Rendu TP Arbres de décision, forêts aléatoires, bagging et boosting

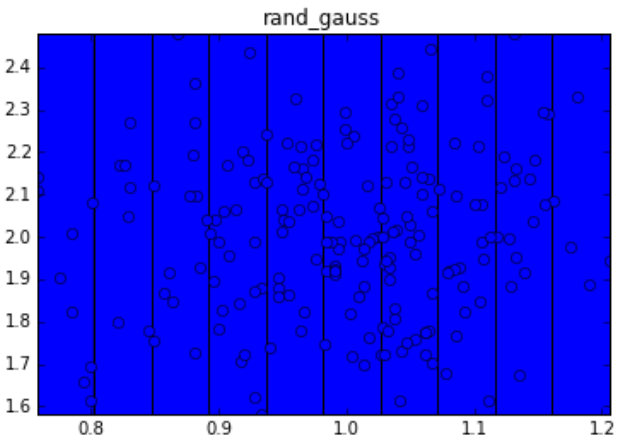
Aliréza BANAEI

1- Dans le cadre de la régression (i.e. quand on cherche à prédire une valeur continue et non discrète), proposez une mesure d'impureté. Justifiez votre choix.

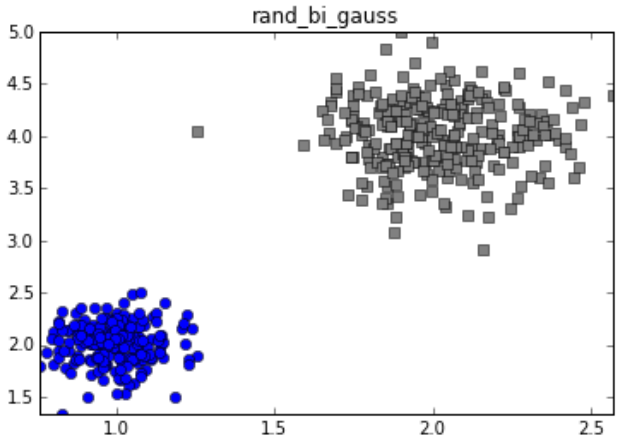
La variance (de Y) mesure la dispersion des données autour de la moyenne et peut être une bonne mesure de la pureté de l'échantillon, dans le mesure où elle est nulle si toutes les données ont la même valeur, et d'autant plus grande qu'elles sont dispersées. C'est donc la variance de Y.

2- Le fichier utils.py contient des fonctions de génération de données synthétiques et d'affichage en deux dimensions. Familiarisez-vous avec ces fonctions et affichez différents jeux de données avec des paramètres personnalisés.

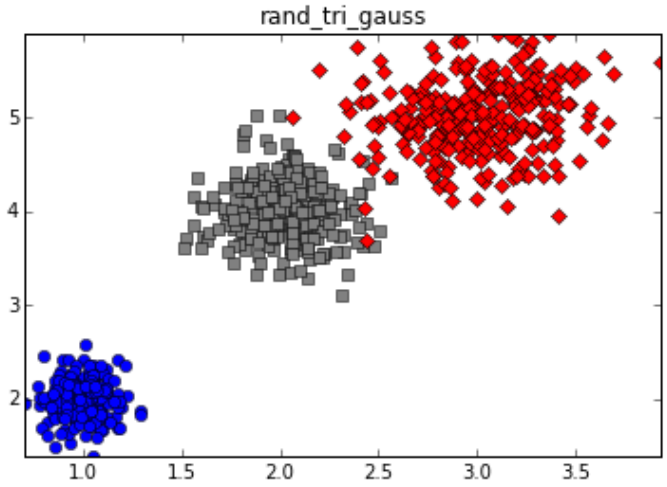
Utilisation de la fonction rand_gauss :



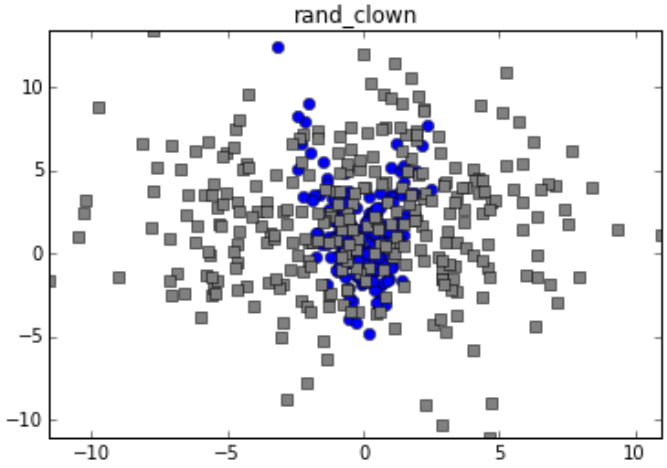
Utilisation de la fonction rand_bi_gauss :



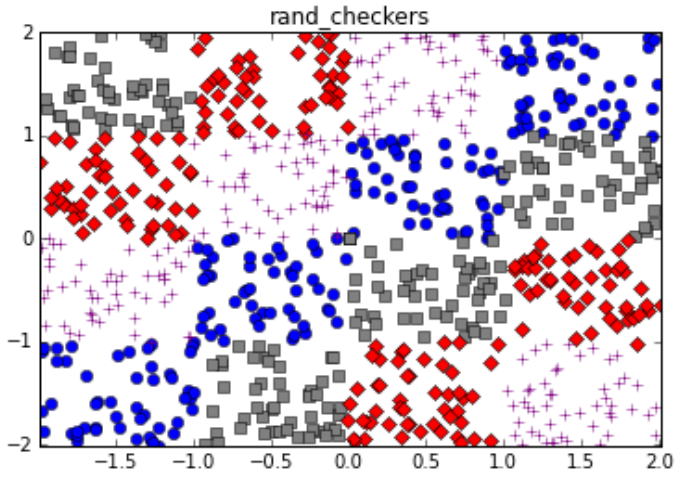
Utilisation de la fonction rand_tri_gauss :



Utilisation de la fonction rand_clown :

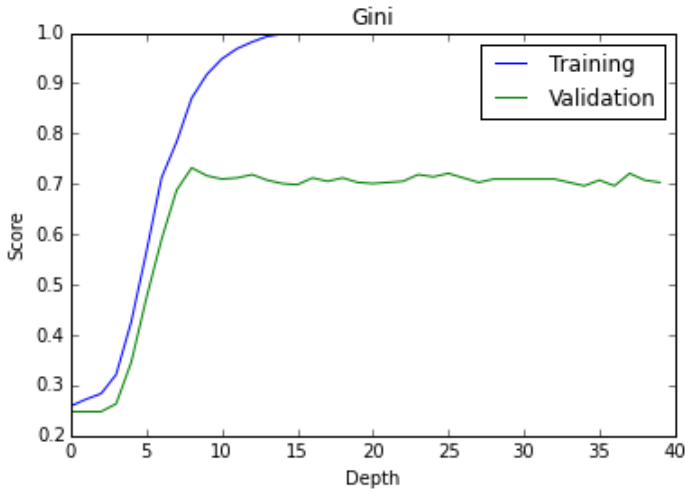


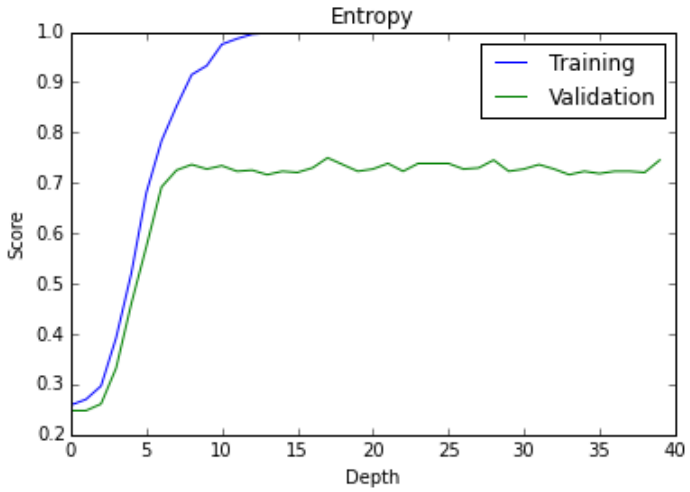
Utilisation de la fonction rand_checkers :



3- Simuler avec rand_checkers un échantillon d'apprentissage de taille n = 456 et un échantillon de test de même taille (attention à bien équilibrer les classes). Créer

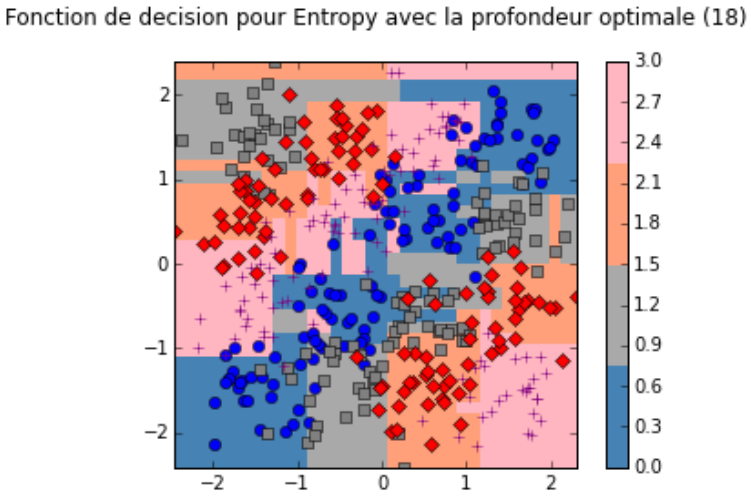
quatre courbes qui donnent le pourcentage d'erreurs commises (en apprentissage et en test) en fonction de la profondeur maximale de l'arbre (deux courbes pour l'indice de Gini et deux autres pour l'entropie).





4- Affcher la classification résultant de l'arbre entraîné en utilisant la profondeur qui minimise l'erreur de test obtenue avec l'entropie (utiliser les fonctions plot_2d et frontiere).

```
Score for validation set = 0.738839285714
```



5- À l'aide du code ci-dessous, exporter un graphique de l'arbre clf obtenu à la question précédente (nécessite l'installation de Graphviz – paquet graphviz).

Voir le fichier my_tree.pdf joint.

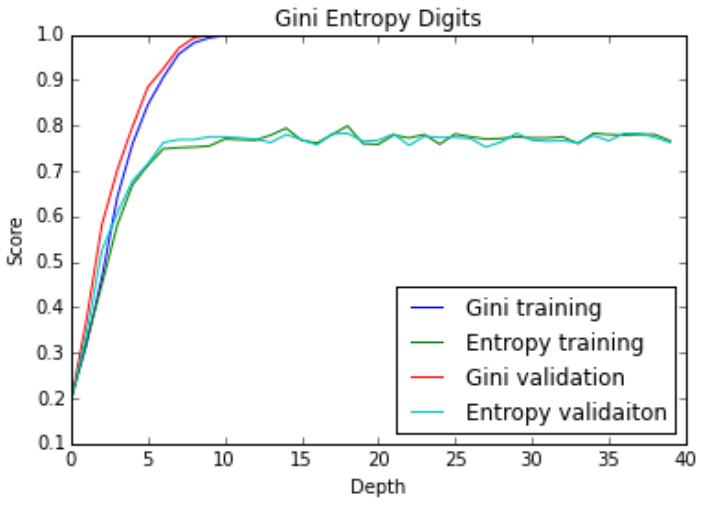
6- Créez n = 200 nouvelles données avec rand_checkers. Pour l'arbre de décision obtenu en question 4, calculer la proportion d'erreurs faites sur ce second échantillon. Commenter.

```
Score for a new validation set = 0.75
```

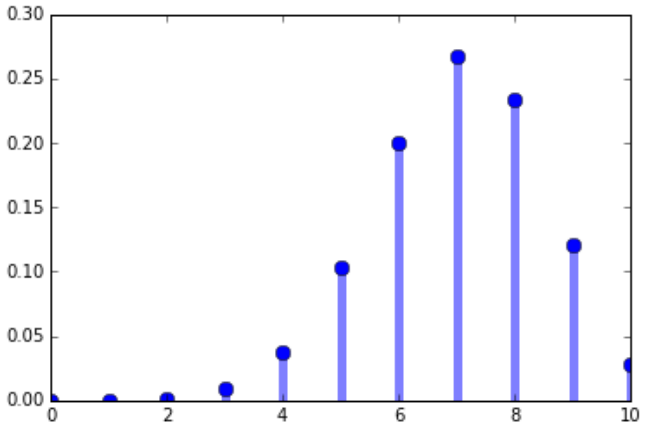
J'ai exécuté ce code plusieurs fois ce code et j'obtiens des scores très différents allant de 0.63 à 0.75. Là j'ai mis le score le plus élevé. Ceci montre la sensibilité de l'arbre de décision aux données et sa grande variation en fonction de celles-ci.

7- Reprendre la question 3 pour le jeu de données digits, disponible dans le module sklearn.datasets

Voici ce que ça donne :



8- En reprenant l'exemple des classifieurs binaires donné ci-dessus (avec f ` : X → {−1, +1}), on suppose que l'on dispose de m = 10 modèles ayant une probabilité p = 0.7 de prédire correctement (le hasard a une probabilité 0.5 de fournir un résultat correct ; les modèles sont donc légèrement meilleurs que le hasard). Exécuter le code suivant est interpréter le graphique.

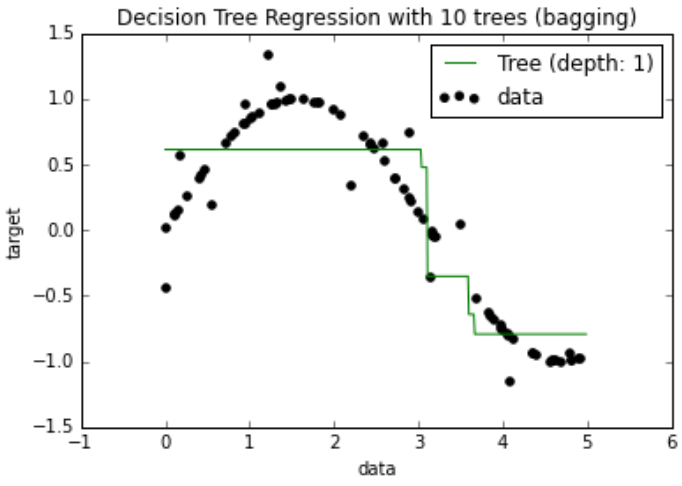


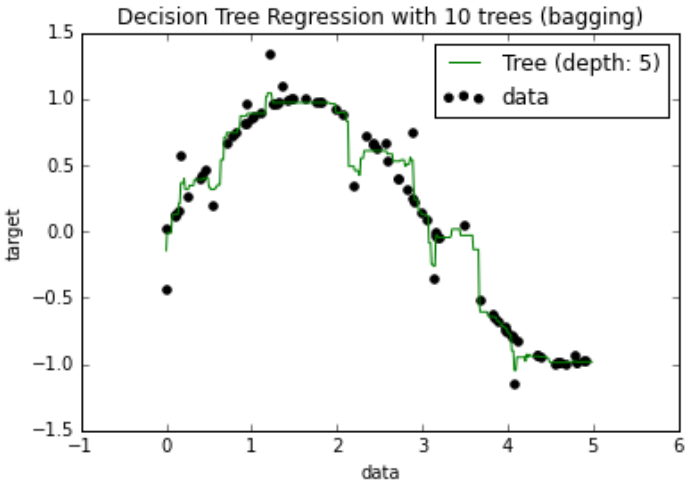
Calculer (numériquement) la probabilité que le modèle agrégé uniformément (a l = 1/m) donne un résultat correct. On remarquera que cette probabilité est donnée par le nombre X d'issus positifs …

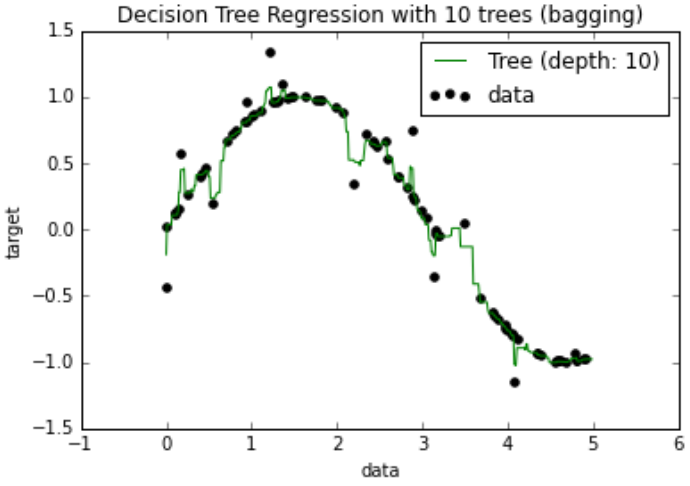
```
Probabilité individuelle = 0.7
Probabilité agrégée = 0.90119134
```

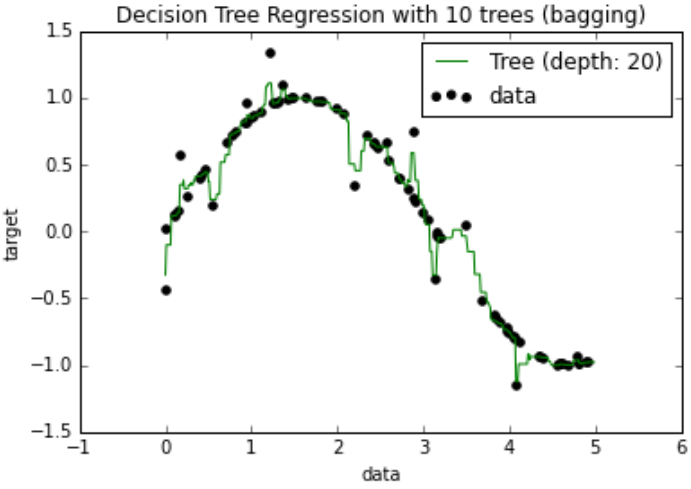
La somme de plusieurs modèles ayant une proba de bonne réponse de 0.7 donne une proba de bonne réponse de 0.90.

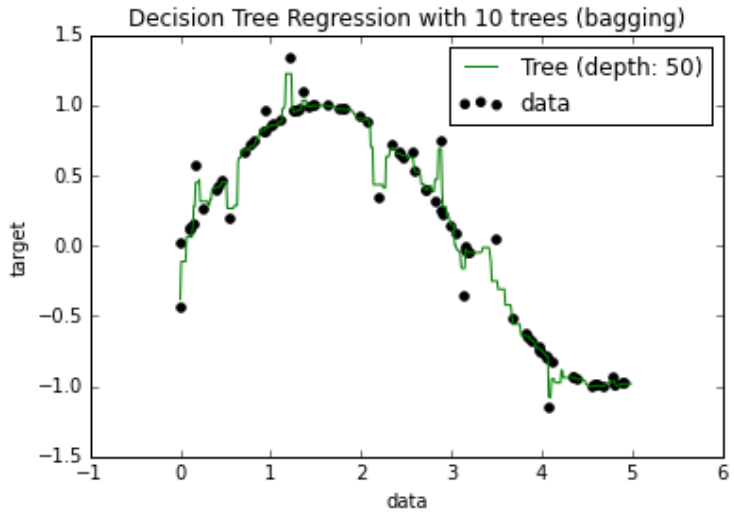
9- Écrire un script mettant en œuvre le bagging avec des arbres souches (i.e. de profondeur 1, aussi appelés tree stumps), puis avec des arbres plus profonds. On partira du code ci-dessous. On pourra utiliser la fonction np.random.randint pour générer les échantillons Bootstrap.




Decision Tree Regression with 10 trees (bagging)
Tree (depth: 5)
data
target
data

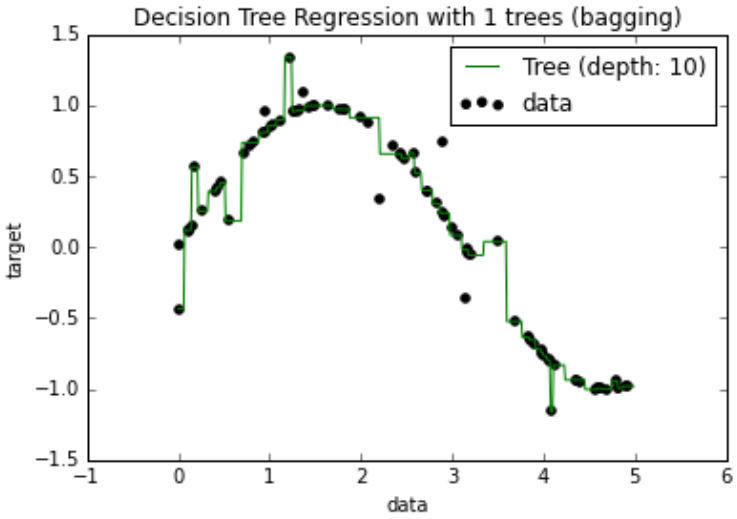

Decision Tree Regression with 10 trees (bagging)
Tree (depth: 10)
data
target
data

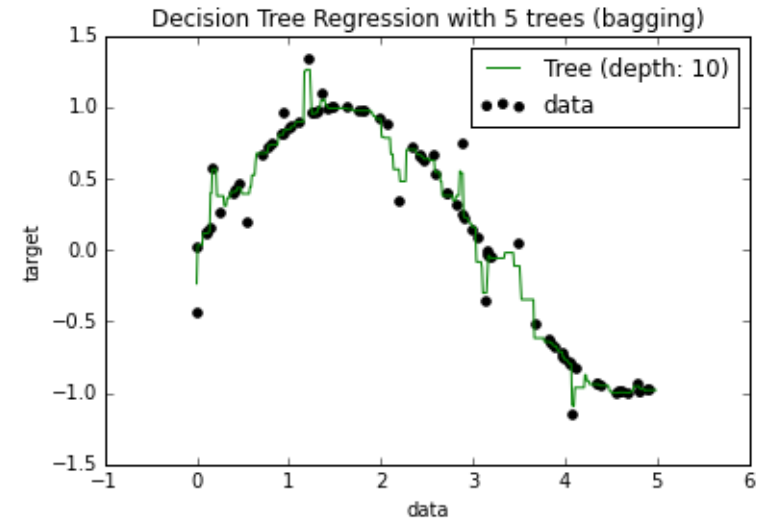

Decision Tree Regression with 10 trees (bagging)
Tree (depth: 20)
data
target
data

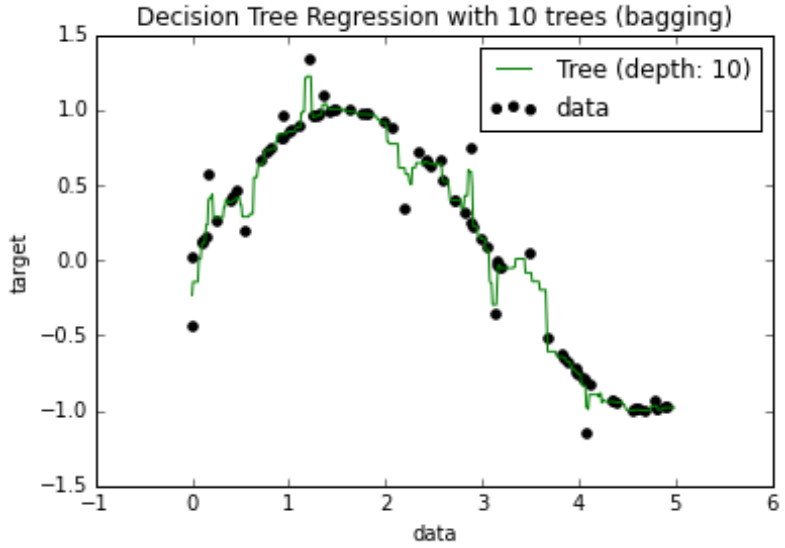


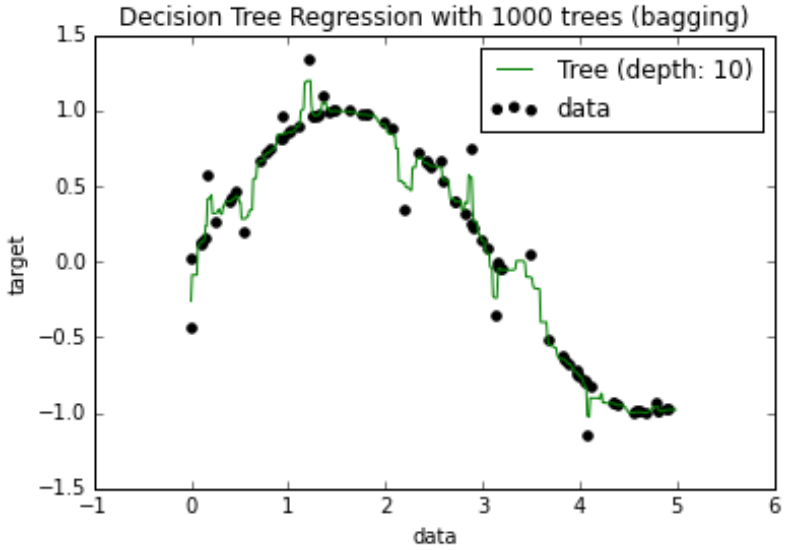
10- Observer le rôle de m ainsi que de la profondeur des arbres (max_depth).

On remarque un overfitting pour des profondeurs d'arbre au-delà de 10. Pour la même profondeur (10), le nombre d'arbres (bagging) rend le résultat de moins en moins sensible au bruit :







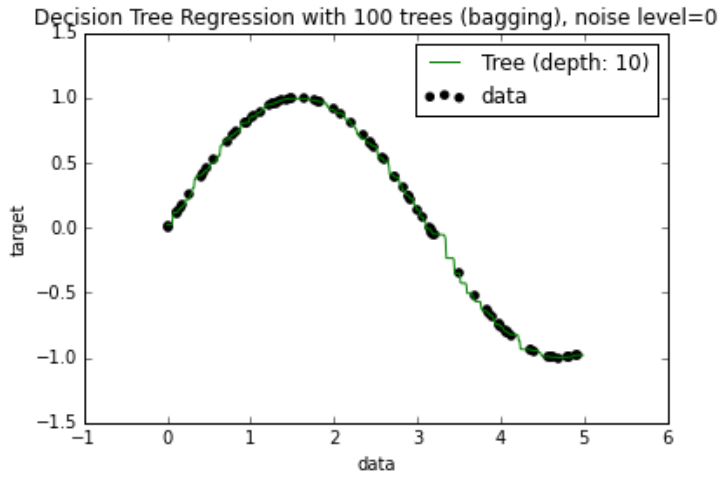


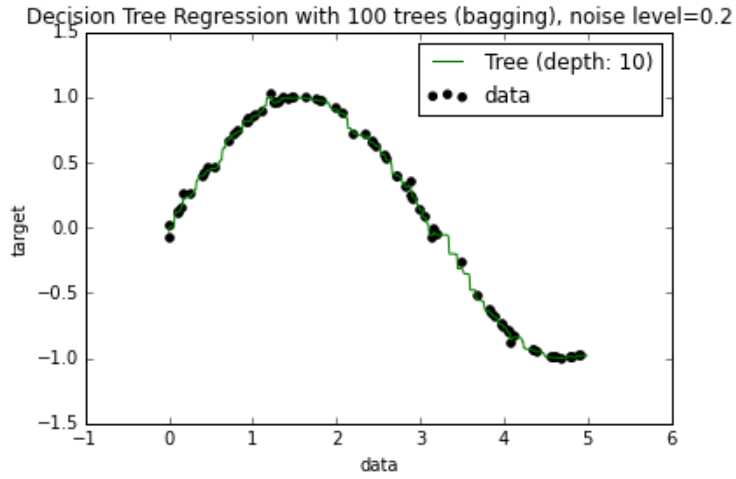
11- A quoi reconnaît-on que les estimateurs construits par les arbres sont biaisés et que le bagging réduit leur variance ?

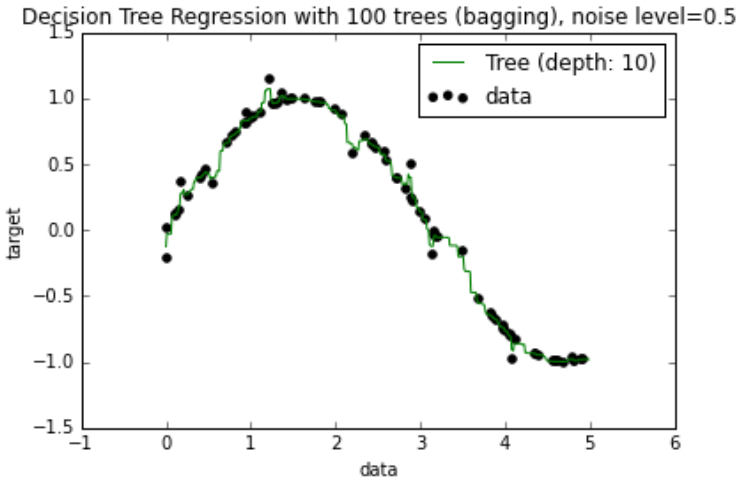
A la diminution de la sensibilité au bruit (les valeurs out-sider).

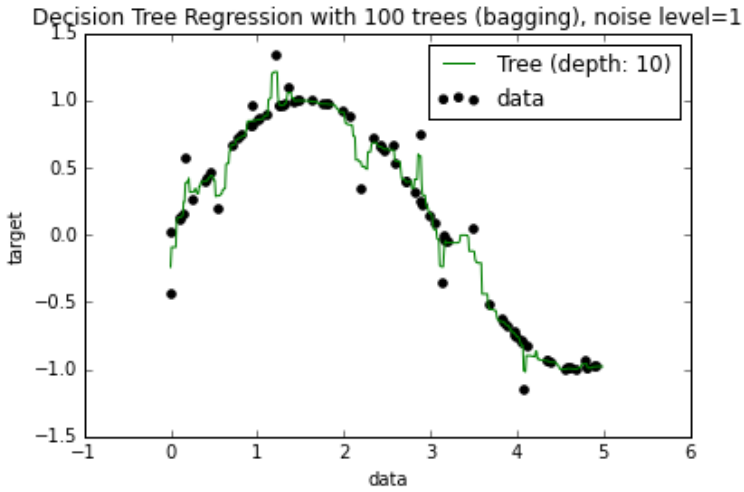
12- En jouant sur le niveau de bruit, mettre en évidence le sur-apprentissage.

Voici les résultats pour 4 niveaux de bruit (0, 0.2, 0.5, 1). Ce niveau de bruit correspond à un coefficient multipliant une variable aléatoire entre -0.5 et 0.5 qui s'ajoute aux données.



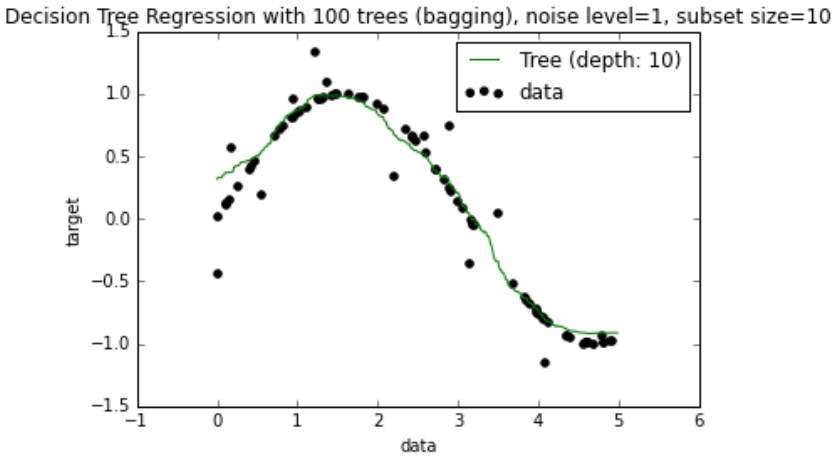


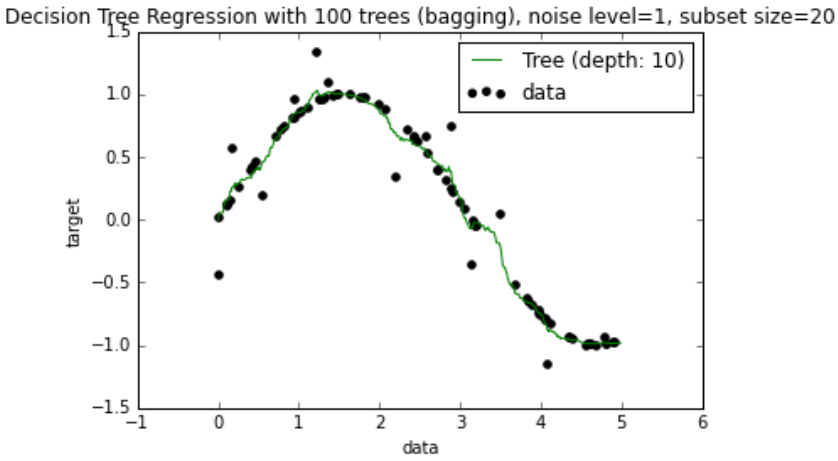


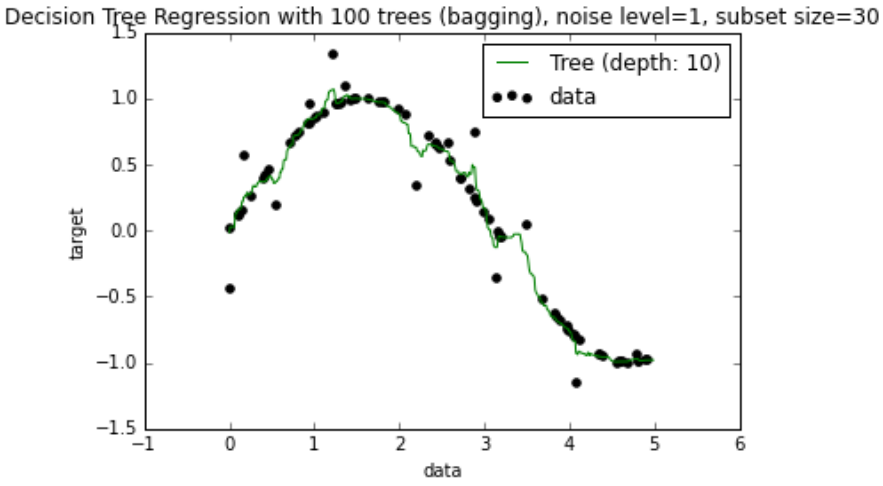


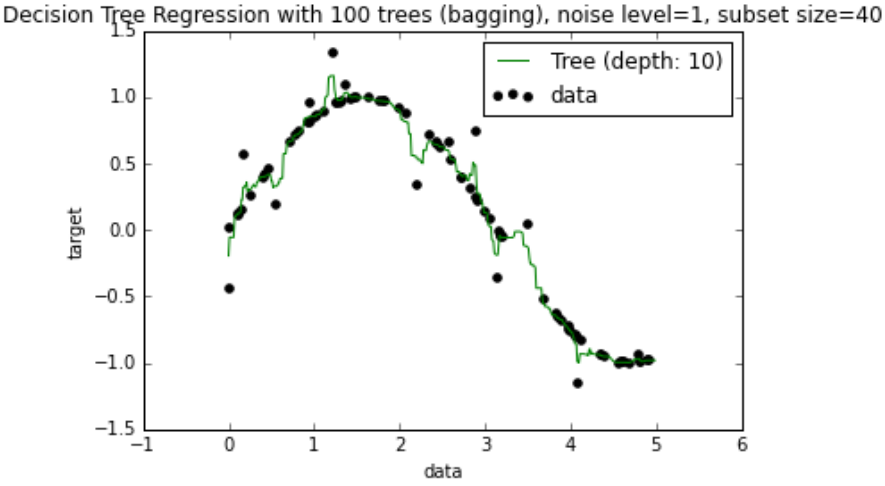
13- Observer qu'on peut réduire ce phénomène en échantillonnant aléatoirement et sans remise au lieu de prendre des échantillons bootstrap.

En prenant des sous-échantillons de plus en plus grands (subset size) sachant que l'effectif des données est de 80, avec un niveau de bruit maximal (1), on commence à observer l'overfitting à partir de 40 (moitié des données) :









14- Compléter le script ci-dessous afin d'évaluer le score des forêts aléatoires (*RandomForestRegressor* et *RandomForestClassifier*) et des arbres agrégés (*BaggingRegressor* et *BaggingClassifier*) sur les jeux de données *boston*, *diabetes* (régression), *iris* et *digits* (classification). Pour chaque problème et chaque méthode, afficher la moyenne et l'écart type du score évalué par une procédure de validation croisée en 5 étapes (scores = cross_val_score(clf, X, y, cv=5)).

```
****************************************
Boston
****************************************
Score RandomForestRegressor = 0.902423250465
Cross Val : mean = 0.771954476368
Cross Val : std = 0.0985847535624
Score BaggingRegressor = 0.901430879781
Cross Val : mean = 0.752479947922
Cross Val : std = 0.122431066717
****************************************
Diabetes
****************************************
Score RandomForestRegressor = 0.624730890948
Cross Val : mean = 0.422001550322
Cross Val : std = 0.0968974226503
Score BaggingRegressor = 0.622360350136
Cross Val : mean = 0.407256347964
```

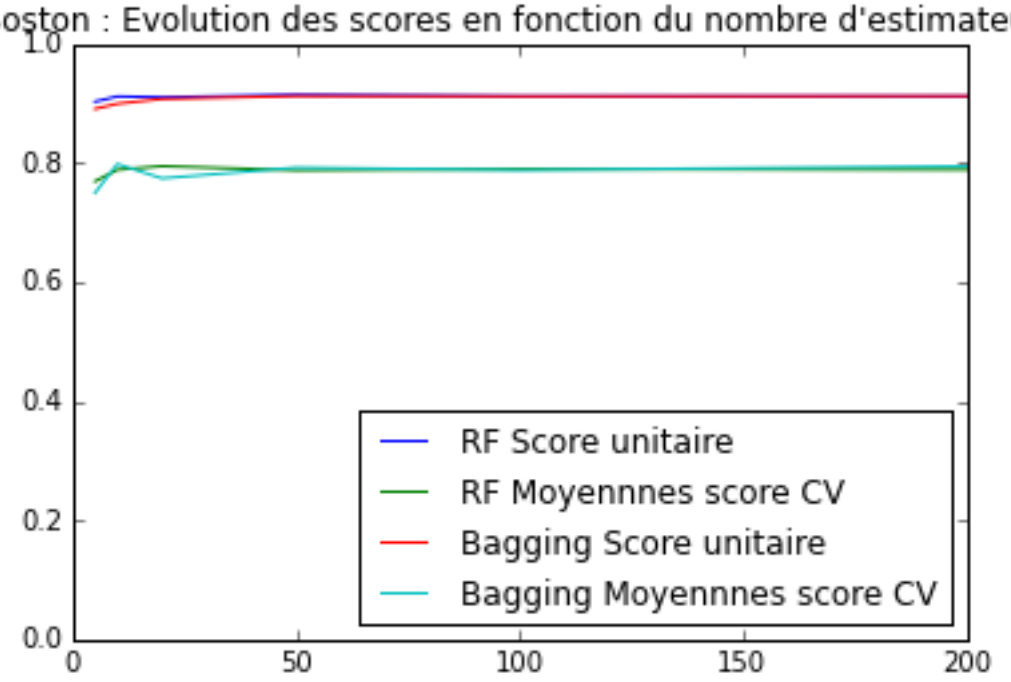
```
Cross Val : std = 0.096666852496
****************************************
Iris
****************************************
Score RandomForestClassifier = 0.98
Corss Val : mean = 0.96
Corss Val : std = 0.0388730126323
Score BaggingClassifier == 0.986666666667
Cross Val : mean = 0.953333333333
Cross Val : std = 0.0452155332208
****************************************
Digits
****************************************
Score RandomForestClassifier = 0.883138564274
Corss Val : mean = 0.851531252646
Corss Val : std = 0.0188555659629
Score BaggingClassifier == 0.804674457429
Cross Val : mean = 0.791155421902
Cross Val : std = 0.0398988680775
```

Rappeler la différence entre les deux approches et en déduire une explication de leurs comportements lorsque le nombre d'estimateurs varie.

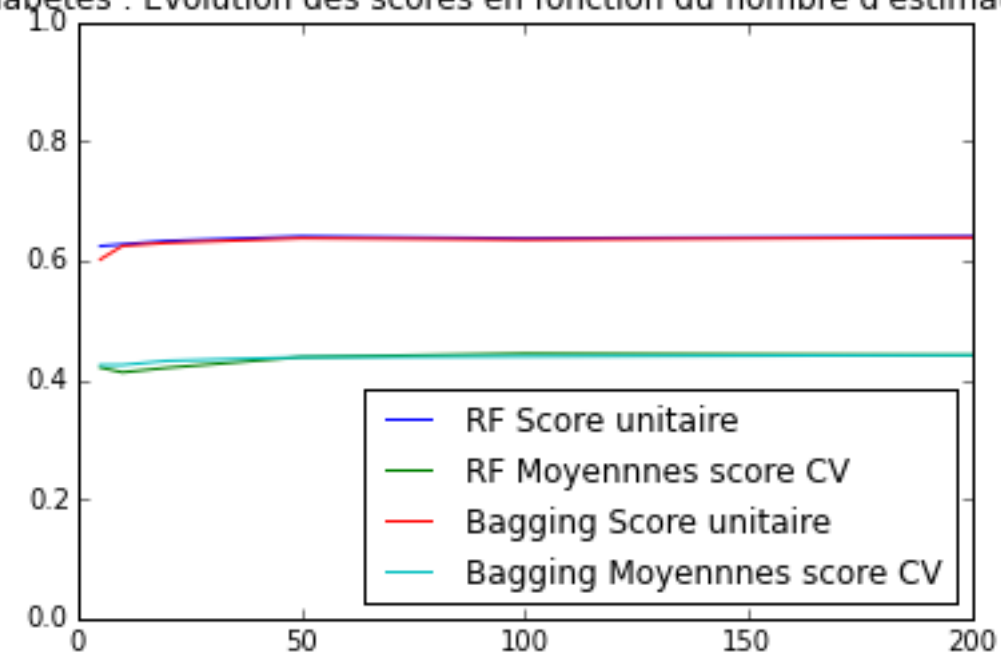
La méthode de forêts aléatoires calcule des estimateurs à partir de "petits" échantillons tirés depuis les données, dans lesquels certaines données sont (aléatoirement) présentes. Tandis que le Bagging part de petites suppressions : il s'agit d'un échantillonnage avec remise dont l'effectif est égal à celui des données, et où quelques données sont (aléatoirement) absentes. Les deux méthodes essaient de minimiser l'influence des données extravagantes (le bruit) et les noyer ainsi dans la masse.

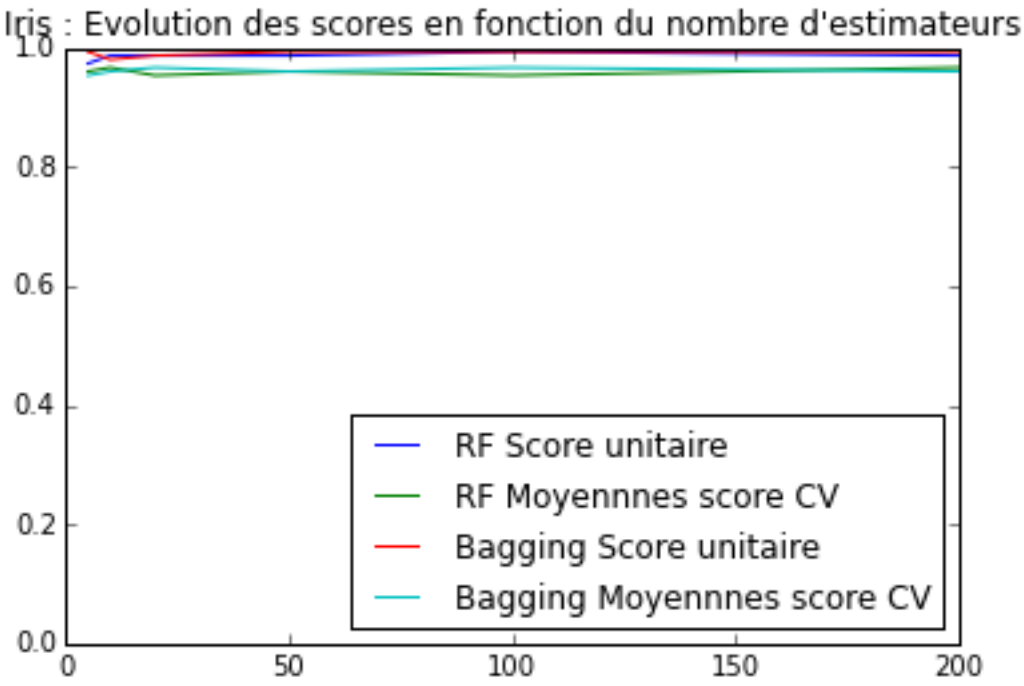
Evolution des scores en fonction du nombre d'estimateurs (voir les courbes suivantes). Je suis assez étonné de voir que l'augmentation du nombre d'estimateurs ait si peu d'effet sur les scores aussi bien les scores individuels que les scores de cross-validation et je n'arrive pas à l'expliquer !

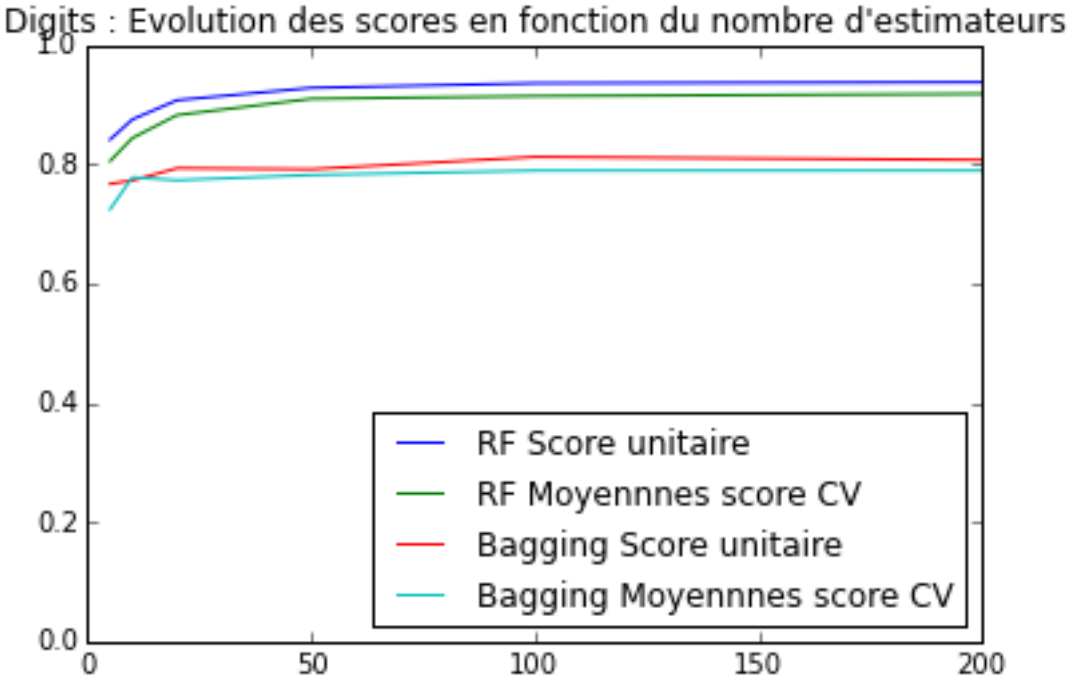
Boston : Evolution des scores en fonction du nombre d'estimateurs



Diabetes : Evolution des scores en fonction du nombre d'estimateurs







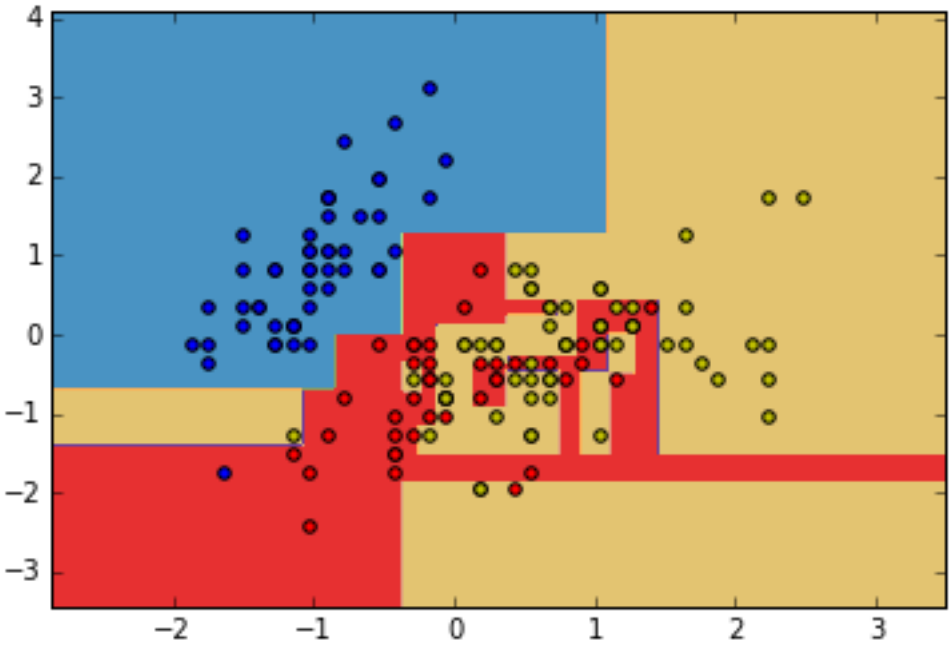
15- En tant que méthode d'agrégation, les forêts aléatoires peuvent retourner des probabilités associées à chaque prédiction. La probabilité d'appartenance d'une donnée à une classe est la proportion d'arbres ayant prédit cette classe. En utilisant le jeu de données iris restreint aux deux premières variables, afficher la probabilité des classes prédites pour chaque observation.

Si j'ai bien compris la question, il s'agit d'afficher, pour chaque observation, 3 probabilités montrant la probabilité d'appartenance de l'observation à chacune des classes 0, 1 ou 2. Voici ce que cela donne pour les quelques premières observations :

```
[[ 0.          1.          0.        ]
 [ 1.          0.          0.        ]
 [ 0.          0.          1.        ]
 [ 0.          1.          0.        ]
 [ 0.          1.          0.        ]
 [ 1.          0.          0.        ]
 [ 0.          1.          0.        ]
 [ 0.          0.4         0.6       ]
 [ 0.          1.          0.        ]
 [ 0.          0.33333333  0.66666667]
 [ 0.          0.4         0.6       ]
 [ 1.          0.          0.        ]
 [ 1.          0.          0.        ]
 [ 1.          0.          0.        ]
 [ 1.          0.          0.        ]
 [ 0.          0.5         0.5       ]
…
```

On est dans le cas d'un seul estimateur.

Mais j'avoue ne pas comprendre l'utilité du graphique (voir le graphique suivant) codé dans le code fourni par rapport à la question posée :

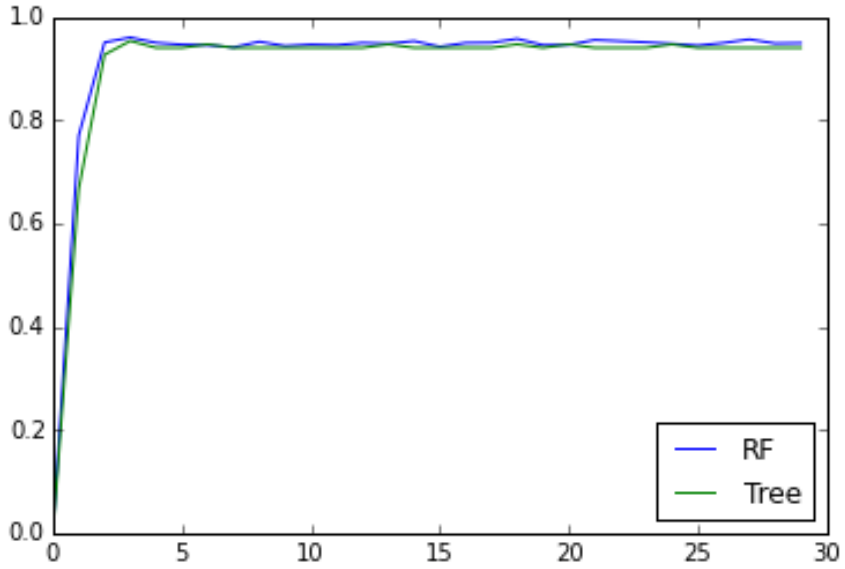


Ce graphique qui est très bien écrit montre les zones de l'espace des observations classées dans l'une des trois catégories. Il est impressionnant qu'on obtienne une si bonne précision avec un seul estimateur !

16- Sur le même jeu de données, comparer les scores des forêts aléatoires et des arbres de décisions seuls (*DecisionTreeClassifier*) en fonction de la profondeur maximale des arbres. Pour cela, vous utilisez une procédure de validation croisée en 5 étapes et ferez varier la profondeur maximale de 1 à 30. Mettre ainsi en évidence la capacité des forêts aléatoires (comparées aux arbres de décision) à réduire le

sur-apprentissage (y compris lorsque l'on utilise des arbres profonds, particulièrement sujets à ce phénomène).

Je n'arrive pas à mettre en évidence une différence significative entre les deux en modifiant la profondeur de 0 à 30 :



J'ai donc scindé les observations en deux groupes (training et validation) pour ne pas utiliser le cross-validation et tester les classifieurs sur des deux groupes (code 16-bis). Cela me donne toujours la même chose et ce n'est pas étonnant car le cross-validation fait la même chose en mieux :