Grenoble INP - Ense3

AI and Autonomous systems

Lab 7 - M2-MARS

Trees and boosting, random forests

Author :
Souhaiel BEN SALEM

17 November 2021

Lab Objectives: The objective of this lab is to introduce Trees, Tree pruning and then boosting by use of random forests. Both classification and estimation problems will be studied.

Introduction:

Decision Trees are considered to be one of the most popular approaches for representing classifiers. A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called "root" that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). Random forests are a scheme proposed by Leo Breiman in the 2000's for building a predictor ensemble with a set of decision trees that grow in randomly selected subspaces of data.

1 NOTEBOOK1: CLASSIFICATION TREE

*

Exercise:

1. **for a predetermined depth(e.g. 3, 4, or 5), analyse and explain the meaning of the information contained in the tree nodes or leaves. What does the node or leave color encode?**
   For a predetermined depth, the sklearn.tree's plot_tree method visualizes the decision tree obtained after training the classifier in the following structure:
   - It displays the root on the top with a white color background.
   - The internal nodes which correspond to the different subsets obtained after splitting the root/other sub-sets based on the attribute value test. These splits are obtained by choosing a test such that at each step we obtain the best splits the set of items.
   - The leaf nodes, also called terminal nodes. They nodes that don't split into more nodes. Leafs are where classes are assigned by majority vote.

   Inside of each node (including the root) we can find the following information:
   - The test on the attribute which is the test used to split the data points contained in the node into two other sub-sets.
   - The impurity function: which measures the extent of purity for the node containing data points from possibly different classes.
   - samples: which is the number of data points contained in the node.
   - value: gives the number of data points in each class according to the split test.

   Inside of every Leaf we foind the same information except when the leaf is completely pure, in that case we don't require a split test and the gini impurity index is 0.

Each color denotes a class, and the intensity of the color represents how pure the node is (i.e the ratio of the majority with respect to all the data points).

2. **starting at a leaf and going up to the root node, is the evolution of the impurity index monotonic?**
   No, the impurity index does not evolve monotonically. In fact, if we focus on a specific leaf and examine the path from the root to that leaf, we can see that the value of the gini index is not changing in a monotonic way (sometimes we get a negative index and afterwards we get a positive index for instance).In fact, it is not important if Gini is mootonic or not, what's important is the Gini gain.

3. **What is the impurity gain between the root node and its two children nodes?**
   The impurity gain between the root and the two children nodes is given by the formula:

$$\Delta GI(S) = GI(S) - P(S_l)GI(S_l) - (1 - P(S_l))GI(S_r)$$

   where $P(S_l) = \frac{N_l}{N}$ and $N_l + N_r = N$
   In our case, we get

$$\Delta GI(S) = 0.5\frac{1530}{3000} \times 0.308(1\frac{1530}{3000} \times 0.296 = 0.19788$$

4. **Would the choice of a larger depth lead to a better classification tree? in which sense?**
   A larger depth would lead to a better classification as the gini will keep increasing with the depth. However, if we build a very deep tree with no constraint, it will eventually overfit the training data.

Exercise 2:

1. **Which of these two matrices is the more relevant for characterizing the classifier?**
   The **test confusion matrix** is the most relevant when it comes to characterizing the performance because it gives us an idea on our classifier's accuracy on unseen data, which is the whole purpose of the classifier (to be used on new, unseen data).

2. **What does the accuracy represent?**
   accuracy is a metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy is:
   Accuracy = Number of correct predictions Total number of predictions.

3. **Is such an accuracy estimate on a single experiment reliable?**
   A single experiment is not enough to assess the performance/accuracy of a model/algorithm. In practice, cross-validation is used to test the generalizability of the model.
   As we train any model on the training set, it tends to overfit most of the time, and in order to avoid this situation, we use regularization techniques. Cross-validation provides a check on how it is performing on unseen data.

Exercice:

1. **From the set of experiments and results above, set the optimal value nbdepth for noise=.4. What do you observe?**
   If we increase the noise, we notice that that points overlap more and the homogeneity of our data decreases which makes it harder to split the data at each iteration.

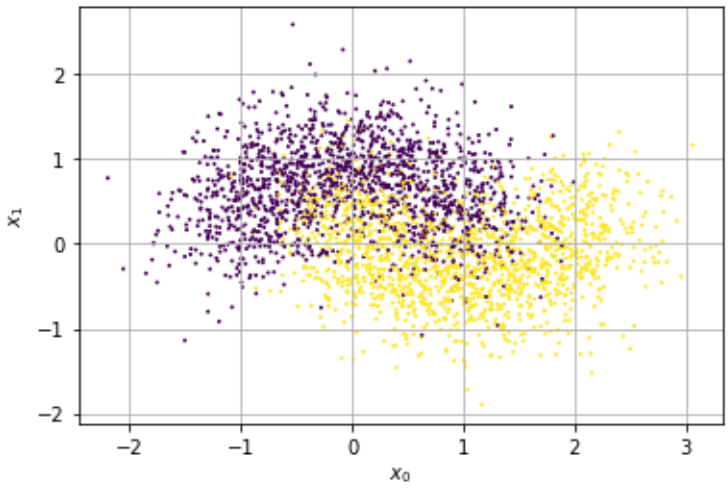

Figure 1: The dataset for ndepth=0.4

This explains why the accuracy of the decision tree dropped from 0.91 to 0.85 and the fact that the maximal optimal depth dropped from 5 to 4.

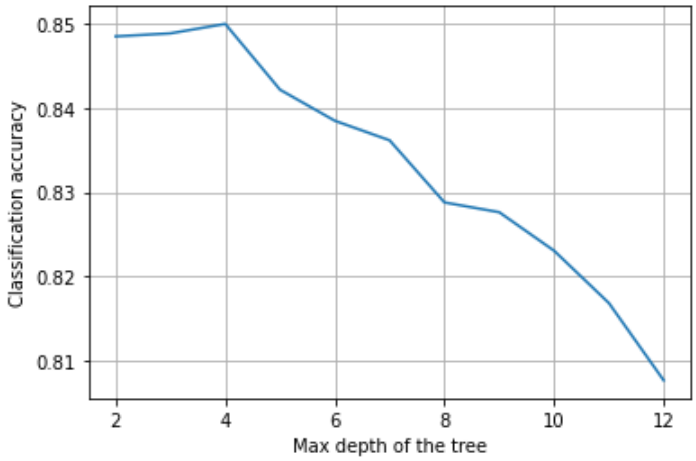

Figure 2: The accuracy corresponding to different tree depths

2. **Run the code for a high value of the maximal depth parameter. What do you observe? Explain**
   If we set the tree depth too high (10 for example) or set it to be the maximum depth (i.e nbdepth = None), we notice that our decision tree overfits the training data which explains the drop in accuracy and the complicated classification boundaries.

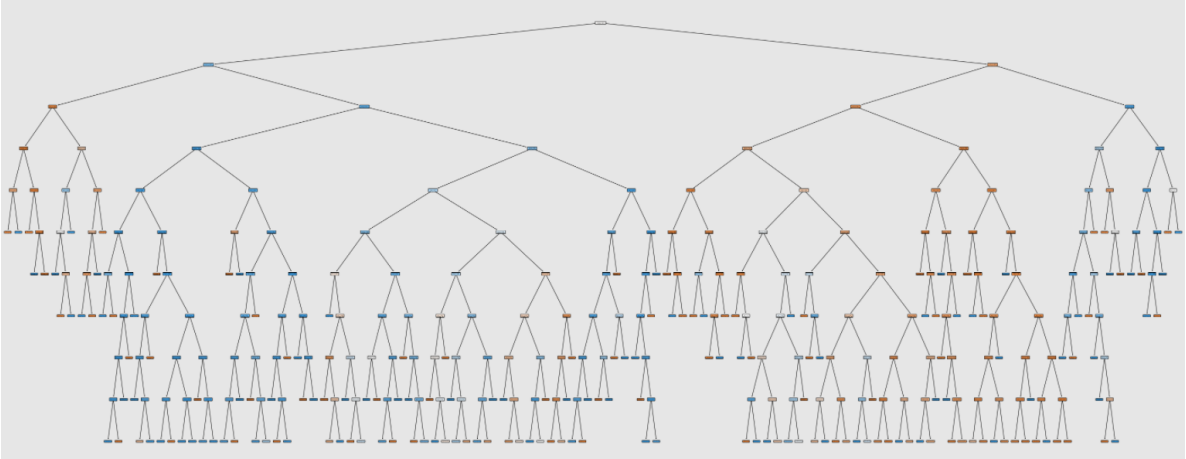

Figure 3: Decision Tree with a depth of 10

The decision boundaries we get with this tree demonstrate the overfitting of our algorithm to the training data:

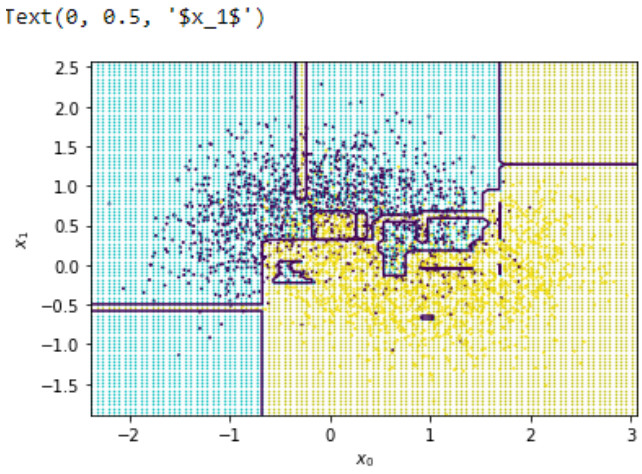

Figure 4: Decision Tree with a depth of 10

We also notice that the best accuracy was obtained for a low depth (k=2) because the data we used is noisy and our tree was not able to generalize by going deeper.

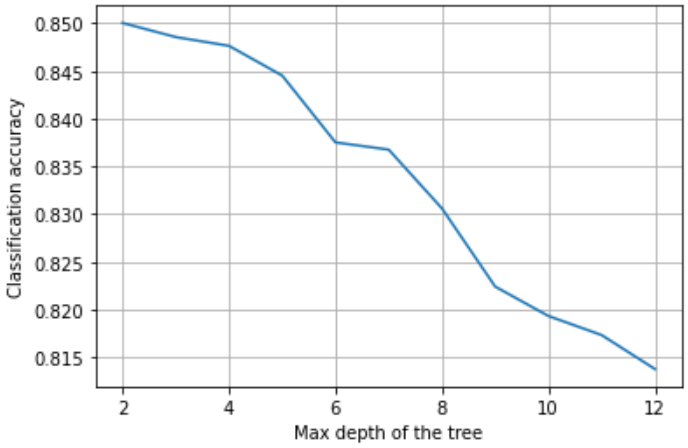

Figure 5: The accuracy plot for our decision Tree with a depth of 10

3. **If the noise variance is set to zero (noise=0.) and nbdepth to a high value, explain why the tree growing is stopped before the max depth is reached.**
   Setting the noise to value of 0, means that our data becomes completely separable. Thus, our decision tree would be able to split the data points with high accuracy and reach the best, pure subsets especially our classification problem is binary (which our case here).
   As result our decision tree would reach final classification labels i.e leafs without the need to go deep. In fact, If our tree reaches the nodes where all the data points are a part of the same target value, splitting no longer makes sense as it does not add value to the prediction.

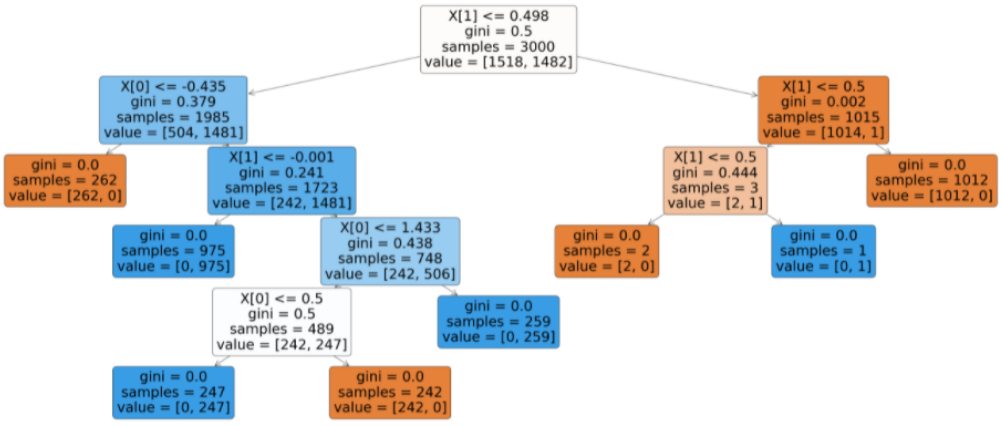

Figure 6: Our tree stopped at a depth of 5 despite setting ndepth=10

The plot of the accuracy with respect to the maximum depth confirms our assumptions and we can see that with a depth=5, our decision tree was able to attend 100% accuracy and thus, it wouldn't make sense if we get deeper because our external nodes are totally pure.

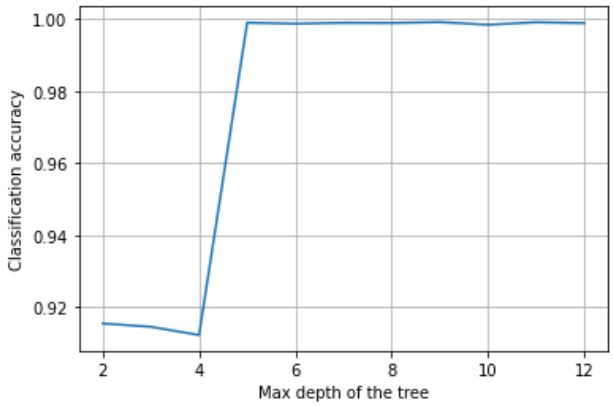

Figure 7: The evolution of accuracy with respect to the maximum depth of the decision tree

4. **Change the noise variance to a higher value. how does the optimal value of the depth evolve? Comment.**
Noise creates trouble for any machine learning algorithm because if not trained properly, algorithms can think of noise to be a pattern and can start generalizing from it, which of course is undesirable. We ideally want the algorithm to make sense of the data and generalize the underlying properties of the data. This is prevented by noise which tends to "fool" the algorithm to making incorrect generalizations.
When we set the noise variance to high value (0.85) for example, the two classes of the data become practically inseparable.

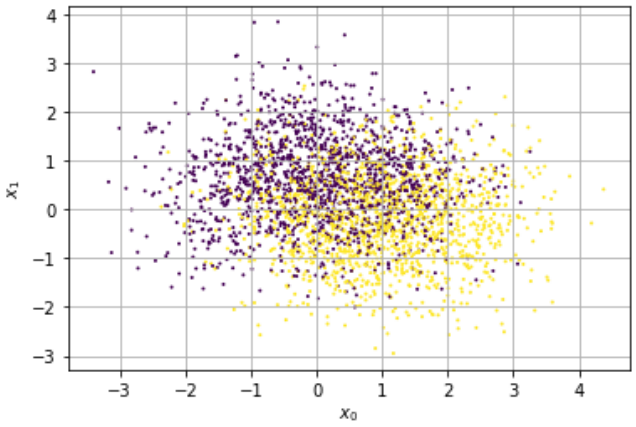

Figure 8: highly overlapping, uninterpretable data

The poor generalization of our decision tree in this case can be seen when looking at the accuracy plot with respect to the maximum depth. The maximum accuracy is around 72% and it keeps decreasing the deeper our tree gets because more depth means more overfitting the noise.

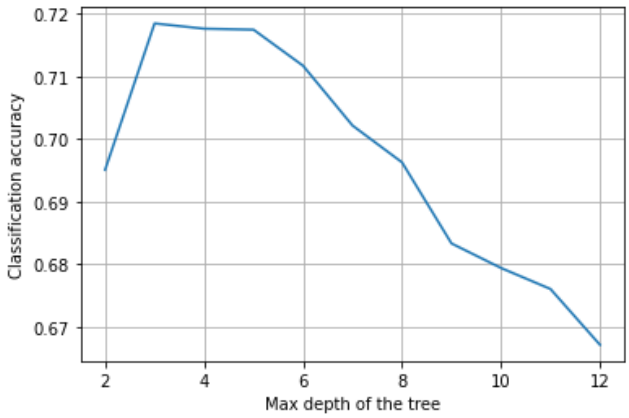

Figure 9: The evolution of accuracy wit respect to the depth of our decision tree

2 NOTEBOOK2: REGRESSION TREE

Exercise4:

1. **How many different prediction values are defined by a tree of depth d?**
   Let N = number of training examples, k = number of features, and d = depth of the decision tree.

   A regression tree would calculate a quality function based on each split of the data, and it does this for each feature in every node that is not a leaf node. This happens as long as there are levels (depth) to continue to.
   If we have N samples then there $(N-1)$ splits. More generally, for a set S with N points in k dimensions, the decision tree needs $k \times (N-1)$ tests.

2. **What is the average number of samples from the training set in each leave? (let N be the size of the training set)**
   The theoretical maximum depth of a balanced regression tree is equal to the number of samples N. This tree would have 2^N leaf nodes, meaning that the average number of sample in each leaf is equal to $\frac{N}{2^N}$.

Exercise5:

1. **Determine the "optimal" depth to use to perform the best (MSE sense) tree based prediction**
   According to the MSE plot with respect to the maximum depth of the regression tree, we can see that the best performance (i.e least MSE) is obtained for **depth=5** .

2. **Change the noise power (set e.g. noise_std to take different values in the range [.01;1] and study (plot) the obtained cross-validated "optimal depth" as a function of the noise power.**
to make the noise [0.01,1], we use the **np.arange()** function:

```
%matplotlib inline
import random
from random import randrange,uniform
import matplotlib.pyplot as plt
import numpy as np

# Make a noisy sine shape function
# np.random.seed(0)
#noise_std =np.random.rand(0.01,1)
X = np.arange(0, 2 * np.pi, 0.01)[:, np.newaxis]
noise_std=np.arange(0.01,1,0.00157393)[:, np.newaxis]
nx=np.random.randn(X.shape[0], 1) * np.random.rand(noise_std.shape[0],1)
y=np.sin(X) + np.random.randn(X.shape[0], 1) * np.random.rand(noise_std.shape[0],1)
print("The number of point in the set is {}".format(len(X)))

# changing y to observe the behaviour on linear regression
# y= .5*X + np.random.randn(X.shape[0],1)*noise_std


plt.figure()
plt.scatter(X, y, s=1)
plt.xlabel("X")
plt.ylabel("y")
noise_std.shape
```

```
The number of point in the set is 629

(629, 1)
```

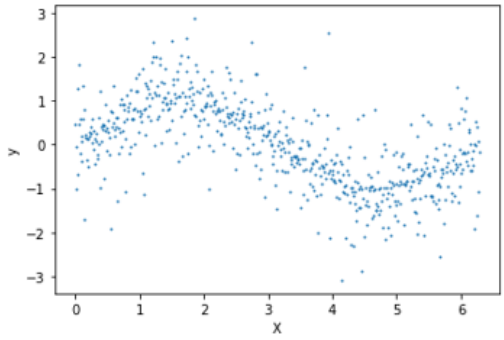

Figure 10: making noise vary in the range of [0.01,1]

To get the obtimal depth obatined by cross validation for value of noise, we can use the two lines of code:

- **i=min(range(len(reg_MSE)), key=reg_MSE.__getitem__)**: to get the index where MSE is minimum
- **optimal_depth=depth[i]**: to get the corresponding depth of the minimum MSE

We loop over the values of the noise (in the range [0.01,1]) and determine the optimal depth value

for each noise power value and finally plot the optimal depth as function of noise power.

3 NOTEBOOK3: COST COMPLEXITY PRUNING OF A REGRESSION TREE

Exercise6:

- **do you think that such a regressor will have good performances for other noise realizations? Explain.**
  The regressor we obtained by not setting up contraints on the maximum depth is clearly overfitting the training data as we kept splitting until we got completely pure subsets (125 leaves). We can visual the predictions of our regression tree agains the original data to emphasis the overfitting problem:

```
: # Define a new set of inputs with the same range as X
X_test = np.arange(0.0, X.max(), 0.01)[
    :, np.newaxis
]  # alternate method, to get a 2D array for 2D array from a scalar time series
# use the computed trees to obtain prediction values

clf = clf.predict(X_test)

plt.scatter(X, y, s=1)
plt.plot(X_test, clf, color="red", label="N depths", linewidth=2)
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

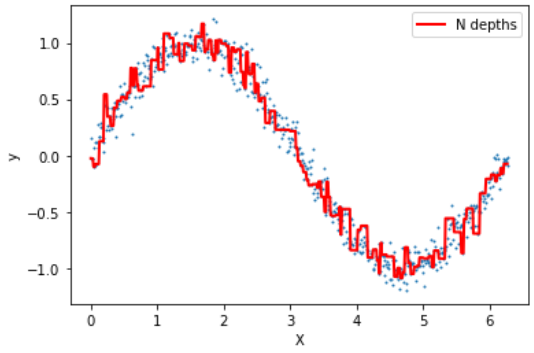

Figure 11: the regression tree's predictions againt the original data

Exercise6:

1. **what is the regression tree that one obtains for $\alpha = 0$?**
   $\alpha = 0$ means no penality on the number of the leafs. so for $\alpha = 0$ we have

   $$C_\alpha(T_t) = C(T_t)$$

   which means that the regression tree in this case is the original, full sized tree trained on the whole dataset.

2. **what is the regression tree that one obtains for very large values of α**
   For a very large α, we are penalizing the number of leaves heavily and will end with a uni-node tree which contains only the root node.

Exercise8:

- **How could the score estimates be improved?**
  The score estimate could be improved using a k-fold cross-validation procedure: we start by building a full regression tree using all of the data (fit to the totality of the data) and then we iterate over different values of α and prune our regression tree until we are left with only the root node. Now we get a sequence of decreasing trees and increasing values for α. Afterwards, we go back to the full dataset and divide it into training and we choose the value α to minimize the cross-validated sum of squares on the testing sets.
  More precisely:
  - Using the whole training set, we get a sequence of trees, $T0 > T1 > ... > troot$, where T_k is the best tree for $\alpha_k \alpha < \alpha_{k+1}$.
  - For k = 0, 1, 2,.. we cross-validate and determine the sum of squares.
  - Select the tree T_k corresponding to the minimum cross-validated error.

Exercise10:

- Pruning is a tradeoff between overfitting and underfitting because it allows the regression tree to grow deeper and be more complex (draw complex decision boundaries) in some regions if necessary and be simple in some other regions with low impurity. This makes the regression tree more flexible and guarantees high accuracy predictions compared to the prescribed maximal depth approach.

4 NOTEBOOK4: RANDOM FORESTS REGRESSORS:

Exercise11:

1. **Change the parameter**
   max$_d$*epthinthecodeaboveandComparethebehavioroftherandomforestregressorwiththebehaviouroftheregressorof*
   The max_depth parameter is the maximum depth that a tree in the forest can grouw into. After setting this parameter to 10, we got an accurate model with low $MSE = 0.0031$ which means that our model is overfitting:

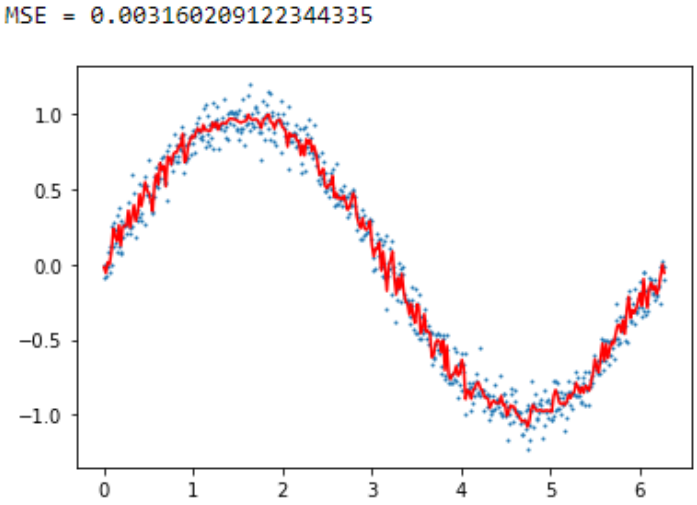

Figure 12: result obtained after setting the maximum depth of the random forest regressor to 10

   On the other hand, if we increase the maximum depth of the regression tree studied in Notebook2_a, the model becomes too complicated and overfits the training data which results in poor performance (we get a minimum MSE of 5.57).

2. **Explain your findings**.
   Random forests deals with the problem of overfitting by creating multiple trees, with each tree trained slightly differently so it overfits differently. Random forests is a classifier that combines a large number of decision trees. The decisions of each tree are then combined to make the final classification. This “team of specialists” approach(random forests) often outperforms the “single generalist” approach of decision trees. The random forest regressor takes an average of the predictions of its trees, and thus, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees and introducing some bias (with the variance reduction being more significant than the bias introduction) which allows them to avoid overfitting and better generalize to unseen data.

Exercise12:

1. **Study the Extremely Randomized Regressor behavior for $\mathbf{max}_{d}epthparametervalues(changeitinthecodeabove)rangingfrom1to6$**
   When we change the maximum depth parameter for for the Extremely randomized regressor from 1 to 6, we notice that the model's accuracy increases (the MSE decreases with depth). This is because more depth means better prediction for individual trees and thus better overall performance for the random forest.

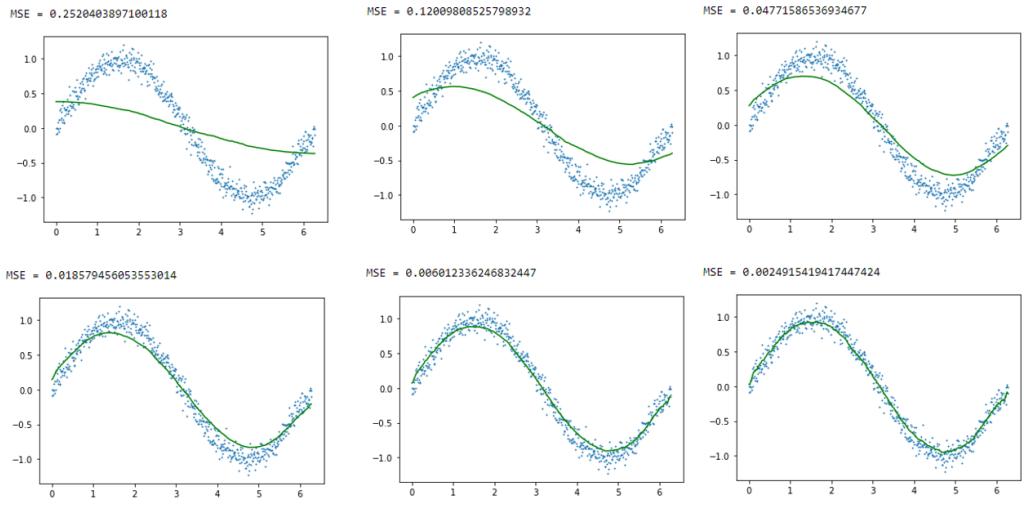

Figure 13: Different results corresponding to different maximum depth values

2. **Propose a method for setting the optimal value of $\mathbf{max}_{d}epthparameter.Implementit$**
   The method to be used for maximum depth estimation is cross-validation.
   The implementation is as follows:

```python
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=10, train_size=0.5, test_size=0.5, random_state=None)
depth = np.arange(2, 20)
reg_MSE = []
for N in depth:
    clf2 = ExtraTreesRegressor(
    n_estimators=1000, criterion="mse", max_depth=N, random_state=None
    )
    mserr = []
    for train_index, test_index in cv.split(X):
        clf2 = clf2.fit(X[train_index], y[train_index].ravel())
        y_pred = clf2.predict(X[test_index])
        y_t = y[test_index].ravel()  # to force same dimensions as those of y_pred
        mserr.append(np.square(y_t - y_pred).sum())
    reg_MSE.append(np.asarray(mserr).mean())


plt.plot(depth, reg_MSE)
plt.xlabel("max Depth of the reg_tree")
plt.ylabel("MSE")
plt.grid()
```

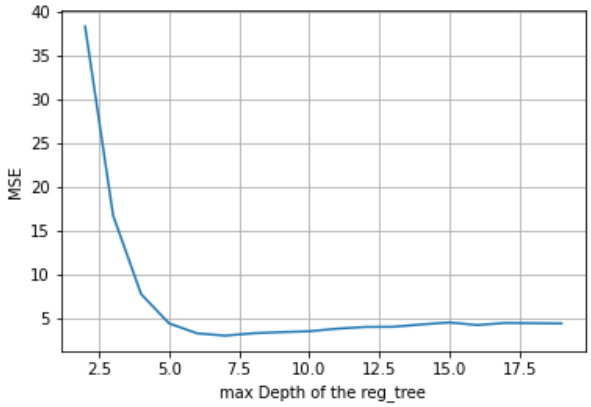

Figure 14: estimation of the optimal maximum depth for the extremely randomized random forest

We notice that starting from a certain maximum depth (**optimal depth**), the MSE holds a somewhat constant value unlike the normal decision/regression trees that overfit and have an explosive MSE when the maximum depth increases above the optimal value. This result, confirms yet again one of the main advantages of random forests which is avoiding overfitting even for large numbers of trees(here 1000) and maximum depth values.

3. **Why does the Extremely Randomized Regressor exhibit good performances, although splits are chosen at random?**
   Extremely Randomized do not resample observations when building a tree. (They do not perform bagging) and they do not use the “best split”.Instead of the “best split” for the predictors, this

algorithm makes a small number of randomly chosen splits-points for each of the selected predictors.The resulting forest contains trees that are more variable, but less correlated than the trees in a Random Forest. This assumption results in even lower variance than that obtained using random forests and yields more often better results than those obtained by random forests.

5 NOTEBOOK4: RANDOM FORESTS REGRESSORS:

Exercise13:

- **As IRIS data file contains only 150 4-dimensional samples, assuming that we impose that no less than 2 samples are contained in a leaf, and that the training test is chosen to contain 100 samples, what is the possible maximal depth?**
  The theoretical maximum depth a decision tree can achieve is one less than the number of training samples which is 99 in this case (but no algorithm will let you reach this point for obvious reasons, one big reason being overfitting).
  If we want to impose a minimum of 2 samples per leaf, the the maximum depth would be $(100/2) - 1 = 49$.

Exercise14:

1. **Compute the confusion matrix associated to this classifier. (Hint : see N1_Classif_tree.ipynb)**
   The confusion matrices (training + testing matrices) of our claasifiers are computed as follows:

```
from sklearn import metrics

y_est = clf.predict(X_train)
print("Confusion matrix from training set :")
C_train = metrics.confusion_matrix(y_est, y_train)
print("C_train =\n {}".format(C_train))
print("accuracy= {}".format(metrics.accuracy_score(y_est, y_train)))

y_est = clf.predict(X_test)
print("Confusion matrix from test set :")
C_test = metrics.confusion_matrix(y_est, y_test)
print("C_test =\n {}".format(C_test))
print("accuracy= {}".format(metrics.accuracy_score(y_est, y_test)))
```

```
Confusion matrix from training set :
C_train =
 [[35  0  0]
 [ 0 34  1]
 [ 0  0 30]]
accuracy= 0.99
Confusion matrix from test set :
C_test =
 [[15  0  0]
 [ 0 16  3]
 [ 0  0 16]]
accuracy= 0.94
```

Figure 15: The two confusion matrices evaluated on both the training and testing sets

2. **Compute the mean accuracy of this tree classifier. (Hint : see N1$_C lassif_tree.ipynb$)**
   The mean accuracy of our decision tree is computed as follows:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=None)
scores = cross_val_score(clf, iris.data, iris.target, cv=cv)
print(
    "Mean Accuracy and 95 percent confidence interval : %0.2f (+/- %0.2f)"
    % (scores.mean(), scores.std() * 2)
)
```

```
Mean Accuracy and 95 percent confidence interval : 0.94 (+/- 0.08)
```

Figure 16: The mean accuracy of our decision tree

3. **Change the value of parameter**
   max$_d epth(ranging from 1 to 5) and record the obtained accuracy. Explain your findings.
   We notice that starting from mdepth=2, the random forest yields a constant accuracy of 95%. This
   Knowing that our random forest is immune to overfitting, we can assume that the subsets achieved by using a depth of 2 are no longer being splitted because they are pure, which means that the optimal depth value is possibly 2.

```
from sklearn.ensemble import RandomForestClassifier

mdepth_array = np.arange(1, 5)
for mdepth in mdepth_array:
    print("mdepth={}".format(mdepth))
    clf = RandomForestClassifier(
        n_estimators=40,
        max_depth=mdepth,
        random_state=None,
        min_samples_split=2,
        criterion="gini",
    )
    scores = cross_val_score(clf, iris.data, iris.target, cv=10)
    print(
        "Mean Accuracy and 95 percent confidence interval: %0.2f (+/- %0.2f)"
        % (scores.mean(), scores.std() * 2)
    )
```

```
mdepth=1
Mean Accuracy and 95 percent confidence interval: 0.92 (+/- 0.21)
mdepth=2
Mean Accuracy and 95 percent confidence interval: 0.97 (+/- 0.07)
mdepth=3
Mean Accuracy and 95 percent confidence interval: 0.97 (+/- 0.07)
mdepth=4
Mean Accuracy and 95 percent confidence interval: 0.97 (+/- 0.07)
```

Figure 17: Accuracy values for the depths going from 1 to 5

4. **Propose a method for setting the 'best' value of parameter n_estimator.**
   To determine the best number of trees in our random, we can proceed by a cross-validation where evaluate the model for different numbers of estimators and keep track of the MSE at each iteration. The best value of n_estimators would yield a minimum MSE.

   Exercise16:

5. **Evaluate the feature importance in the IRIS Data Set , using ExtraTreesClassifier.Compare with the results above.**
   The first step is to build the ETC which is done as follows:

```
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.model_selection import ShuffleSplit
mdepth_array = np.arange(1, 5)
for mdepth in mdepth_array:
    print("mdepth={}".format(mdepth))
    clf2 = ExtraTreesRegressor(
    n_estimators=40, criterion="mse", max_depth=mdepth, random_state=None)
    scores = cross_val_score(clf2, iris.data, iris.target, cv=10)
    print(
        "Mean Accuracy and 95 percent confidence interval: %0.2f (+/- %0.2f)"
        % (scores.mean(), scores.std() * 2)
    )
```

```
mdepth=1
Mean Accuracy and 95 percent confidence interval: 0.08 (+/- 0.34)
mdepth=2
Mean Accuracy and 95 percent confidence interval: 0.14 (+/- 0.56)
mdepth=3
Mean Accuracy and 95 percent confidence interval: 0.18 (+/- 0.71)
mdepth=4
Mean Accuracy and 95 percent confidence interval: 0.29 (+/- 0.88)
```

Figure 18: Building the ETC

Afterwards, we can compute the feature importance:

```
clf2.fit(iris.data, iris.target)
importances = clf2.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf2.estimators_], axis=0)
print(std)
indices = np.argsort(importances)[::-1]
indices
```

```
[0.17667481 0.02985286 0.37912777 0.38343838]

array([2, 3, 0, 1], dtype=int64)
```

Figure 19: computing the features' importance

And finally we can plot the feature importance with respect to their weights:

```
# Print the feature ranking
print("Feature ranking:")

for f in range(4):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
    # Plot the feature importances of the forest
plt.figure(figsize=(5, 6))
plt.title("Feature importances")
plt.bar(
    range(np.asarray(iris.data).shape[1]),
    importances[indices],
    color="r",
    yerr=std[indices],
    align="center",
)
plt.xticks(range(np.asarray(iris.data).shape[1]), indices)
plt.xlim([-1, np.asarray(iris.data).shape[1]])

plt.show()
```

```
Feature ranking:
1. feature 3 (0.501053)
2. feature 2 (0.444694)
3. feature 0 (0.052475)
4. feature 1 (0.001778)
```

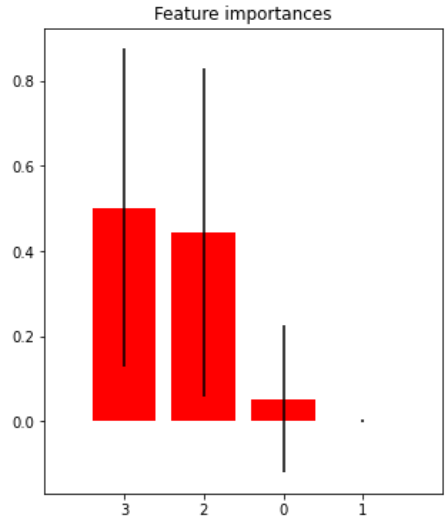

Figure 20: plotting the magnitude of the features importance

Here, we notice that both the random forest and the ERT ”decide” the features importance hierarchy similarly but assign different weights to each one with ERC assigning the lower weights (which explains why it has lower variance).

6. **What can be concluded about the features importance?**
   From this experiment, we can conclude that the feature importance is independent of the type of claasifier.

Conclusion:

Decision trees and random forests are one of the most important and powerful tool of machine learning due to their unique properties and interpretability. To this day, they are considered as the state of the art tool for many application such as hyper-spectral imaging analysis. During this lab, we experimented with these algorithms through examples covering both classification and regression and we have also explored some extensions of the these methods and their statistical properties.