Lamoureux Léo 21602944
Pires Pinto Lola 21706287

ALGO : RANDOM FOREST

Etape 3

Nous allons nous concentrer sur les hyper-paramètres suivants :

1) n_estimators
2) max_depth
3) min_samples_split
4) min_samples_leaf
5) max_leaf_nodes
6) max_samples
7) max_features
8) min_weight_fraction_leaf
9) min_impurity_decrease

Définitions dont nous aurons besoin pour la suite :
Noeud pur: ***un noeud est pur à 100% lorsque toutes ses données appartiennent à une seule classe.***
Noeud impur: ***un noeud est 100% impur lorsqu'un il est divisé uniformément = 50% de données de chaque classe.***

Pour chaque valeur des paramètres, nous avons effectué 10 tests puis retenu les moyennes des précisions obtenues, pour avoir une courbe plus précise et s'assurer de ne pas avoir obtenu une valeur aberrante.
Nous testons l'algorithme sur les données sur lesquelles il à été entraîné et sur les données qu'il ne connaît pas encore (celles de test), pour vérifier la cohérence des résultats.

Dans la plupart des cas où nous parlons de sur-ajustement, la précision sur les données de test devrait logiquement chuter avec l'augmentation de celle sur les données de train.
Or, ce n'est pas toujours le cas dans nos analyses.
Plusieurs explication possibles :

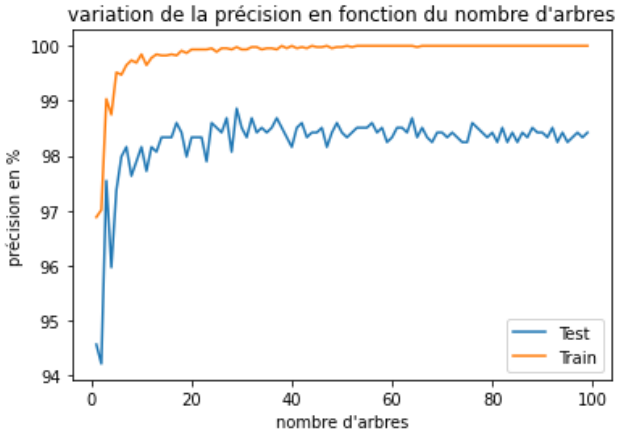
- Les exemples des données du test existent également dans le train-set. Le modèle sera donc performant.
- Le problème est trop facile pour le modèle, il est donc performant. (cela peut paraître logique car les datasets sont fournis avec les algorithmes sur scikit-learn, ils sont donc peut-être optimisés). Cette solution s'applique plutôt au dataset de classification, car la précision en régression est assez basse.

Les exécutions des codes pour obtenir les courbes peuvent être longues , installez vous confortablement si vous voulez tout recompiler :)

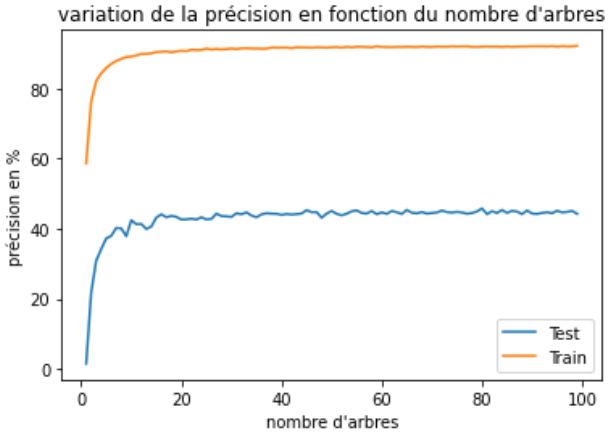
1) n_estimators

Il s'agit du nombre d'arbres utilisés dans la forêt.

Classification



Régression



Pour un petit nombre d'arbres (< 10) la précision du modèle est légèrement plus faible. Ce qui est logique car moins il y a d'arbres, moins il y a de prédictions faites sur différentes parties du dataset donc moins le résultat sera précis.
En augmentant le nombre d'arbres, la précision du modèle augmente et à partir d'une quinzaine d'arbres environ, la précision oscille autour de la moyenne. On ne gagne plus rien à augmenter la valeur. En effet, chaque arbre se concentrant sur une vision du problème, plus il y a d'arbres, plus ils risquent de traiter les mêmes données

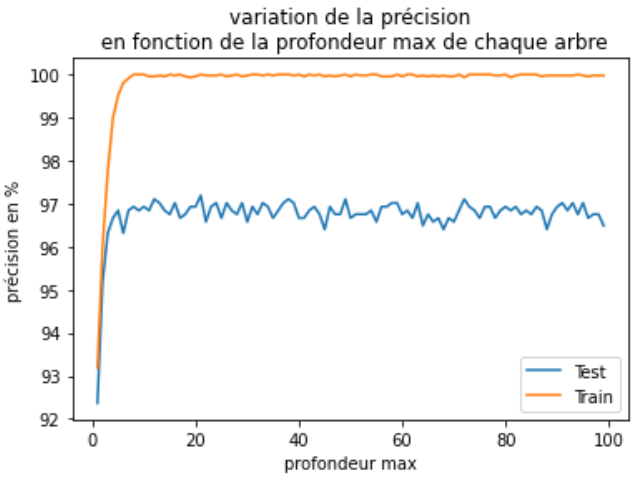
(max-samples n'est pas défini ici) et plus il y a de chance qu'ils effectuent les mêmes estimations. Ce qui n'apporte rien au modèle et ne ferait que le ralentir.

Pour conclure, un nombre trop petit d'arbres impacte de manière négative la précision du modèle mais utiliser toujours de plus en plus d'arbres est aussi inutile.

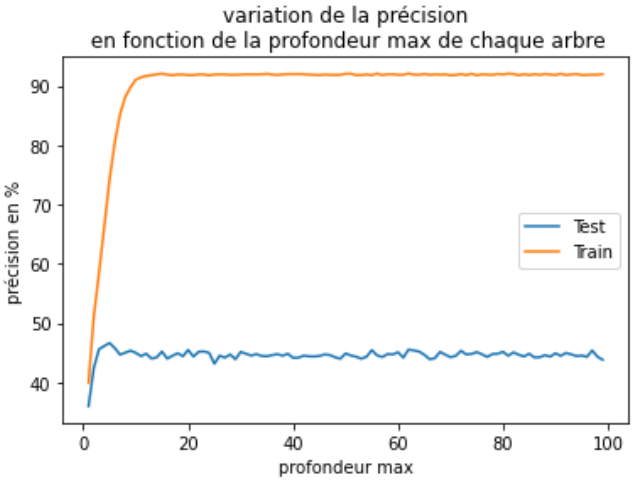
2) max_depth

Il s'agit de la profondeur maximale des arbres utilisés. (la hauteur maximale de chaque arbre de décision). Si le paramètre n'est pas explicité, chaque arbre arrête de se développer lorsque toutes ses feuilles sont pures.

Classification



Régression



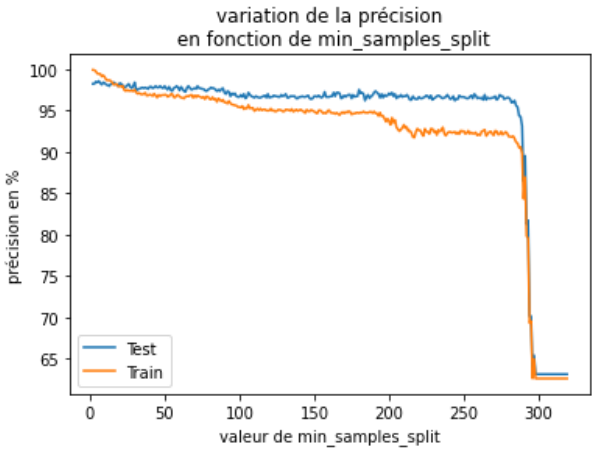
Pour une profondeur faible (< 10) la précision du modèle est légèrement plus faible que la moyenne.
Pour être performant l'algorithme a besoin d'une profondeur qui ne soit pas trop faible. Ce qui semble logique car plus la profondeur est élevée, plus l'arbre traite d'informations sur les données, donc plus la prédiction est précise.
En augmentant cette profondeur, la précision augmente, car l'arbre peut diviser ses noeuds comme il le souhaite de manière à ce que chaque feuille soit pure dans le résultat final.

La précision se stabilise ensuite et n'évolue plus au-delà d'un certain point.
Il est possible qu'en atteignant une certaine profondeur, l'arbre se spécialise trop (les critères de divisions deviennent trop spécifiques) et qu'il n'arrive donc plus à généraliser les classes de données.
Par exemple, l'arbre pourrait diviser des noeuds déjà purs, sur des critères inutiles par rapport au problème (trop précis).
C'est ce qu'on appelle le sur-ajustement.
Ceci pourrait expliquer que la précision sur les données du train-set soit aussi élevée. Dans ce cas, on pourrait s'attendre à ce que la précision sur les données de test chute, or ici elle reste plutôt constante.

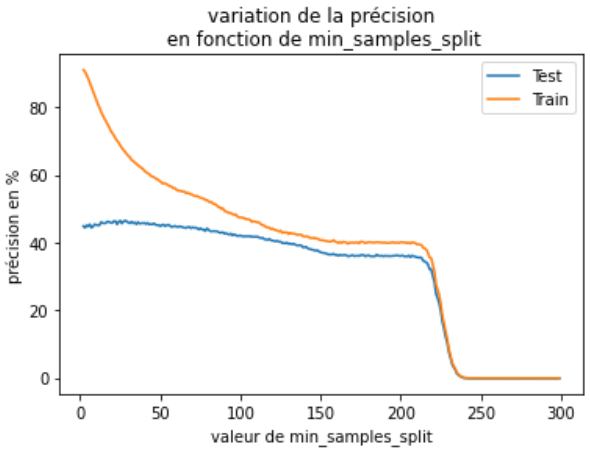
3) min_samples_split

Min_sample_split correspond au nombre minimum d'échantillons requis dans un noeud pour que celui-ci puisse se diviser à nouveau. En augmentant sa valeur, nous pouvons donc réduire le nombre de fractionnement qui se produisent dans chaque arbre de décision.
Dans certains cas, cela peut aider à prévenir le sur-ajustement du modèle en l'empêchant de trop se spécialiser (si il se divise trop, les critères de division deviendront trop précis).

Classification



Régression



Pour une valeur faible de ce paramètre, le modèle semble être en sur-ajustement. Cela paraît logique car le nombre de fractionnement est beaucoup plus élevé pour une valeur faible de min_sample_split.

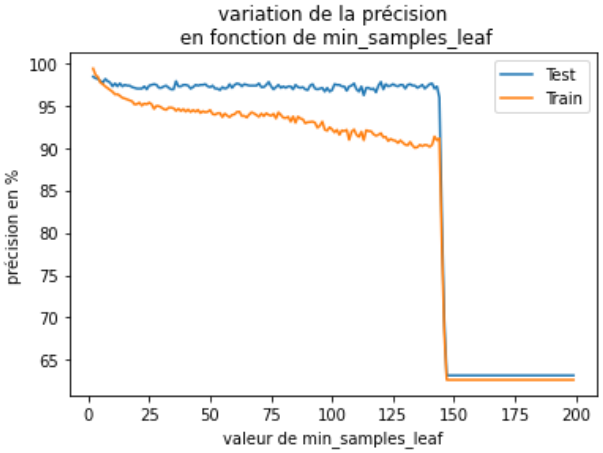
L’arbre se spécifie donc beaucoup. (on peut voir que la précision pour le train_set est maximale.)
Au fur et à mesure que la valeur du paramètre augmente, le modèle semble réajuster ce problème de sur-ajustement (il se spécifie moins), et la précision sur le test-set se stabilise.

À partir d’un certain point, la précision chute drastiquement car l’exigence minimale de fractionnement d’un noeud est devenue trop élevée et les arbres ne se divisent plus assez pour que le modèle puisse s’ajuster sur les données.
C’est un cas de sous-ajustement.
Ceci peut-être dû au fait qu’un noeud ne contenant pas assez d’échantillons pour se diviser deviendra donc une feuille même s'il est impur.

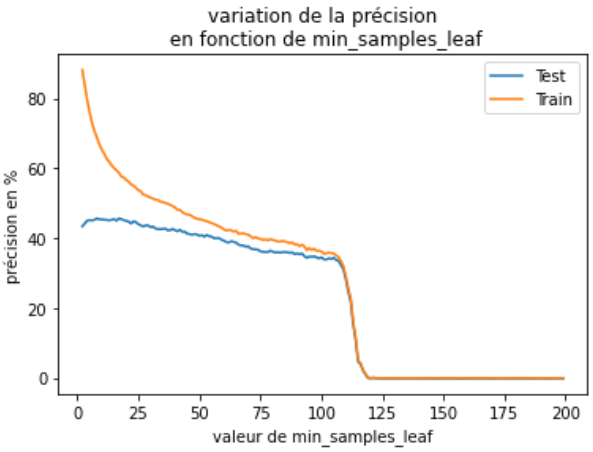
4) min_samples_leaf

Ce paramètre correspond au nombre minimum d'échantillons requis pour créer un noeud feuille. Cela oblige l'arbre à ne garder que des noeuds avec un certain nombre d’échantillons et permet donc de conditionner sa décision finale.

Classification



Régression



Tout comme min_sample_split, le modèle est en sur-ajustement lorsque la valeur du paramètre est faible (les arbres se divisent beaucoup, ils peuvent donc trop se spécialiser). Le modèle corrige également ce sur-ajustement au fur et à mesure que la valeur de min_sample_leaf augmente (on voit que la précision du test-set se stabilise et que celle du train-set diminue).

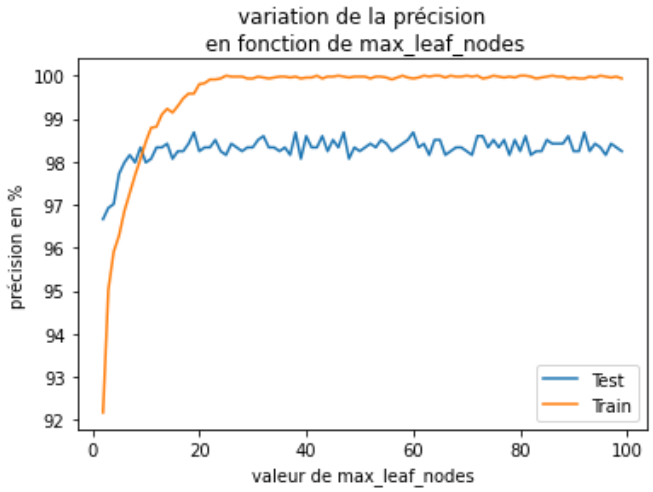
On observe, tout comme pour le min_sample_split, une chute de précision.
Par défaut, l'arbre développe ses feuilles jusqu'à ce qu'elles soient pures. Or en le forçant à avoir trop d'échantillons pour créer une feuille, beaucoup de feuilles, pourtant pures, ne seront pas gardées dans le résultat final car elles ont trop peu d'échantillons. On gardera son noeud père, qui est généralement moins pur (voir impur). L'algo sera donc moins précis.

À partir d'un certain point le modèle ne pourra plus s'adapter au dataset (chute de la précision). C'est encore un cas de sous-ajustement.

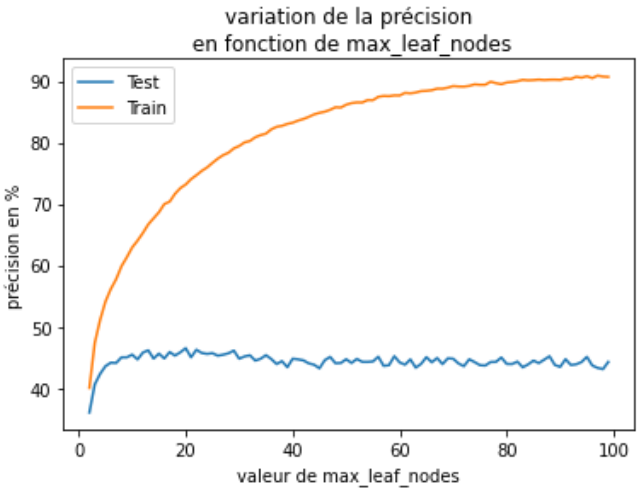
5) max_leaf_nodes

Il s'agit du nombre maximal de feuilles par arbre.
Ce paramètre permet de restreindre la croissance de chaque arbre en conditionnant la division d'un noeud. En effet, un noeud ne peut se fractionner que si le nombre de feuilles n'a pas atteint le *max_leaf_nodes* défini.

Classification



Régression



On constate que la précision du modèle est plus basse pour un nombre maximal de feuilles très petit : peut-être que l'arbre ne peut pas se développer assez pour que le modèle s'ajuste au mieux sur les

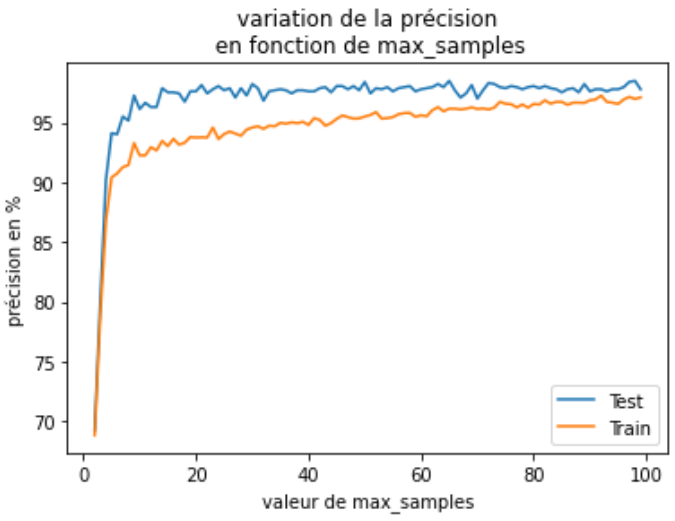
données. Dans ce cas, des feuilles prises en compte (des noeuds n'ayant pas pu se diviser) seront moins pures. C'est peut-être un cas de sous-ajustement.

Au fur et à mesure de l'augmentation du nombre de feuilles, la précision augmente puis stagne autour de la moyenne. Les arbres peuvent développer assez de feuilles pour permettre un meilleur ajustement.
On voit néanmoins que la précision du train-set évolue de plus en plus vers un cas de sur-ajustement.
Cela peut-être dû au fait que les arbres commencent à trop se fractionner et se spécialiser. Dans ce cas, on pourrait s'attendre à ce que la précision du test chute.

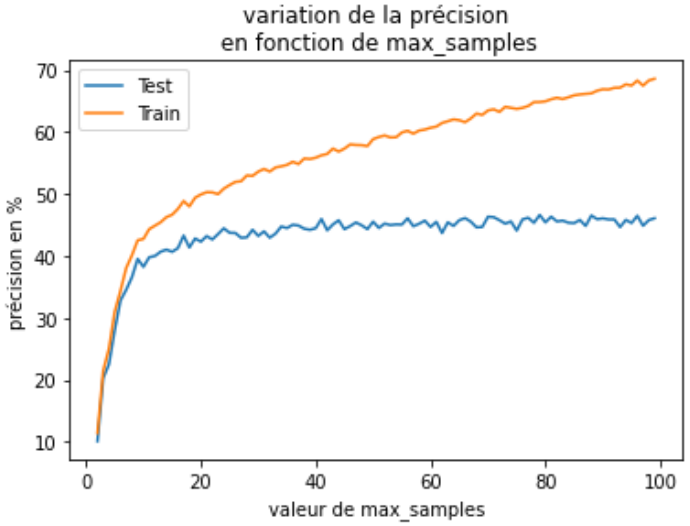
6) max_samples

Max_samples correspond à la fraction de données du dataset qui sera utilisée par chaque arbre de décision.

Classification



Régression



On peut observer que la précision commence à augmenter fortement puis stagne lorsque la valeur de *max_samples* est environ à 10%.
Cela peut paraître contre intuitif car on peut penser que plus les arbres possèdent de données, plus le modèle sera précis.

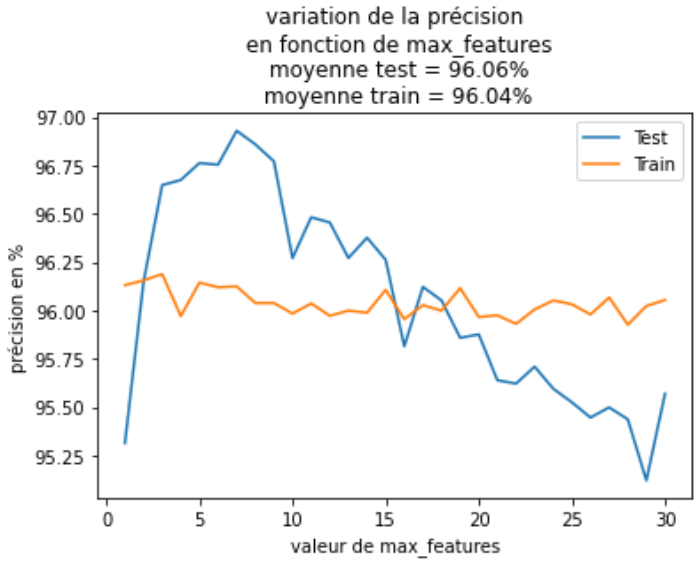
Mais il s'avère selon notre graphe qu'environ 10-15% seulement de données des datasets par arbre suffisent pour obtenir une bonne précision.
Cela peut aussi sembler logique car si chaque arbre utilisait une trop grande partie des données du dataset, ils contiendraient chacun des données aussi contenues dans les autres arbres. Ces données auraient besoin d'être prises en compte que par un seul arbre, cette redondance n'apporte rien au résultat final du modèle.

On peut en conclure qu'il vaut mieux ne pas donner toutes les données du dataset à chaque arbre de décision mais seulement une partition équitable. En effet, donner une trop grande partie des données à chaque arbre ne ferait que ralentir l'algorithme.

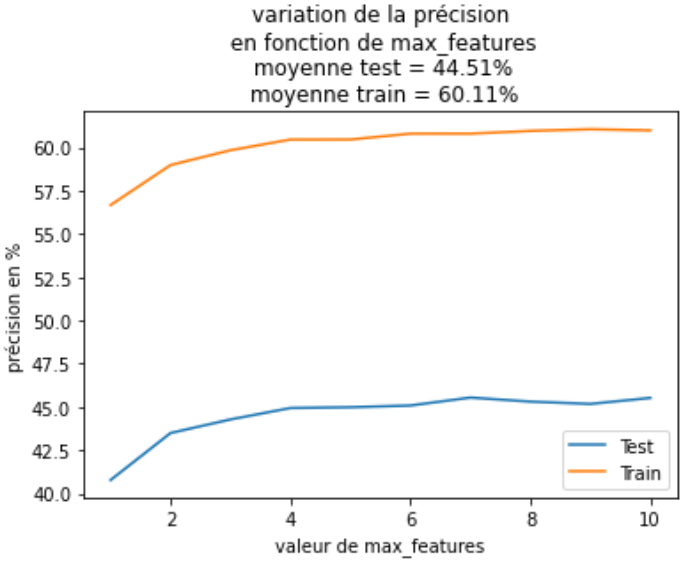
7) max_features

Chaque fractionnement est effectué à partir de certaines caractéristiques (= features, correspond aux colonnes du dataset) sélectionnées au préalable. *Max_features* correspond au nombre imposé de caractéristiques à choisir par arbre.

Classification



Régression



Plus on augmente la valeur de max_features plus chaque arbre aura d'options à considérer.

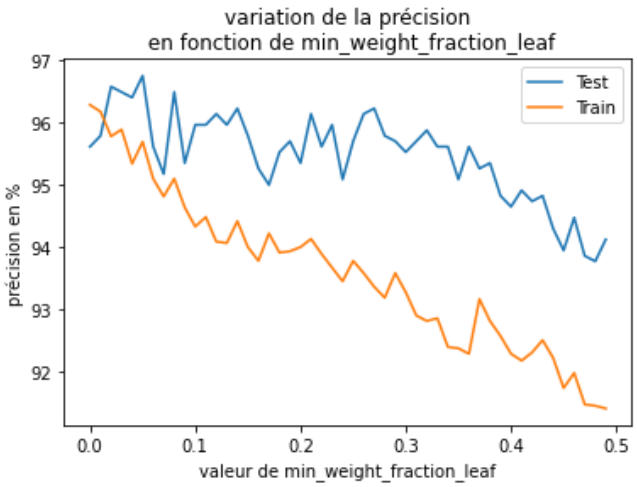
On peut donc penser que la précision augmentera avec. Or Si tous les arbres considèrent trop d'option ils auront des options en commun. La forêt sera donc moins diversifiée, on retrouve une redondance qui n'apporte rien à l'algorithme et peut même impacter de manière négative les performances.
On s'attend donc à ce que la précision du modèle commence par augmenter (avec une valeur faible de max_features qui commence à augmenter), puis stagne, voire diminue quand cette valeur devient trop grande. C'est ce qu'on observe pour la classification.

Il faut donc équilibrer le nombre d'options à prendre en compte dans chaque arbre pour garder une forêt assez diversifiée pour obtenir une bonne précision et garder un algorithme rapide.

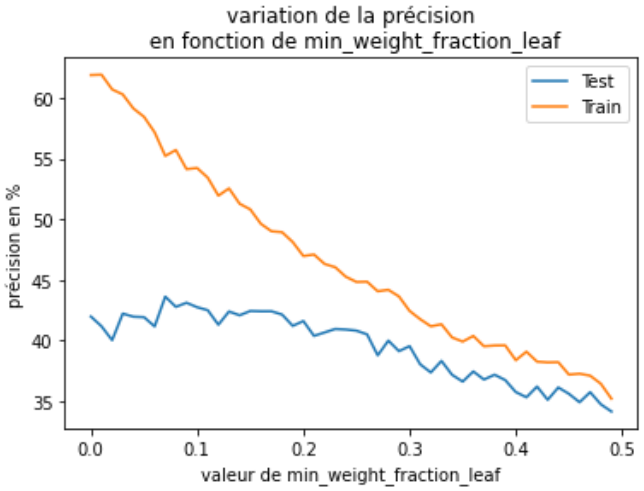
8) min_weight_fraction_leaf

C'est la fraction du nombre total d'échantillon minimal pour créer une feuille.

Classification



Régression



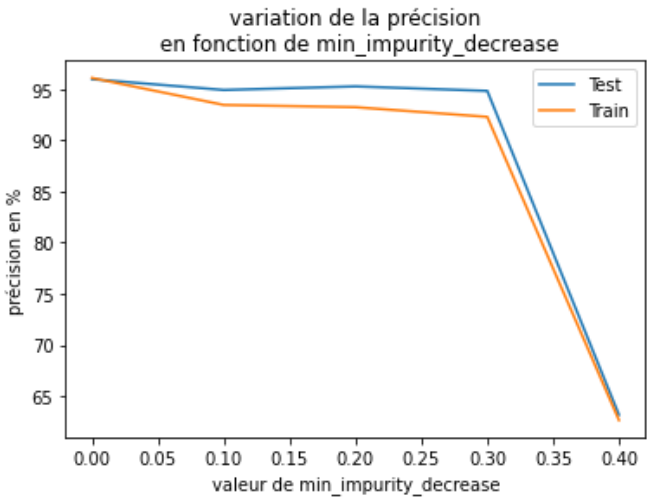
Au fur et à mesure de l'augmentation de la valeur du paramètre, on observe une baisse de précision des modèles.
Plus la fraction minimale d'échantillons pour être une feuille est élevée, plus la précision diminue. Ce qui peut paraître logique car ici encore on conditionne le fractionnement d'un noeud.

Il semblerait que la valeur par défaut à 0.0 soit la valeur optimale. Il est donc préférable de ne pas imposer de fraction minimale.

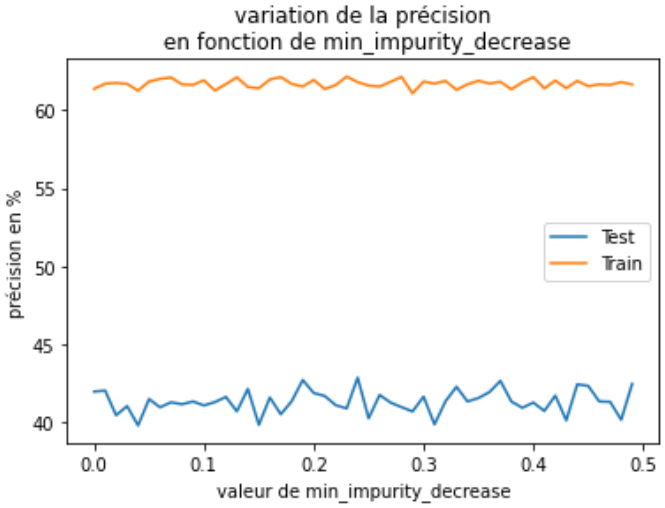
9) min_impurity_decrease

Ce paramètre conditionne le fractionnement d'un noeud. En effet, un noeud ne peut se diviser que si la différence d'impureté entre les noeuds pères et fils est inférieure ou égale à min_impurity_decrease.

Classification



Régression



Pour la classification :
Tout comme les paramètres précédents, on observe une chute brutale de la précision lorsque l'exigence minimale pour le fractionnement d'un noeud devient trop élevée.
Pour la régression :
L'augmentation de la valeur de min_impurity_decrease ne semble pas affecter le modèle de régression. En effet, la précision reste plus ou moins stable tout le long.

Ces deux derniers paramètres ne semblent pas très intéressant à faire varier.

Autres paramètres :

Les paramètres suivants n'ont pas d'impact direct sur la précision de nos modèles. Ils sont utiles si on veut accélérer leurs formations.

n_jobs : Il indique le nombre de processus exécutables en parallèle.

warm_start : Réutilise le résultat du dernier apprentissage. Permet de gagner du temps dans les procédures d'ajustement.

ccp_alpha : Paramètre pour l'élagage. Plus il est élevé, plus le nombre de noeuds élagués augmente. Il permet de diminuer le coût-complexité.

random_state : C'est une graine. Pour une valeur définie, on obtient les mêmes résultats si elle est donnée avec les mêmes paramètres et les mêmes données d'entraînement. Ce paramètre rend donc une solution facile à répliquer.

Autres :

criterion : Permet de mesurer la qualité d'un fractionnement.
Pour la classification :
Deux valeurs possibles : "gini" ou "entropy".

- "Gini" se base sur l'impureté d'une nœud
- "Entropy" se base sur le gain d'information.

Pour la régression :
Deux valeurs possibles :

- "squarred_error"
- "absolute_error".

Soit la MAE et la MSE, les deux métriques que nous avons étudiées à l'étape 2.

bootstrap : (booléen) Paramètre utilisé pour la construction des arbres. S'il est à *False*, toutes les données du dataset seront

utilisées pour chaque arbre. par défaut à *True* (nous permet de faire varier d'autres paramètres).

oob_score : (booléen) Ce paramètre est un moyen d'évaluer le modèle de Random Forest. Il est calculé à partir du nombre de lignes correctement prédites des échantillons "hors sac". Ces échantillons sont ceux qui sont non contenus dans les échantillons "boostrap".

verbose : Ce paramètre permet de contrôler la verbosité lors de l'ajustement et la prédiction. En le définissant sur un nombre plus élevé, on peut ainsi voir plus d'informations sur le processus de construction des arbres.

class_weight : Seulement pour la classification, associe des poids d'importance à chaque classe. Nous pourrions l'utiliser pour affecter un poids plus important à la classe représentant un cancer "Malin", car les erreurs sur des cas de cancer avérés sont plus graves.