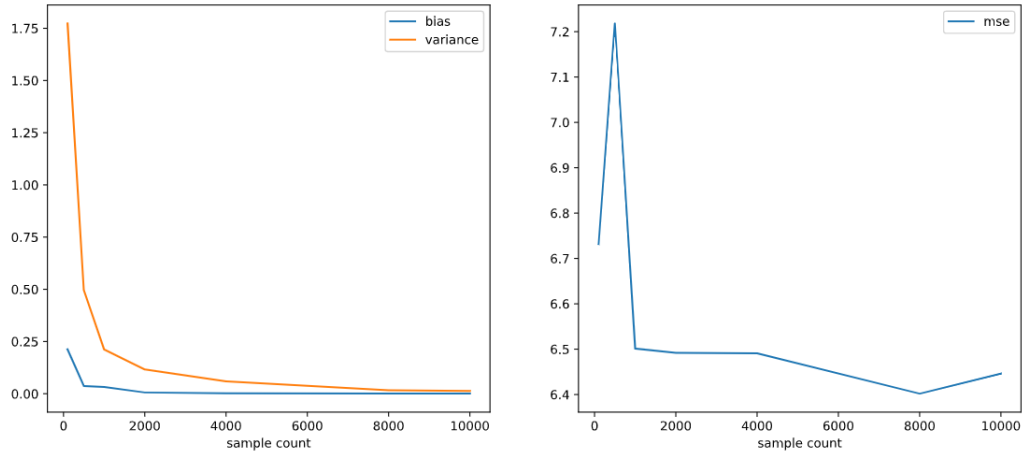

FIGURE 5 – Simulation d'une Régression Linéaire en fonction du nombre d'échantillons

Dans la figure 5, on remarque que le biais est faible au départ de la simulation, il a approximativement une valeur de 0.25 et qu'il tends vers 0 lorsque le nombre d'échantillons augmente. Cela indique possiblement que l'ensemble de données est régi par une fonction linéaire et que le modèle c'est approché de cette "vraie" fonction.

Pour ce qui est de la variance, elle a tout d'abord une valeur approximativement égale à 1.75 avant de décroitre rapidement avec l'augmentation du nombres d'échantillons jusqu'à plus ou moins 1000 échantillons, où la diminution se fait de manière plus réduite. Elle finie par tendre vers 0 lorsque l'on dépasse les 8000 échantillons.

On observe que l'erreur quadratique croit en début de simulation avant d'atteindre son paroxysme à la valeur de 7.2 lorsque 500 enregistrements sont utilisés. Finalement, elle décroit rapidement entre 500 et 1000 enregistrements avant d'atteindre sa valeur la plus basse avec 8000 enregistrements qui est de 6.4.

5.1.2 Arbre de décision

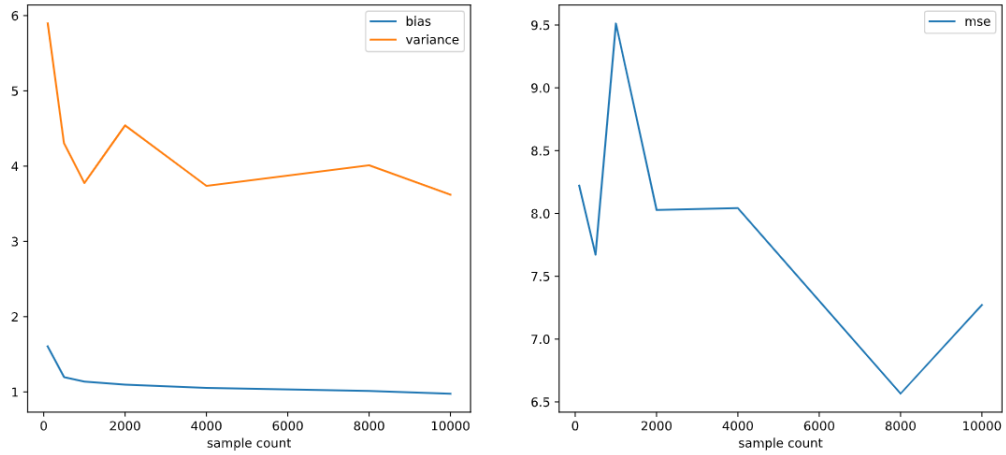

FIGURE 6 – Simulation d'un Arbre de décision en fonction du nombre d'échantillons

Dans la figure 6, on remarque que le biais tends vers 1 en partant de 1.5 en début de simulation. Ce n'est à proprement parler pas un mauvais résultat mais en le comparant à la régression linéaire, cette valeur est plus importante.

La variance part de la valeur 6 en début de simulation, pour ne jamais descendre en dessous 3.5. En comparatif de la régression linéaire, la valeur de variance est plus importante. On remarque aussi qu'elle n'est pas constamment entrain de décroitre ou de croitre mais qu'elle oscille entre les deux.

L'erreur quadratique oscille entre décroissement et accroissement jusqu'à la valeur clé de 2000 enregistrements où celle-ci ne fait que décroitre pour atteindre son minimum avec 8000 enregistrements avant de croitre à nouveau.

5.1.3 Bagging

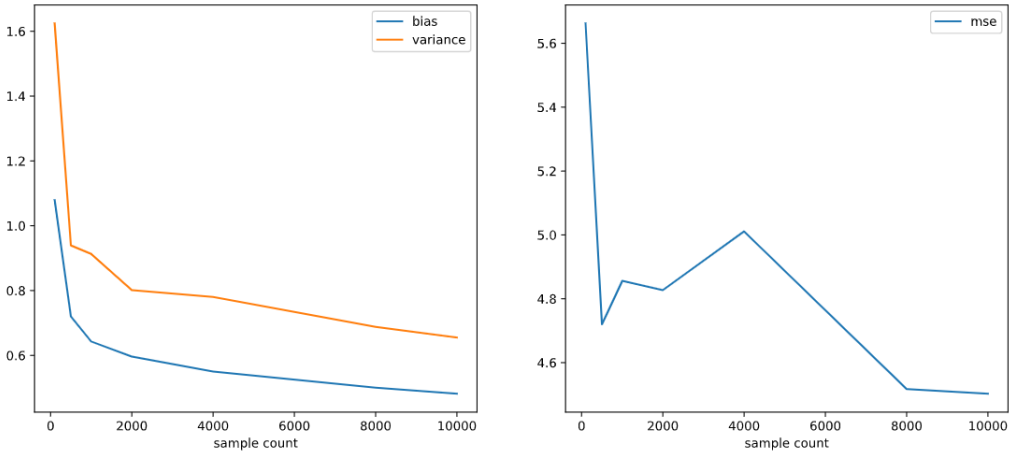

FIGURE 7 – Simulation d'un Bagging en fonction du nombre d'échantillons

Dans la figure 7, on remarque que le biais tends vers 0.5 en partant de 1.1 en décroissement tout d'abord rapidement et ensuite de manière moins marqué en arrivant à 500 enregistrements.

La variance part de la valeur 1.6 en début de simulation, pour ensuite décroitre rapidement jusqu'à 500 enregistrements et finir par décroitre plus lentement à partir de 2000 enregistrements pour finalement atteindre 0.7 qui est sa valeur minimum.

L'erreur quadratique décroit fortement en début de simulation pour s'accroitre à nouveau après 500 enregistrements. A 4000 enregistrements, l'erreur quadratique ne fait plus que décroitre pour atteindre son minimum en 4.5.

5.1.4 Random Forest

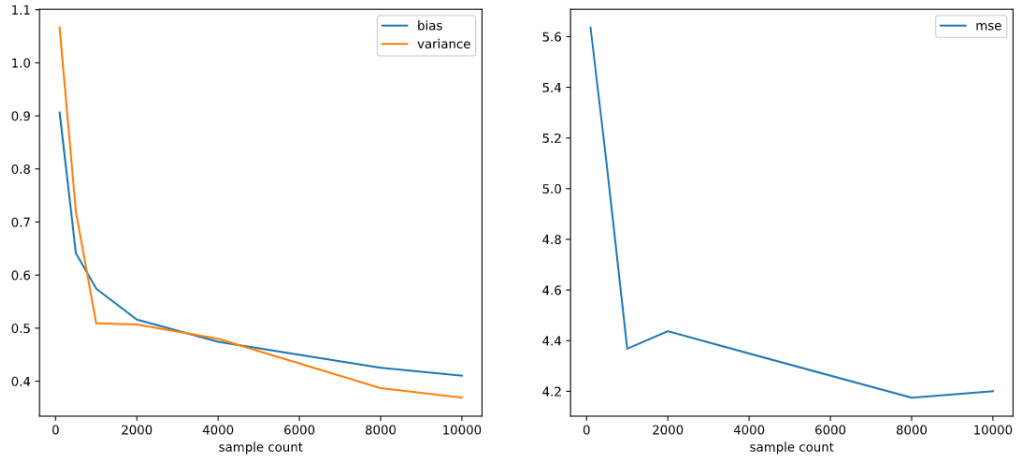

FIGURE 8 – Simulation d'une Forêt aléatoire en fonction du nombre d'échantillons

Dans la figure 8, on remarque que le biais et la variance se croisent deux fois lors de la simulation. Le biais décroit rapidement en début de simulation et décroit plus lentement au passage des 2000 enregistrements. Il arrive finalement à son minimum de 0.41 en fin de simulation.

La variance, elle aussi, décroit rapidement avant les 2000 enregistrements pour finalement décroitre plus lentement. Elle atteint son minimum de 0.39 en fin de simulation également.

L'erreur quadratique décroit rapidement jusqu'à 1000 enregistrements, s'accroit légèrement jusque 2000 enregistrements pour finalement atteindre son minimum à 8000 enregistrements avant de repartir à la hausse.

5.1.5 Ada Boost

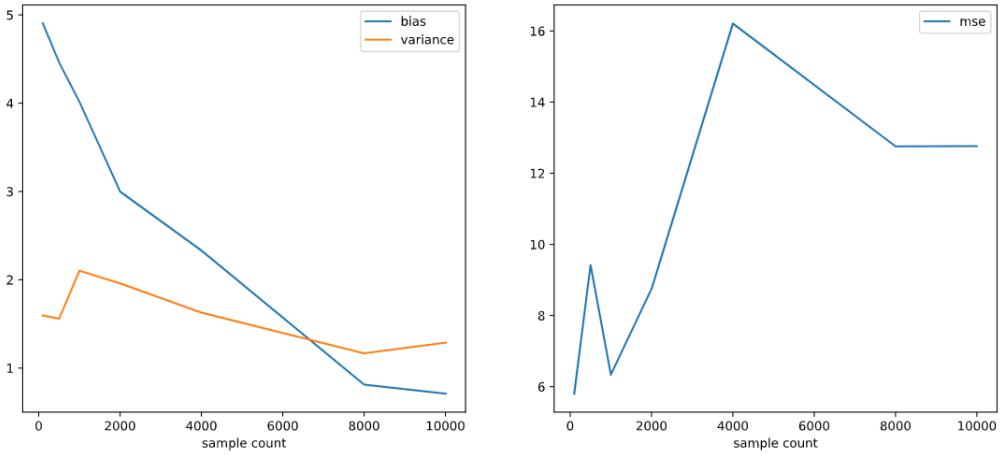

FIGURE 9 – Simulation Ada Boost en fonction du nombre d'échantillons

Dans la figure 9, On remarque que le biais et la variance se croise en un point (6500 enregistrements). Le biais ne fait jamais que décroitre mais avec trois rythmes différents , d'abord rapidement jusqu'à 2000 enregistrements ; un peu plus modérément entre 2000 et 8000 enregistrements et finalement assez lentement entre 8000 et 10000 enregistrements.

La variance décroit légèrement lors du début de la simulation pour croitre un peu plus entre 500 et 1000 enregistrements et ensuite décroit de manière modéré ente 1000 et 8000 enregistrements.

L'erreur quadratique croit dans un premier temps, puis décroit pour croitre de manière assez rapide entre 1000 et 4000 enregistrements. Ensuite, de 4000 à 8000 enregistrements, elle décroit pour finalement se stabiliser à 13.

5.1.6 K plus proches voisins

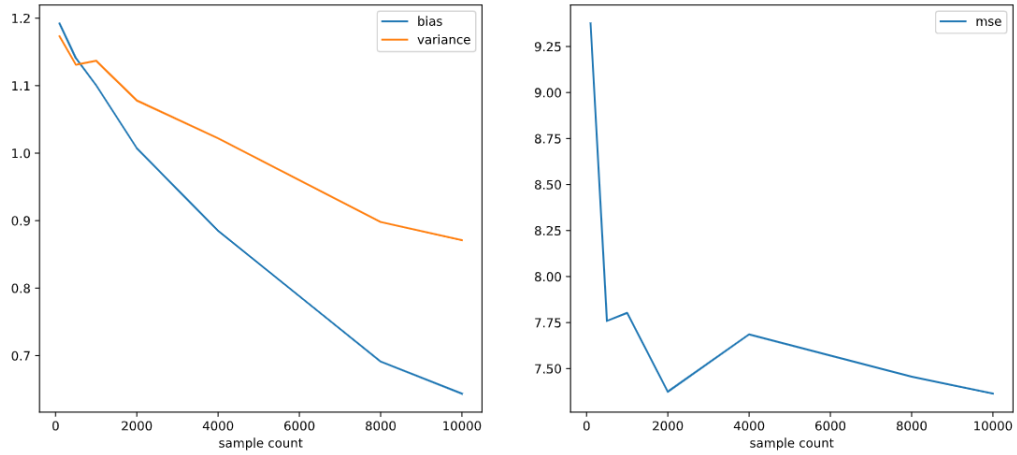

FIGURE 10 – Simulation K plus proches voisins en fonction du nombre d'échantillons

Dans la figure 10, On remarque que le biais ne fait que décroitre au fur et à mesure du nombres d'enregistrements utilisés. Sa valeur minimal se situe à 0.65 avec 10000 enregistrements.

La variance suit une tendance identique à celle de biais si ce n'est que la décroissance est moins prononcée. Sa valeur minimal est de 0.9 avec 10000 enregistrements.

L'erreur quadratique montre d'abord une décroissance rapide avec ensuite une alternance entre accroissement et décroissement avec un premier minimum atteint à 2000 enregistrements avant de croitre à nouveau. On remarque aussi que le minimum peut-être atteint à nouveau avec 10000 enregistrements.

5.2 Simulation K voisin le plus proche sur K

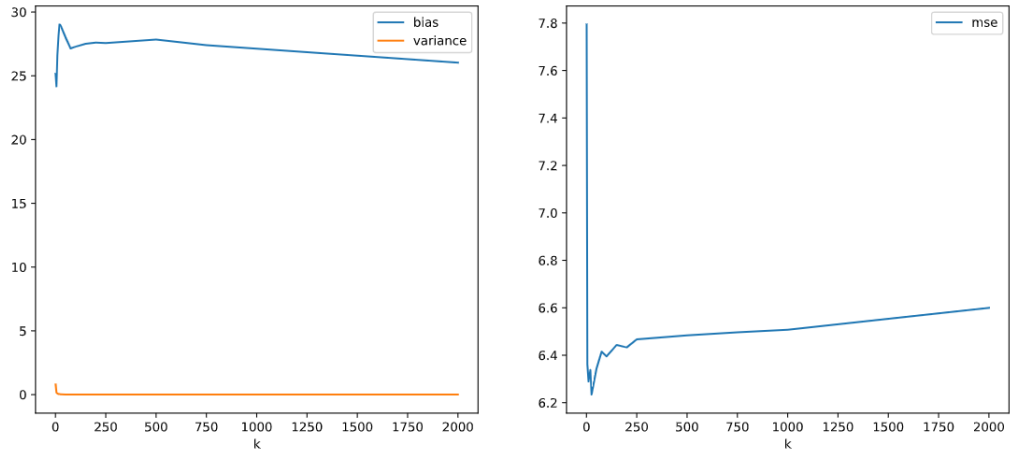

FIGURE 11 – Simulation K plus proches voisins en fonction du nombre de voisins requis

Dans la figure 11, on remarque que le biais est fortement élevé. Cela indique que le l'ensemble de données utilisé n'est pas le plus adapté pour l'utilisation d'un algorithme tel que celui des k-voisins les plus proches.

On remarque que l'erreur quadratique diminue lors du début de la simulation pour finalement remonté par la suite de la simulation. Nous avons là un représentation du compromis biais-variance.

6 Conclusion

Annexes

A Preuve mathématique

Reprenons l'expression du compromis :

$$E_{x,y,D}[(y - g^{(D)}(x))^2] \tag{4}$$

On peut exprimer (4) comme suit :

$$E_x[E_{y,D}[(y - g^{(D)}(x))^2|x]] \tag{5}$$

En fixant x, on peut simplifier (5) :

$$E_{y,D}[(y - g^{(D)}(x))^2] \tag{6}$$

En ajoutant $-f(x) + f(x)$ à l'équation 6, on obtient :

$$E_{y,D}[(y - f(x) + f(x) - g^{(D)}(x))^2] \tag{7}$$

En considérant $y - f(x)$ comme étant a et $f(x) - g^{(D)}(x)$ comme b et en appliquant la formule $(a + b)^2 = a^2 + b^2 + 2ab$ dans l'équation (7), on obtient :

$$E_{y,D}[(y - f(x))^2] + E_{y,D}[(f(x) - g^{(D)}(x))^2] + 2E_{y,D}[(y - f(x))(f(x) - g^{(D)}(x))] \tag{8}$$

En utilisant (1) pour remplacer y dans l'équation (8), on obtient :

$$E_{y,D}[(f(x)+\epsilon-f(x))^2]+E_{y,D}[(f(x)-g^{(D)}(x))^2]+2E_{y,D}[(y-f(x))(f(x)-g^{(D)}(x))] \tag{9}$$

En simplifiant l'équation (9), on obtient :

$$E_{y,D}[(\epsilon)^2] + E_{y,D}[(f(x) - g^{(D)}(x))^2] + 2E_{y,D}[(y - f(x))(f(x) - g^{(D)}(x))] \tag{10}$$

En utilisant la définition de la Variance de ϵ de la section 3.3, on peut simplifier l'équation (10) comme suit :

$$\sigma^2 + E_{y,D}[(f(x) - g^{(D)}(x))^2] + 2E_{y,D}[(y - f(x))(f(x) - g^{(D)}(x))] \tag{11}$$

En utilisant (26) (voir section B) dans l'équation (11), on obtient :

$$\sigma^2 + E_{y,D}[(f(x) - g^{(D)}(x))^2] + 0 \tag{12}$$

En ajoutant $-\bar{g}(x) + \bar{g}(x)$ à l'équation (12) dans le terme $E_{y,D}[(f(x) - g^{(D)}(x))^2]$, on obtient :

$$\sigma^2 + E_{y,D}[(f(x) - \bar{g}(x) + \bar{g}(x) - g^{(D)}(x))^2] + 0 \tag{13}$$

En considérant $f(x) - \bar{g}(x)$ comme étant a et $\bar{g}(x) - g^{(D)}(x)$ comme b et en appliquant la formule $(a + b)^2 = a^2 + b^2 + 2ab$ dans l'équation (13), on obtient :

$$\sigma^2+E_{y,D}[(f(x)-\bar{g}(x))^2]+E_{y,D}[(\bar{g}(x)-g^{(D)}(x))^2]+2E_{y,D}[(f(x)-\bar{g}(x))(\bar{g}(x)-g^(D)(x))] \tag{14}$$

En vérifiant les espérances, on peut encore simplifier l'équation (14) en :

$$\sigma^2+(f(x)-\bar{g}(x))^2+E_D[(\bar{g}(x)-g^{(D)}(x))^2]+2E_{y,D}[(f(x)-\bar{g}(x))(\bar{g}(x)-g^(D)(x))] \tag{15}$$

En utilisant la preuve intermédiaire (31) (voir section C), on obtient l'équation suivante :

$$\sigma^2 + (f(x) - \bar{g}(x))^2 + E_D[(\bar{g}(x) - g^{(D)}(x))^2] + 0 \tag{16}$$

et finalement en ré-appliquant l'espérance de x que nous avions retiré pour faciliter la notation, on obtient :

$$\sigma^2 + E_x[(f(x) - \bar{g}(x))^2] + E_{x,D}[(\bar{g}(x) - g^{(D)}(x))^2] \tag{17}$$

ce qui prouve bien que $E_{x,y,D}[(y - g^{(D)}(x))^2]$ est équivalent à (17)

B Preuve intermédiaire de $2E_{y,D}[(y-f(x))(f(x)-g^{(D)}(x))]$

Prouvons que $E_{y,D}[(y - f(x))(f(x) - g^{(D)}(x))]$ est $= 0$

$$E_{y,D}[(y - f(x))(f(x) - g^{(D)}(x))] \tag{18}$$

On peut distribuer dans l'équation (18), on obtient :

$$E_{y,D}[yf(x) - yg^{(D)}(x) - f^2(x) + f(x)g^{(D)}(x)] \tag{19}$$

En utilisant (1) dans l'équation (19), on obtient :

$$E_{y,D}[(f(x) + \epsilon)f(x) - (f(x) + \epsilon)g^{(D)}(x) - f^2(x) + f(x)g^{(D)}(x)] \tag{20}$$

En séparant les différents éléments et en simplifiant dans (20), on obtient :

$$E_{y,D}[f^2(x)+\epsilon f(x)]-E_{y,D}[f(x)g^{(D)}(x)+\epsilon g^{(D)}(x)]-E_{y,D}[f^2(x)]+E_{y,D}[f(x)g^{(D)}(x)] \tag{21}$$

En vérifiant les espérances, on peut encore simplifier l'équation (21) en :

$$f^2(x) + \epsilon f(x) - E_D[f(x)g^{(D)}(x) + \epsilon g^{(D)}(x)] - f^2(x) + E_D[f(x)g^{(D)}(x)] \tag{22}$$

En utilisant (3) dans l'équation (22), on obtient :

$$f^2(x) + \epsilon f(x) - f(x)\bar{g}(x) + \epsilon g^{(D)}(x) - f^2(x) + f(x)\bar{g}(x) \tag{23}$$

Pour faciliter la notation, nous avions fixé x, l'équation (23) donne en réalité :

$$E_x[f^2(x) + \epsilon f(x) - f(x)\bar{g}(x) + \epsilon g^{(D)}(x) - f^2(x) + f(x)\bar{g}(x)] \tag{24}$$

En utilisant la définition de l'espérance de ϵ dans l'équation (24), on obtient :

$$E_x[f^2(x) + 0f(x) - f(x)\bar{g}(x) + 0g^{(D)}(x) - f^2(x) + f(x)\bar{g}(x)] \tag{25}$$

En simplifiant l'équation (25), on obtient finalement :

$$E_x[f^2(x) - f(x)\bar{g}(x) - f^2(x) + f(x)\bar{g}(x)] = E_x[0] = 0 \tag{26}$$

On a donc prouvé mathématiquement que (18) est bien égale à 0

C Preuve intermédiaire de $2E_{y,D}[(f(x)-\bar{g}(x))(\bar{g}(x)-g^{(D)}(x))]$

Prouvons que $E_{y,D}[(f(x) - \bar{g}(x))(\bar{g}(x) - g^{(D)}(x))]$ est $= 0$

$$E_{y,D}[(f(x) - \bar{g}(x))(\bar{g}(x) - g^{(D)}(x))] \tag{27}$$

On peut distribuer dans l'équation (27), on obtient :

$$E_{y,D}[f(x)\bar{g}(x) - f(x)g^{(D)}(x) - g^2(x) + \bar{g}(x)g^{(D)}(x)] \tag{28}$$

en vérifiant les espérances dans l'équation (28), on obtient :

$$f(x)\bar{g}(x) + E_D[-f(x)g^{(D)}(x)] + E_D[\bar{g}(x)g^{(D)}(x)] - g^2(x) \tag{29}$$

en appliquant la formule du g moyen (3), on obtient l'équation suivante :

$$f(x)\bar{g}(x) - f(x)\bar{g}(x) + \bar{g^2}(x) - \bar{g^2}(x) \tag{30}$$

Finalement, en simplifiant l'équation (30), on obtient :

$$f(x)\bar{g}(x) - f(x)\bar{g}(x) + \bar{g^2}(x) - \bar{g^2}(x) = 0 \tag{31}$$

D Code de Simulation sur le nombre d'enregistrements

```
import sys
import tqdm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor

#Load the data and prepare train & test data
population_data= pd.read_csv("CASP.csv")

test_data = population_data.sample(1000, replace=False, random_state=100)
train_data = population_data.drop(test_data.index, axis=0).reindex()
test_data = test_data.reindex()

X_train = population_data.drop(['F9'], axis=1)
y_train = population_data['F9']
X_test = test_data.drop(['F9'], axis=1)
y_test = test_data['F9']

def calculate_variance_of_model(samplePredictions, y_test):
        predictions_mean_model = samplePredictions.mean(axis =1)
        colNames = samplePredictions.columns
        variance = np.zeros(len(colNames))
        i = 0
        for colName in colNames:
        variance[i] = np.mean(np.square(samplePredictions[colName] - predictio
        rmse = mean_squared_error(y_test, samplePredictions[colName])
        i += 1
        return round(np.mean(variance),3), round(np.mean(rmse),3)

def calculate_bias_of_model(samplePredictions, y_hat_pop):
        return np.square((np.abs(samplePredictions.mean(axis=1) -y_hat_pop).m

def samplePredForEstimator(estimator, X_train, y_train):
        sample_Model = estimator.fit(X_train, y_train)
        return sample_Model.predict(X_test)

def get_bias_variance(sampleCount, noOfModels, y_hat_pop, estimator_name, esti
        bias_variance_result = pd.DataFrame(columns=['sample_count','no_of_mo
        print('Builds ', estimator_name)
        print('Total No of models built is ',str(noOfModels))

        samplePredictionsModel = pd.DataFrame()
```

```
        with tqdm.tqdm(total=noOfModels, file=sys.stdout) as pbar:
        for i in range(0, noOfModels):
        pbar.set_description('Building Model : %d' % (1 + i))
        sample = train_data.sample(sampleCount,replace=False)
        X_train = sample.drop(['F9'], axis=1)
        y_train = sample['F9']

        samplePredictionsModel['sample'+str(i+1)] = samplePredForEstimator(es
        pbar.update(1)

        var_mse_linear_model = calculate_variance_of_model(samplePredictionsM

        bias_linear_model = calculate_bias_of_model(samplePredictionsModel, y

        s_linear = pd.Series(data={'sample_count':sampleCount, 'no_of_models'
                'bias':bias_linear_model, 'variance' : var_mse_linear_model[0

        bias_variance_result = bias_variance_result.append(s_linear)

        bias_variance_result.reset_index(inplace=True)
        bias_variance_result.drop(['index'], axis=1, inplace=True)

        return bias_variance_result

def bias_variance_mse_graphics(noOfSamples, noOfModels, y_hat_pop, estimator_n
        lenNoOfSamples = len(noOfSamples)
        bias_variance_result = pd.DataFrame()
        print('Building Model for samples ', noOfSamples)
        for i in range(0, lenNoOfSamples):
        noOfSample = noOfSamples[i]
        print('Building models with sample size : ', noOfSample)
        bias_variance_result = bias_variance_result.append(get_bias_variance(
        bias_variance_result.reset_index(inplace=True)
        bias_variance_result.drop(['index'], axis=1, inplace=True)

        # Plotting the obtained results
        fig, ax = plt.subplots(1,2, figsize = (14,6))
        ax[0].plot(bias_variance_result.sample_count, bias_variance_result.bi
        ax[0].plot(bias_variance_result.sample_count, bias_variance_result.va
        ax[0].legend(['bias','variance'])
        ax[0].set_xlabel('sample count')

        ax[1].plot(bias_variance_result.sample_count, bias_variance_result.ms
        ax[1].legend(['mse'])
        ax[1].set_xlabel('sample count')
        plt.show()

estimators = [LinearRegression(), DecisionTreeRegressor(),
BaggingRegressor(), RandomForestRegressor(), AdaBoostRegressor(loss='square')
KNeighborsRegressor()]

for estimator in estimators :
        population_model = estimator.fit(X_train, y_train)
        y_hat_pop = population_model.predict(X_test)
```

```
estimator_name = str(estimator).replace('()','')
bias_variance_mse_graphics([100,500,1000,2000,4000,8000,10000], 20, y
```

E Code de Simulation sur le nombre de voisins

```
import sys
import tqdm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor

#Load the data and prepare train & test data
population_data= pd.read_csv("CASP.csv")

test_data = population_data.sample(1000, replace=False, random_state=10
train_data = population_data.drop(test_data.index, axis=0).reindex()
test_data = test_data.reindex()

X_train = population_data.drop(['F9'], axis=1)
y_train = population_data['F9']
X_test = test_data.drop(['F9'], axis=1)
y_test = test_data['F9']

def calculate_variance_of_model(samplePredictions, y_test):
        predictions_mean_model = samplePredictions.mean(axis =1)
        colNames = samplePredictions.columns
        variance = np.zeros(len(colNames))
        i = 0
        for colName in colNames:
        variance[i] = np.mean(np.square(samplePredictions[colName] - pr
        rmse = mean_squared_error(y_test, samplePredictions[colName])
        i += 1
        return round(np.mean(variance),3), round(np.mean(rmse),3)

def calculate_bias_of_model(samplePredictions, y_hat_pop):
        return np.square((np.abs(samplePredictions.mean(axis=1) -y_hat_

def samplePredForEstimator(n_neigh, X_train, y_train):
        sample_Model = KNeighborsRegressor(n_neighbors=n_neigh).fit(X_t
        return sample_Model.predict(X_test)

def get_bias_variance(sampleCount, noOfModels, estimator_name, n_neigh)
        bias_variance_result = pd.DataFrame(columns=['neigh','no_of_mod
        print('Builds ', estimator_name)
        print('Total No of models built is ',str(noOfModels))

        samplePredictionsModel = pd.DataFrame()

        with tqdm.tqdm(total=noOfModels, file=sys.stdout) as pbar:
        for i in range(0, noOfModels):
```

```
            pbar.set_description('Building Model : %d' % (1 + i))
            sample = train_data.sample(sampleCount,replace=False)
            X_train = sample.drop(['F9'], axis=1)
            y_train = sample['F9']

            samplePredictionsModel['sample'+str(i+1)] = samplePredForEstima
            pbar.update(1)

            var_mse_linear_model = calculate_variance_of_model(samplePredic

            bias_linear_model = calculate_bias_of_model(samplePredictionsMo

            s_linear = pd.Series(data={'neigh':n_neigh, 'no_of_models':noO
                    'bias':bias_linear_model, 'variance' : var_mse_linear_m

            bias_variance_result = bias_variance_result.append(s_linear)

            bias_variance_result.reset_index(inplace=True)
            bias_variance_result.drop(['index'], axis=1, inplace=True)

            return bias_variance_result

    def bias_variance_mse_graphics(noOfSamples, noOfModels, estimator_name,
            lenNoOfNeighbors = len(noOfNeighbors)
            bias_variance_result = pd.DataFrame()
            print('Building Model for samples ', noOfSamples)
            for i in range(0, lenNoOfNeighbors):
            noOfNeighbor = noOfNeighbors[i]
            print('Building models with k neighbors : ', noOfNeighbor)
            bias_variance_result = bias_variance_result.append(get_bias_var
            bias_variance_result.reset_index(inplace=True)
            bias_variance_result.drop(['index'], axis=1, inplace=True)
            print(bias_variance_result)

            # Plotting the obtained results
            fig, ax = plt.subplots(1,2, figsize = (14,6))
            ax[0].plot(bias_variance_result.neigh, bias_variance_result.bia
            ax[0].plot(bias_variance_result.neigh, bias_variance_result.var
            ax[0].legend(['bias','variance'])
            ax[0].set_xlabel('k')

            ax[1].plot(bias_variance_result.neigh, bias_variance_result.mse
            ax[1].legend(['mse'])
            ax[1].set_xlabel('k')
            plt.show()

    noOfNeighbors = [1,5,10,15,20,25,50,75,100,150,200,250,500,750,1000,200
    bias_variance_mse_graphics(40000, 20, 'KNN', noOfNeighbors)
```

Références

[1] Belkin M., Hsu D., Ma S., Mandal S., Reconciling modern machine learning practice and the bias-variance trade-off *arXiv :1812.11118v2*, November 1-4, 2015, pp. 337-350.

[2] Fernandes de Mello R., Antonelli Ponti M., *Machine Learning A practical Approach on the Statistical Learning Theory*, Springer, Cham, 2018.

[3] Geman S., Bienenstock E., Doursat R., Neural Networks and the Bias/-Variance Dilemma *Neural Computation* 4, 1-58, 1992 `http://direct.mit.edu/neco/article-pdf/4/1/1/812244/neco.1992.4.1.1.pdf`

[4] Neal B., On the Bias-Variance Tradeoff : Textbooks Need an Update *arXiv :1912.08286v1*, December 2019

[5] Shalev-Shwartz S., Ben-David S., *Understanding Machine Learning From Theory to Algorithms*, Cambridge University Press, 2019 (12th printing).

[6] `http://scott.fortmann-roe.com/docs/BiasVariance.html`, consulté le 18 Juin 2021 à 09 :25

[7] `https://elitedatascience.com/bias-variance-tradeoff`, consulté le 21 Juin 2021 à 10 :16

[8] `https://github.com/sayanam/MachineLearning/blob/master/ExperimentationWithBiasAndVariance/BiasAndVariance_V2.ipynb`, consulté le 15 Juillet 2021 à 18 :20

Table des figures