Data Analysis and Machine-Learning

Chapter 8:

Decision Tree Model



*I never was, am always to be,*

*No one ever saw me, nor ever will,*

*And yet I am the confidence of all*

*To live and breathe on this terrestrial ball.*

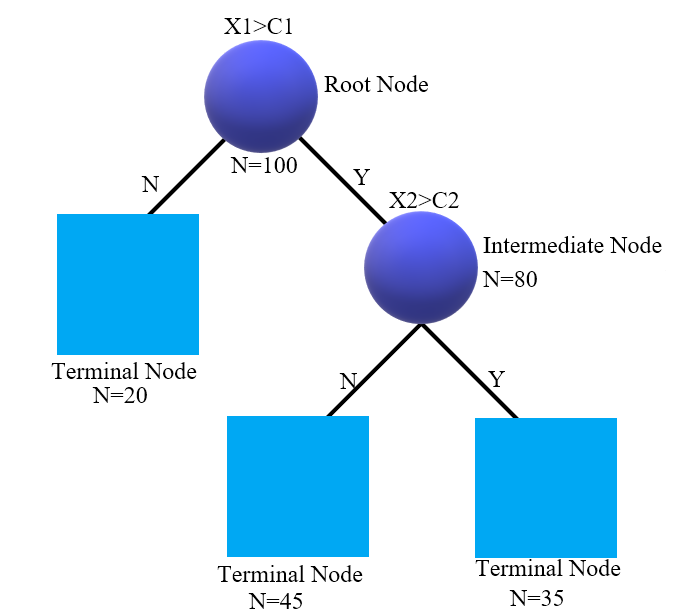
*What am I?*

1. Introduction

As it can be inferred from the very name of the model, decision tree generates a tree-shaped classification/prediction model based on the given data’s *inherent patterns*. Simply put, the mechanism of this algorithm can be understood as the process of throwing multiple **questions** and narrowing down to an appropriate answer accordingly to the answers to those questions.

More specifically, decision tree algorithm groups data in terms of uniformity, which is done by *partitioning* the data into two or more numbers of subsets. For classifications, observation values with similar *categories or classes* are grouped together as subsets. And for predictions, observations values with similar *numerical values* are grouped together into subsets.

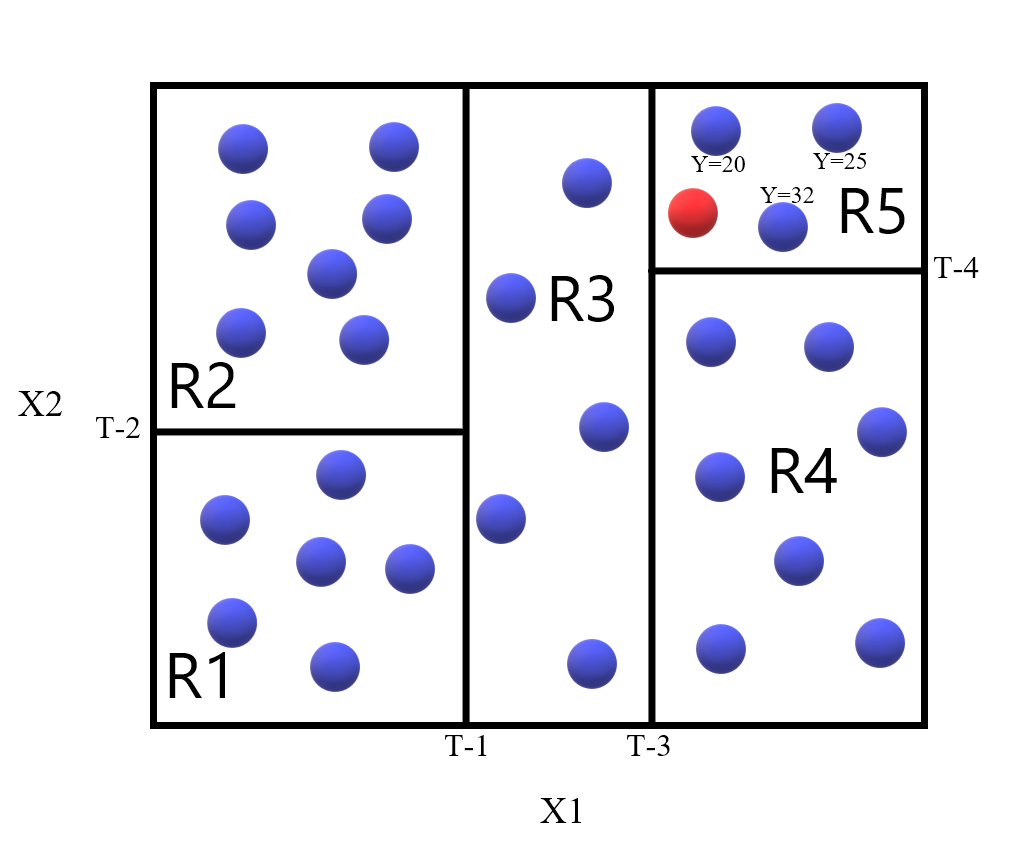
The resulting output after the whole process appears in a shape of an upturned tree, with root node located on top, intermediate nodes in the middle, and terminal nodes at the end of the model output:



Notice that the total number of observations equals to the sum of the total number of observations in each terminal node.

2. Decision Tree Diagram

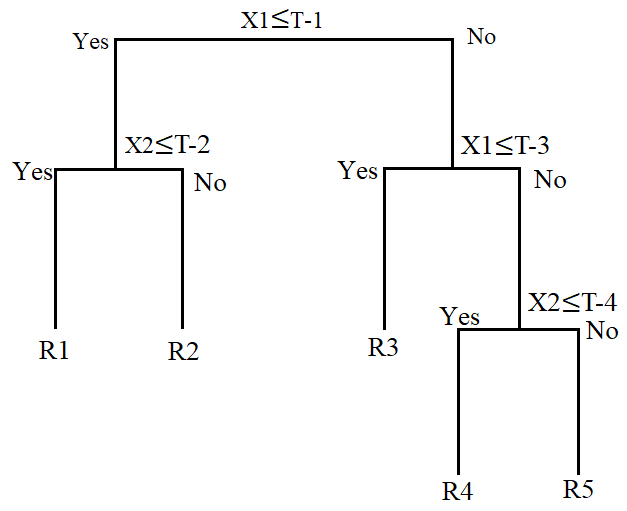
As introduced earlier, observations values with similar *numerical values* are grouped together into subsets for prediction tree models. Consider following figure for understanding.



With X1 on the x-axis and X2 on the y-axis, Ys are represented as blue dots, each of which indicates numerical values (recall that as for the prediction model, Ys are expressed numerically). Now, data are partitioned into subsets throughout the phases from T-1 to T-4, generating areas R1~R5. Suppose a new observation Y is presented (red dot). How should we predict the value of this observation? A standard way would be to calculate the arithmetic mean of the values in area R5, since the new observation Y is located in the corresponding area. Thus, the prediction based on average calculation would be: (20+25+32)/3 = 25.67.

3. Tree Expression

The very same process can be drawn in a tree-shape, as follows.



Notice that this tree model is fundamentally expressing the same thing with the diagram illustrated earlier. For starters, the tree model may be rather less intuitive compared to the diagram expression, so why rather use the tree expression? The reason is simple. Diagram visualization can only convey at most 3-dimensions, and higher dimensions are impossible to be drawn. As it can be inferred from the vertical structure of the tree expression, the main advantage of using the tree model expression is that high-dimensional model can be illustrated simply by adding more nodes.

4. General Equation of the Decision Tree Model

Now, let us consider the equation for this regression tree model, which is not at all different from what I have illustrated earlier. The general equation would look like:

For better understanding, consider the following equation which expresses the specific tree model made earlier:

The condition within the function indicates whether x1 and x2 are within the area Rm (R1~R5). Thus if x1 and x2 are not in the area, the resulting value would be 0. Since the presented observation is within area R5, we get a constant value C5 while all other constants simply turn to 0.

The equation expression for classifications can be written in a similar manner, when K for classification model is defined as:

Where max of K(m) refers to the category with largest probability among the ratios of observations belonging to the k-class at terminal node m (i.e., Pmk).

Thus supposing a new data observation at R5 for classifications,

Similarly to the prediction model, the generalized equation in this case would be:

5. Optimization

5.1. Cost (Error) Function

Now that we understand the diagram, tree expression, and the general equation of the decision tree model, we need to know the cost function for optimization of the model. The best partition is achieved when the following cost (error) function is minimized:

When

Replacing f(x) with the general equation obtained earlier:

Solving this equation, we get:

In other words, errors are minimized when predicted with the *average* of Ys within each partition.

For classification models, three types of cost functions (***Impurity Measures***) can be used accordingly to the model design, where (1) Misclassification Rate measures the match rates between y and k(m), (2) Gini Index and (3) Cross-entropy calculate based on the concept of probability.

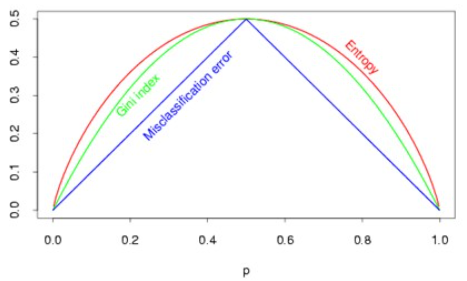
In the previous chapters on logistic regression (generalized linear model), I have illustrated the concept and equation for cross-entropy using *log*, which in this case can be written as:

As a result, cross-entropy provides minimum node impurity of 0 to maximum of 1 (*unless scaled to 0.5*), with 0 referring to the lowest degree of impurity and 1 referring to the highest degree of impurity.

For Gini Index, which also incorporates the concept of probability in terms of impurity calculation, can be written as:

As a result, Gini Index provides minimum node impurity of 0 to maximum of 0.5, with 0 referring to lowest degree of impurity and 0.5 referring to the highest degree of impurity.

For misclassification errors, the equation incorporates the concept of ‘matching’ between y and k(m), as follows:



5.2. Optimal Division Variable and Point

Now that we know that the minimization point can be found at the average, it is possible to determine the division variable ‘j’ and division point ‘s’.

For

Knowing that:

Best Division variable and point (j and s, respectively) can be defined as:

Based on the obtained determining equation, the specific process of finding the best division variable (among X1~Xn) and point resembles the process of gridsearch, as the minimization point is specified via considering all number of cases (e.g., X1>=10, X2>=20, X3>=30,...) until the best parameter is located.

In line with the prediction model using the cost function to calculate optimal parameters, impurity measures are used to set optimal division variable and point in classification models. For example, if misclassification rate is used as the cost function:

For

When

Using Misclassification Rate,

For example, impurity rates would be high for cases where the ratio between class 0 and class 1 is 50:50, compared to the case where the ratio is 90:10 (low impurity rate). Best division variable and point can be located where the decrease of impurity is maximized. When deciding if the impurity rate is high or low, one significant point to remember is the maximum rate for each impurity measures. For example, the maximum P for Gini Index is 0.5 (with lowest impurity 0 to highest impurity at 0.5), while the maximum P for cross-entropy (**unless scaled to 0.5**) is normally **1** (with lowest impurity 0 to highest impurity at 1).

6. Decision Tree Algorithm – Interpretational Understanding

Fully comprehending the concepts of decision tree through scatter diagrams, tree expressions with nodes, regular expression equations, and calculation method for cost functions and optimization, looking at how the algorithm actually works on real data would help understanding the applicative features of decision trees. Here, I will demonstrate the step-by-step procedures of decision tree algorithms using various sample data.

#Import Essential Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

#Generate Sample

x, y = datasets.make\_classification(

    n\_samples=50, n\_features=2, n\_informative=2, n\_classes=2, n\_redundant=0,

    n\_clusters\_per\_class=2, random\_state=123

)

np.c\_[y, x]

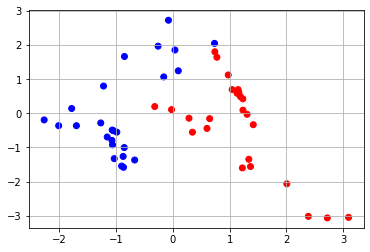
x1, x2 = x[:,0], x[:,1]

#Sample scatter plot

plt.scatter(x1, x2, c=y, cmap='bwr')

plt.grid()

plt.show()



#Decision Tree Algorithms

import matplotlib.image as img

plt.figure(figsize=(20,30))

plt.subplot(4,2,1)

model1 = DecisionTreeClassifier(max\_depth=1).fit(x, y)

dt.dimensionchange(model1, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x1,x2,c=y, cmap='RdYlGn')

plt.title('max depth: 1', fontsize=20)

plt.grid()

from sklearn import tree

import graphviz

data = tree.export\_graphviz(

    model1, out\_file='1.dot',

    filled=True, rounded=True,

    special\_characters=True

)

graph = graphviz.Source(data)

plt.subplot(4,2,2)

image = img.imread('1.png')

plt.title('Tree Expression with Depth=1', fontsize=20)

plt.imshow(image)

plt.subplot(4,2,3)

model2 = DecisionTreeClassifier(max\_depth=2).fit(x, y)

dt.dimensionchange(model2, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x1,x2, c=y, cmap='RdYlGn')

plt.title('max depth: 2', fontsize=20)

plt.grid()

data = tree.export\_graphviz(

    model2, out\_file='2.dot',

    filled=True, rounded=True,

    special\_characters=True

)

graph = graphviz.Source(data)

plt.subplot(4,2,4)

image2 = img.imread('2.png')

plt.title('Tree Expression with Depth=2', fontsize=20)

plt.imshow(image2)

plt.subplot(4,2,5)

model3 = DecisionTreeClassifier(max\_depth=3).fit(x, y)

dt.dimensionchange(model3, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x1,x2, c=y, cmap='RdYlGn')

plt.title('max depth: 3', fontsize=20)

plt.grid()

data = tree.export\_graphviz(

    model3, out\_file='3.dot',

    filled=True, rounded=True,

    special\_characters=True

)

graph = graphviz.Source(data)

plt.subplot(4,2,6)

image3 = img.imread('3.png')

plt.title('Tree Expression with Depth=3', fontsize=20)

plt.imshow(image3)

plt.subplot(4,2,7)

model4 = DecisionTreeClassifier(max\_depth=4).fit(x, y)

dt.dimensionchange(model4, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x1,x2, c=y, cmap='RdYlGn')

plt.title('max depth: 4', fontsize=20)

plt.grid()

data = tree.export\_graphviz(

    model4, out\_file='4.dot',

    filled=True, rounded=True,

    special\_characters=True

)

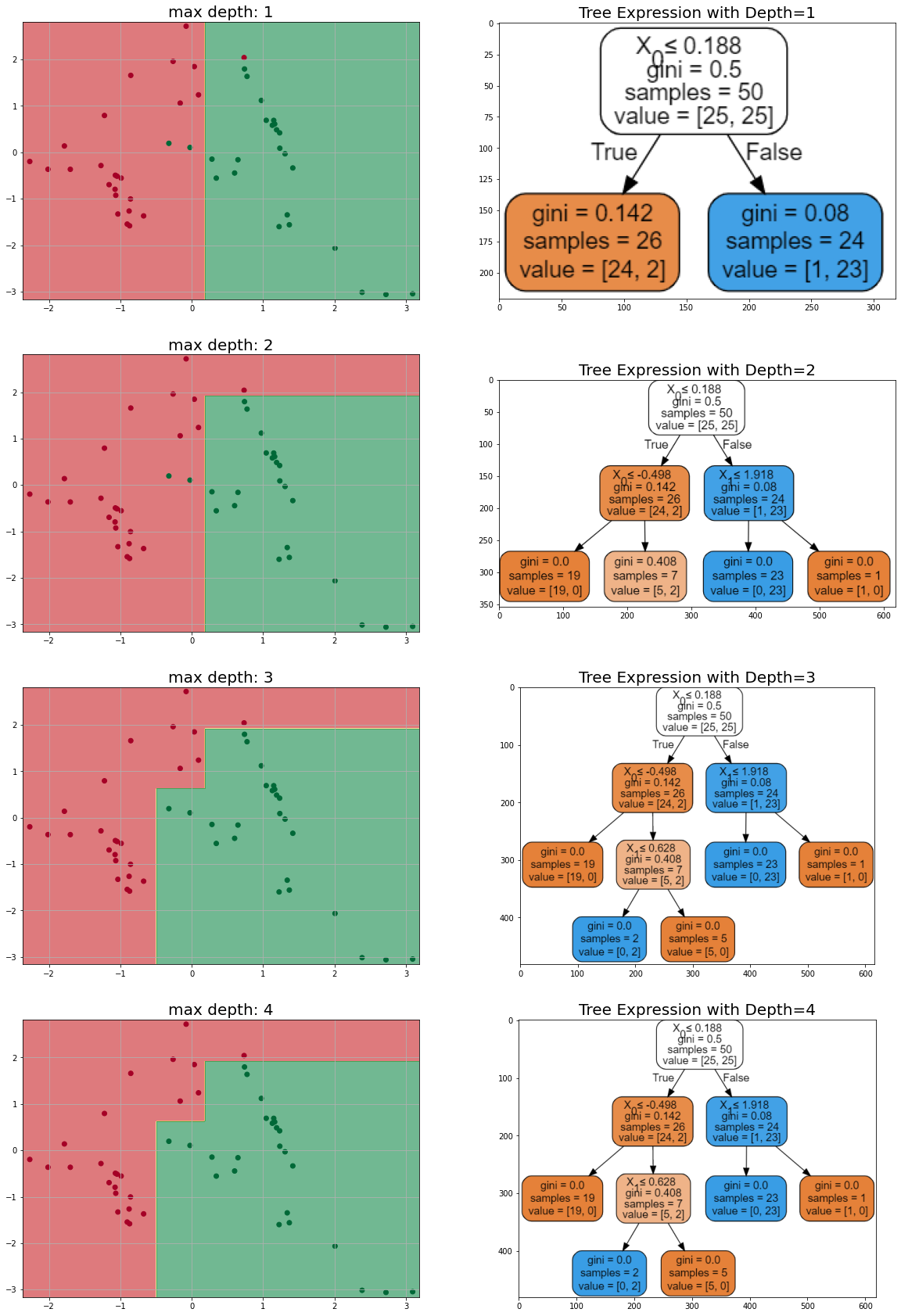
graph = graphviz.Source(data)

plt.subplot(4,2,8)

image4 = img.imread('4.png')

plt.title('Tree Expression with Depth=4', fontsize=20)

plt.imshow(image4)



Notice that accordingly to the increase in max depth, the number of intermediate nodes and terminal nodes increase (until reaching maximum purity level at depth 3). This is done by adding a line (as we have seen in the scatter diagram before, T-1, T-2, and so on), generating additional partitioned subsets. The initial impurity level calculated by Gini Index is 0.5, which is the maximum level of impurity.

At depth 1, you can see that samples (N=50) are divided evenly (26:24), with certain amount of impurity (0.142 and 0.08), which is quite low considering the range of Gini Index (0~0.5). Also, notice that the values ([24,2], [1,23]) indicate how many distinguished category data (misclassifications) are within the subset area (2 greens in red area, and 1 red in green area).

As the tree model gets to the depth of 3 (3rd figure), Gini Index at all terminal nodes gets to 0.0, which means that maximum purity level is reached at depth 3. Values are entirely partitioned into either left or right ([19,0], [0,2], [5,0], [0,23], [1,0]). However, small number of total samples (e.g., N=2 for the second terminal node and N=1 for the fifth terminal node) may indicate the possibility of having overfitting problems when test data are presented.

7. Decision Tree Algorithm – Various Sample Applications

Now let us generate various types of samples including classification data and continuous data to evaluate the general performance of decision tree algorithm.

#Various types of sample generation and model performance visualization

plt.figure(figsize=(20,30))

plt.subplot(6,2,1)

np.random.seed(123)

x = np.random.rand(30, 2)

y = np.zeros( len(x) )

y[-3/4 + 1/2 \* x[:, 0] + 1 \* x[:, 1] > 0] = 1

x1, x2 = x[:,0], x[:,1]

plt.scatter(x1, x2, c=y, cmap='RdYlGn')

plt.title('classification data', fontsize=20)

plt.grid()

plt.subplot(6,2,2)

samplemodel1 = DecisionTreeClassifier().fit(x, y)

dt.dimensionchange(samplemodel1, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x1, x2, c=y, cmap='RdYlGn')

plt.grid()

plt.subplot(6,2,3)

x, y = datasets.make\_circles(n\_samples=500, noise=0.1, random\_state=123)

x1, x2 = x[:,0], x[:,1]

plt.scatter(x1, x2, c=y, cmap='RdYlGn')

plt.title('circular data', fontsize=20)

plt.grid()

plt.subplot(6,2,4)

samplemodel2 = DecisionTreeClassifier().fit(x, y)

x1, x2 = x[:, 0], x[:, 1]

dt.dimensionchange(samplemodel2, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='RdYlGn')

plt.grid()

plt.subplot(6,2,5)

np.random.seed(123)

x = np.random.uniform(-1,1, (400,2))

y = np.zeros((400), dtype=np.int32)

y[ x[:,0] \* x[:,1] > 0] = 1

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='RdYlGn')

plt.title('group data - 1', fontsize=20)

plt.grid()

plt.subplot(6,2,6)

samplemodel3 = DecisionTreeClassifier().fit(x, y)

x1, x2 = x[:, 0], x[:, 1]

dt.dimensionchange(samplemodel3, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='RdYlGn')

plt.grid()

plt.subplot(6,2,7)

x, y = datasets.make\_blobs(n\_samples=500, center\_box=(-1,1), centers=6, cluster\_std=0.1, random\_state=20)

y = y % 2

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='RdYlGn')

plt.title('group data - 2', fontsize=20)

plt.grid()

plt.subplot(6,2,8)

samplemodel4 = DecisionTreeClassifier().fit(x, y)

x1, x2 = x[:, 0], x[:, 1]

dt.dimensionchange(samplemodel4, x1, x2, cmap='RdYlGn', alpha=0.6)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='RdYlGn')

plt.grid()

plt.subplot(6,2,9)

np.random.seed(410)

xx = np.linspace(-1, 1, 500)

x = np.linspace(-1, 1, 50)

y\_train = x \*\* 2 + 0.8 \* x + 1.5 + 0.55 \* (2 \* np.random.rand(len(x)) - 1)

y\_test = x \*\* 2 + 0.8 \* x + 1.5 + 0.55 \* (2 \* np.random.rand(len(x)) - 1)

plt.plot(x, y\_train, 'o', label='train')

plt.plot(x, y\_test, 'xr', label='test')

plt.grid()

plt.title('continuous data - 1', fontsize=20)

plt.legend()

plt.subplot(6,2,10)

samplemodel5 = DecisionTreeRegressor(max\_depth=3).fit(x.reshape(-1,1), y\_train)

plt.plot(x, y\_train, 'o', label='train')

plt.plot(x, y\_test, 'xr', label='test')

plt.plot(xx, samplemodel5.predict(xx.reshape(-1,1)), c='r', linewidth=3)

plt.grid()

plt.legend()

plt.subplot(6,2,11)

np.random.seed(1234)

x\_train = np.linspace(0, 1, 80)

y\_train = np.sin(1.8 \* np.pi \* x\_train) + (np.random.randn(80)/10)

x\_test = np.linspace(0, 1, 20)

y\_test = np.sin(1.8 \* np.pi \* x\_test) + (np.random.randn(20)/10)

plt.plot(x\_train, y\_train, 'o', label='train')

plt.plot(x\_test, y\_test, 'xr', label='test')

plt.legend()

plt.title('continuous data - 2', fontsize=20)

plt.grid()

plt.subplot(6,2,12)

samplemodel6 = DecisionTreeRegressor(max\_depth=4).fit(x\_train.reshape(-1,1), y\_train)

plt.plot(x\_train, y\_train, 'o', label='train')

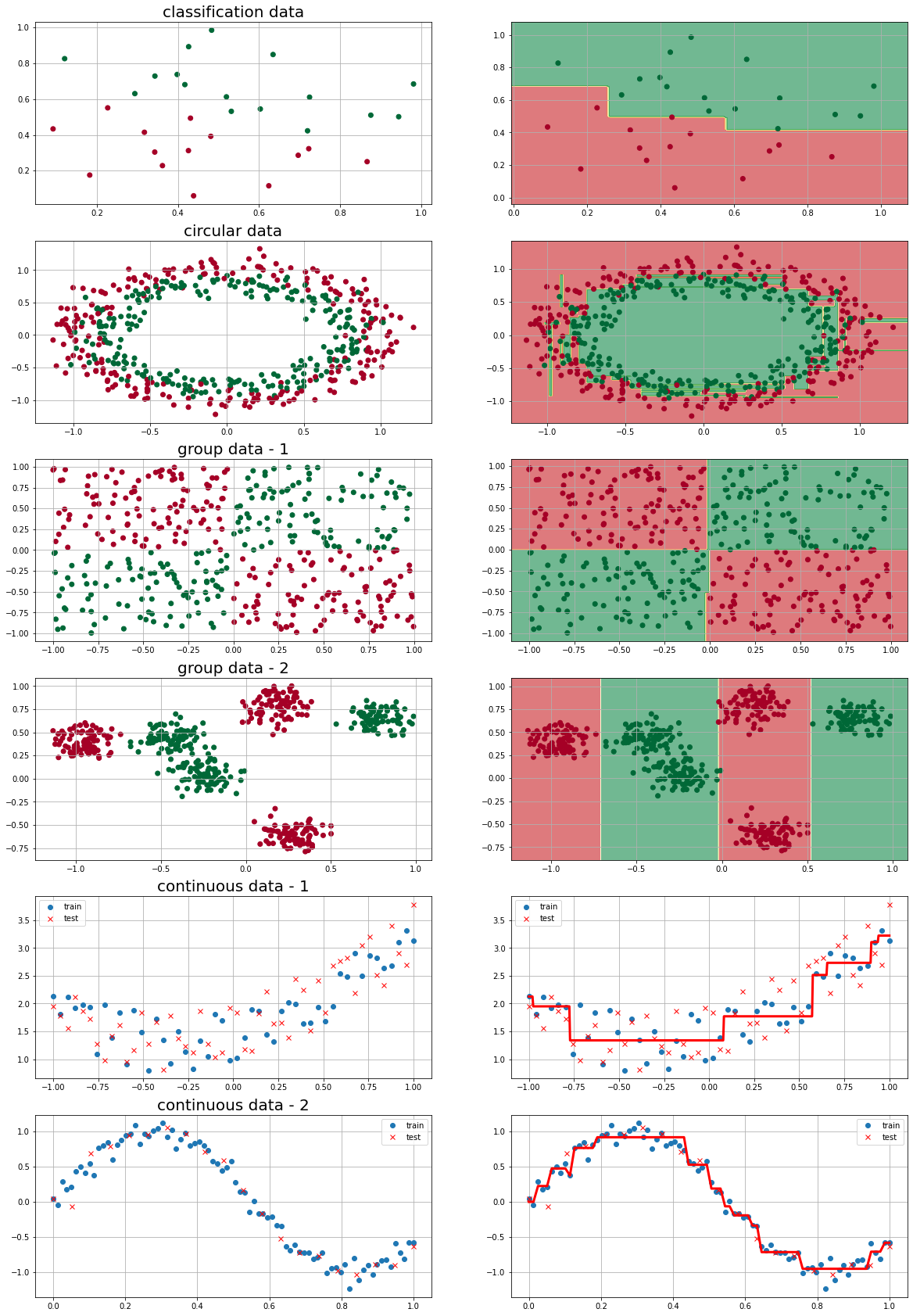
plt.plot(x\_test, y\_test, 'xr', label='test')

plt.plot(x\_train, samplemodel6.predict(x\_train.reshape(-1,1)), c='r', linewidth=3)

plt.legend()

plt.grid()

plt.show()



8. Main Limitations of the Individual Tree Model

As illustrated above through applying tree models on various types of data structures, decision tree algorithm is well-known for its model accuracy and prediction performance. However, there are few critical limitations which calls for an alternative solution.

First, the peculiar characteristic of decision trees, namely the hierarchical structure starting from root nodes to terminal nodes result in venerability to errors in the intermediate stage. Since all nodes are connected to the root node while trickling down to the terminal node, errors in the intermediate stage is emitted to the next level, without corrections. Second, in a related context, individual decision tree model is affected sensitively by even few numbers of noises and minor fluctuations of learning data. Third and finally, as I have illustrated earlier, overfitting possibilities increase accordingly to the increasing number of depth due to smaller samples being included for each subset region.

These problems call for an alternative solution, namely, the ***Random Forest*** Method, which as implied from its name per se uses multiple number of individual tree models to generate a summative (i.e., ensemble) model.

9. Random Forest Model

Random forest model can be classified as one type of methodology from ***Ensemble***, which is a method of boosting prediction accuracy via taking majority voting or average calculation on multiple ***base model*** results, as follows:

Base1(x)=1

Base2(x)=0

Base3(x)=1

Base4(x)=1

Ensemble(x)=1

In order for the resulting ensemble model to have better prediction performance than the base models, two prerequisite conditions should be satisfied: (1) base models should be independent to each other. (2) each base model should have at least better performance than the random prediction model (e.g., random model’s error rates of 0.5 for binary classification problems). In other words when the error rate of the ensemble model is defined as:

When e is defined as the error rates of the base model and,

N is defined as the number of base models,

Ensemble model’s error rate is minimized at the point where consisting base models’ error rates are lower than the random model. Thus, for instance in binary classifications, base models with error rates lower than 0.5 (random model’s error rate) should be selected to construct an ensemble model.

Decision tree models covered in chapter 8 are particularly known to be useful as the base model for an ensemble model, and the reason is twofold. First, decision tree models have low computation al complexity, which enables constructing a model rapidly even in cases for enormous data sizes. Second, decision tree models are non-parametric, which means no prerequisites regarding the distribution of data (e.g., normal distribution) are required in prior to generating a model. In this context, Random Forest model refers to the particular type of ensemble model that utilizes decision tree model as the base model. Generally, random forest model depicts higher prediction accuracy than the individual tree model, and is widely used as the feature selection method for high-dimensional data.

The general sequence of random forest model is as follows. First, multiple numbers of training data are generated via ***bootstrapping*** technique, which we will cover in depth soon. Second, decision tree models are constructed based on the generated training data (using random variables). Third and finally, prediction results are ensembled together to generate a final prediction result.

Two core ideas are attached to the concept of random forest model: **diversity** and **randomness**. First, multiple numbers of training data are generated, and individual decision tree models are constructed for each dataset. This process is referred to as **bagging (Bootstrap Aggregating)**. Second, in the process of constructing a decision tree model, variables are selected randomly. This process is referred to as **random-subspacing**.

10. Bootstrapping

Bagging refers to the process of aggregating each model generated from bootstrap samples. In order to understand the concept of bagging, therefore, prior understanding on bootstrap is necessary. Bootstrapping refers to the sampling technique of generating as many data as the original data by ***sampling with replacement***. Each individual dataset generated under this procedure is called bootstrap set. Consider following illustration for better understanding.

In the illustration above, first notice that each bootstrap set has an **equal** **total** **number** **of data** to the original dataset. In bootstrap set 1, x2 is selected twice, while x5 is not selected. In bootstrap set 2, x3 is selected twice and x4 is not selected. This happens due to **sampling with replacement**. Each datum can be either selected multiple times, or not selected at all. Theoretically, the probability of a certain datum not being selected at all for a bootstrap can be calculated as:

which is around 0.367 (36.7%).

11. Bootstrap Aggregating (Bagging)

Now, suppose that we acquired 20 bootstrap sets for a binary classification problem (0 or 1). Using these sets, we can create 20 individual models, with P(Y=1) for each test instance, and final predicted class label (which will be either 0 or 1). When aggregating the results (bagging) for a classification problem, the ensemble model takes majority voting on the predicted class labels, as follows:

For instance, if N is 20, and we got:

Alternatively, it would be possible to take a weighted voting method instead of simple majority voting to incorporate the training accuracies of each individual model, as follows:

Or, it is possible to focus on the predicted probability of the test instances (P(y=1)), which would be:

12. Random Subspace

Now that we have satisfied diversity condition via bagging process, next is to meet the requirements for randomness condition. In order for individual decision tree model to be ***independent*** to each other, the generating process of each model should be based on randomness, which can be done via random subspacing.

Throughout bootstrapping, we have acquired N number of training data from the original training data; and for each training data, suppose we have generated N number of individual tree models. Normally in order to make a decision tree model, we find dividing variables and dividing points by searching all variables (X1~Xn).

Instead in random subspacing, we randomly select limited number of input variables among the total number of input variables to construct a model, and select a dividing variable and point among that limited pool of input variables. The procedure of selecting a dividing variable and point per se is the same with the procedures for making an individual tree model. Full-grown tree can be generated by repeating this process trickling down to each node.

The core point is that we consider only the **subset of variables** for each step instead of considering all variables. In other words, we randomly select smaller number of subset variables from the total number of variables in terms of exploring branch point of the decision tree model, and only consider those variables in terms of making the model.

13. Generalization Error

The two prerequisites, i.e., diversity via bagging and randomness via random subspaces, work as the very legitimacy of random forest model. Although each individual tree can be vulnerable to overfitting problems, strong **law of large numbers** achieved from the process of satisfying the two preconditions prevents overfitting, and the errors converge into limiting values. This is to say, generalization error of a random forest model can be defined as:

where p refers to the average correlation between each decision tree and s refers to the average of gaps between the number of correctly predicted trees and wrongly predicted trees.

As p shows the correlation rate between each individual decision tree, it can be inferred from the equation that the lower value of p (which means that each tree is independent to each other) would be better. Also, s would increase accordingly to the individual tree’s prediction accuracy, which means larger s and thus smaller [(1-s\*\*2)/s\*\*2] would be better.

In sum, two insights can be extracted from the generalization error equation. In order to minimize the error of a random forest model and to maximize model performance, we should either (1) minimize the value of p by minimizing the correlation rate between each individual tree model, and/or (2) maximize the value of s by concentrating on the prediction accuracies of each individual model.

14. Feature Selection

Looking back on linear regression or logistic regression that were covered in the earlier chapters, each model supposes a certain type of probability distribution (e.g., normal distribution for regression models, and Bernoulli distribution for logistic regression). Thus it is possible to get an information regarding the statistical significance of each individual variable, which comes in the form of coefficients or weights. Random forest model, on the other hand, does not provide specific information regarding the statistical significance of a variable, which is because, as we covered earlier, the model is basically non-parametric (i.e., the model does not suppose a certain probability distribution).

Thus, the significance of a variable is determined indirectly using the concept of Out of Bag Errors (OOB Errors). Suppose that we want to figure out the significance of X(i) variable. Out of Bag data refer to the set of data that have not been included in the bootstrap sets. These data are then applied to the made tree models, from Tree(1), Tree(2), to Tree(t), to calculate OOB errors r(i), which can be denoted as:

since there are t number of generated tree models. OOB errors are then calculated on the data subset with randomly mixed values of X(i) variable, and thus we get:

In most cases, therefore, we can assume that e(i) will have larger values than r(i), and thus the difference of the two values can be denoted as:

And the average of such differences can be denoted as:

and the variance of differences can be denoted as:

Then, let us consider the meaning of ‘d’ that we just calculated. When the value of d (in other words, e – r) for a certain variable X(i) is large, this would mean that the error rates will increase greatly if that variable is replaced by other values, meaning that X(i) is significant by the extent of d. Thus, the variable significance of X(i) can be denoted as:

where ‘d’ becomes the main criteria for determining the significance of a variable and the variance S of d works as the penalty for scaling reasons.

15. Demonstrations

Now that we have covered the main concepts of the random forest model, let me demonstrate the procedures ***manually*** using samples. Rather than importing the random forest library directly, the following manual steps will help reviewing the procedures and principles fundamentally.

#Import Essential Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier

from collections import Counter

from sklearn.metrics import accuracy\_score

from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

import sys

sys.path.append('C:\\Users\\Master\\Desktop\\dataTools')

import dataTools as dt

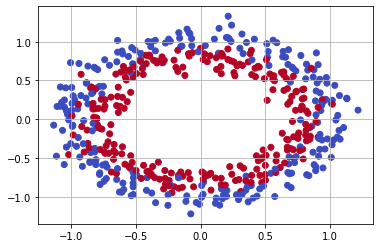
#Sample Generation

x, y = datasets.make\_circles(n\_samples=500, noise=0.1, random\_state=123)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='coolwarm')

plt.grid()

plt.show()



#Bootstrapping (Manually) - sampling with replacement

subsample = []

for i in range(11):

    index = [x for x in range(x.shape[0])]

    sampling = np.random.choice(index, x.shape[0]//2, replace=True)

    subsample.append(sampling)

subsample

OUTPUT:

[array([304, 186, 460, 123, 320, 388, 423, 328, 17, 248, 226, 22, 95,

386, 8, 447, 147, 72, 122, 338, 497, 430, 367, 8, 335, 309,

91, 71, 396, 159, 452, 127, 300, 3, 276, 161, 318, 189, 182,

230, 200, 421, 405, 394, 236, 89, 286, 326, 0, 189, 90, 370,

299, 154, 196, 89, 266, 417, 493, 338, 145, 34, 220, 51, 309,

390, 284, 114, 72, 352, 322, 192, 479, 283, 180, 181, 302, 405,

195, 280, 52, 265, 175, 57, 151, 101, 226, 308, 416, 118, 488,

364, 110, 373, 376, 280, 253, 336, 360, 207, 22, 182, 224, 462,

182, 267, 431, 252, 0, 83, 57, 4, 116, 131, 43, 494, 369,

251, 68, 341, 23, 136, 486, 468, 71, 205, 6, 474, 359, 81,

45, 455, 106, 212, 203, 152, 419, 124, 23, 334, 255, 62, 281,

221, 399, 490, 432, 93, 491, 428, 309, 157, 112, 346, 143, 302,

16, 151, 334, 140, 451, 317, 443, 340, 410, 302, 216, 99, 492,

369, 97, 98, 498, 173, 466, 295, 227, 382, 362, 223, 226, 231,

198, 450, 262, 236, 151, 52, 204, 27, 497, 153, 286, 416, 419,

113, 64, 14, 107, 89, 365, 48, 456, 422, 19, 226, 482, 332,

99, 462, 136, 389, 467, 438, 140, 29, 36, 358, 96, 407, 97,

376, 458, 38, 494, 372, 275, 395, 37, 105, 352, 155, 351, 309,

485, 298, 55, 60, 328, 327, 134, 495, 79, 300, 412, 483, 65,

382, 124, 154]),

array([161, 422, 278, 265, 439, 318, 73, 238, 109, 180, 413, 72, 175,

184, 165, 46, 454, 237, 129, 497, 487, 429, 87, 5, 255, 211,

116, 101, 297, 168, 485, 293, 1, 92, 402, 198, 155, 335, 378,

283, 249, 455, 1, 55, 386, 313, 471, 454, 355, 193, 47, 457,

168, 135, 295, 483, 83, 441, 409, 348, 14, 284, 93, 27, 300,

[show more (open the raw output data in a text editor) ...](vscode-file://vscode-app/c:/Users/Master/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)

372, 401, 172, 418, 62, 55, 131, 168, 231, 357, 493, 64, 146, 239, 398, 47, 195, 156, 418, 165, 113, 4, 63, 201, 180, 219, 433, 89, 190, 143, 73, 298, 452, 86, 301, 452, 22, 260, 339, 216, 135, 80, 14, 488, 200, 268, 49, 379, 174, 36, 377, 133, 394, 119, 366, 471, 424, 4, 213, 386, 246, 477, 17, 57, 67, 263, 271, 398])]

#Bagging (manually)

bagging = []

for index in subsample:

    x\_train = x[index]

    y\_train = y[index]

    model = DecisionTreeClassifier().fit(x\_train, y\_train)

    bagging.append(model.predict(x))

display(len(bagging))

display(len(bagging[0]))

OUTPUT:

11

500

bagging\_prediction = []

for i in range(len(bagging[0])):

    total = []

    for j in bagging:

        total.append(j[i])

    cnt = Counter(total)

    bagging\_prediction.append(cnt.most\_common(1)[0][0])

print(bagging\_prediction)

OUTPUT:

[1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]

#Evaluation

yhat = np.array(bagging\_prediction)

accuracy\_score(y, yhat)

OUTPUT:

0.934

#Decision Tree + Bagging

base\_model = DecisionTreeClassifier()

baggingmodel = BaggingClassifier(

    base\_estimator=base\_model,

    n\_estimators=11

)

baggingmodel.fit(x,y)

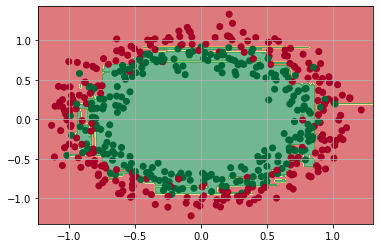
#Result Visualization

dt.dimensionchange(baggingmodel, x[:,0], x[:,1], cmap='RdYlGn', alpha=0.6)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap='RdYlGn')

plt.grid()

plt.show()



#Random Forest (Library)

model = RandomForestClassifier(

    n\_estimators=1000,

    random\_state=12

)

model.fit(x,y)

dt.dimensionchange(model, x[:,0],x[:,1],cmap='RdYlGn',alpha=0.6)

plt.scatter(x[:,0],x[:,1],c=y, cmap='RdYlGn')

plt.grid()

plt.show()

