Week 7: Trees and forests

Introduction

Now we will cover tree-based learning as non-parametric modelling and classification and regression trees. You will also learn ensemble learning (of trees) as a method of optimisation and regularisation techniques to avoid overfitting of models.

Recommended readings

Before reading this week's slides you should read the following chapters from Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.). O'Reilly Media, Inc.

- Chapter 03 Classification (section "Multiclass Classification" onwards)
- Chapter 06 Decision Trees (all sections)
- Chapter 07 Ensemble Learning and Random Forests (all sections)

In the following slides, we summarise some of the important points discussed in the above chapters. We will then focus on the Ed activities based on the content discussed in the chapters.

Decision trees

Decision tree learning algorithms are intuitive, relatively simple to implement, and some of the most widely used learning algorithms. They can be used for learning classification models (with discrete value predictions) and regression models (with continuous value predictions). Most commonly, however, they are used for learning classification models. Consider a dataset below that considers when to play tennis (decision).

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure: A decision tree. Adapted from *Machine Learning* by T. Mitchell, 1997

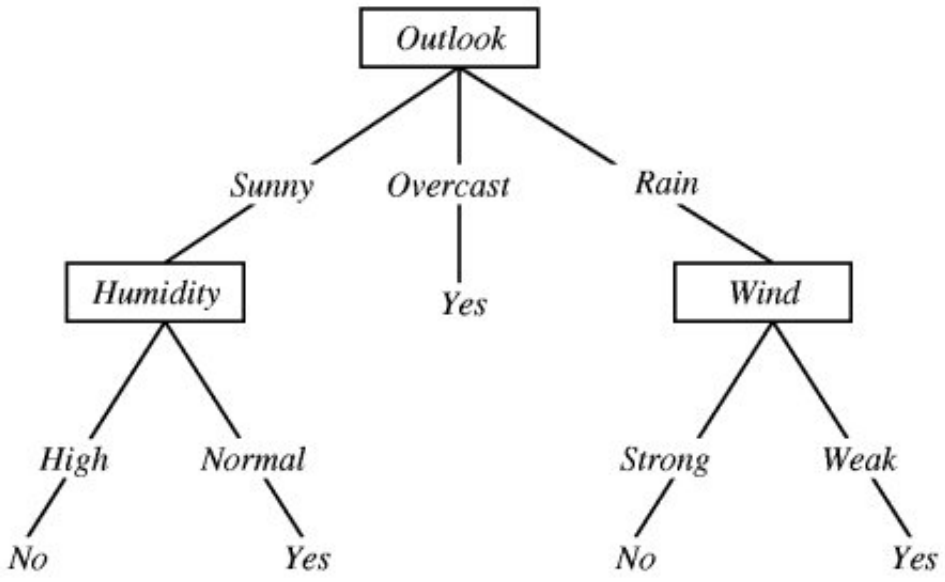
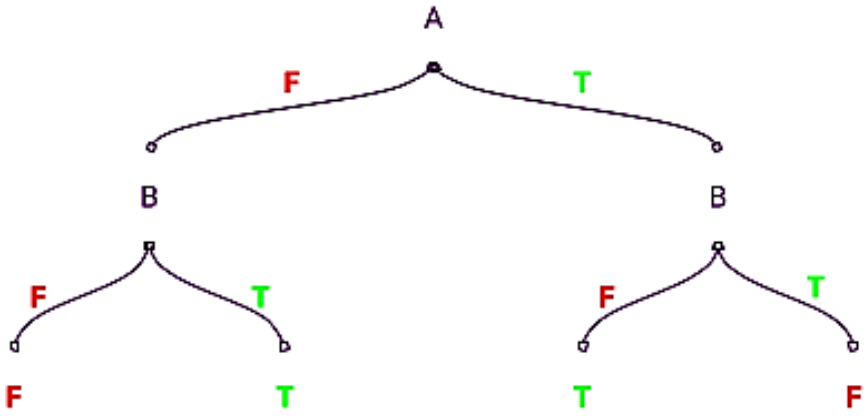


Figure: A decision tree. Adapted from *Machine Learning* by T. Mitchell, 1997

A decision tree takes a set of attribute-value pairs and returns a decision representing the predicted value. In the following decision tree for *PlayTennis* (Mitchell, 1997), each internal node corresponds to a test for one of the attributes (properties), and there is one branch for each possible attribute value. Decision tree expressing logic gates

XOR gate

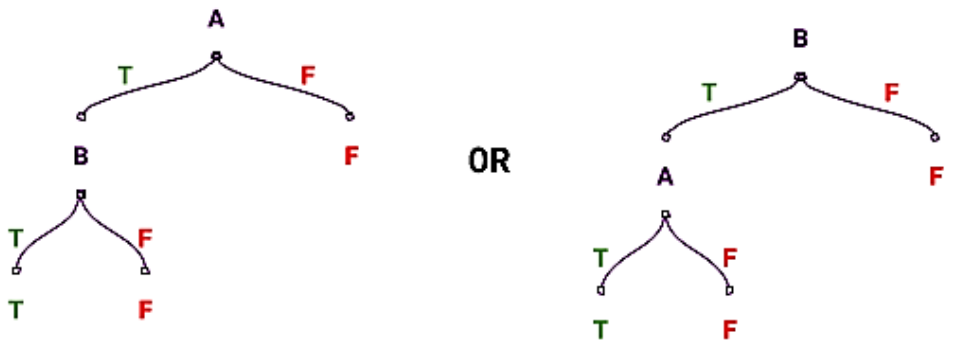
A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F



Source: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/

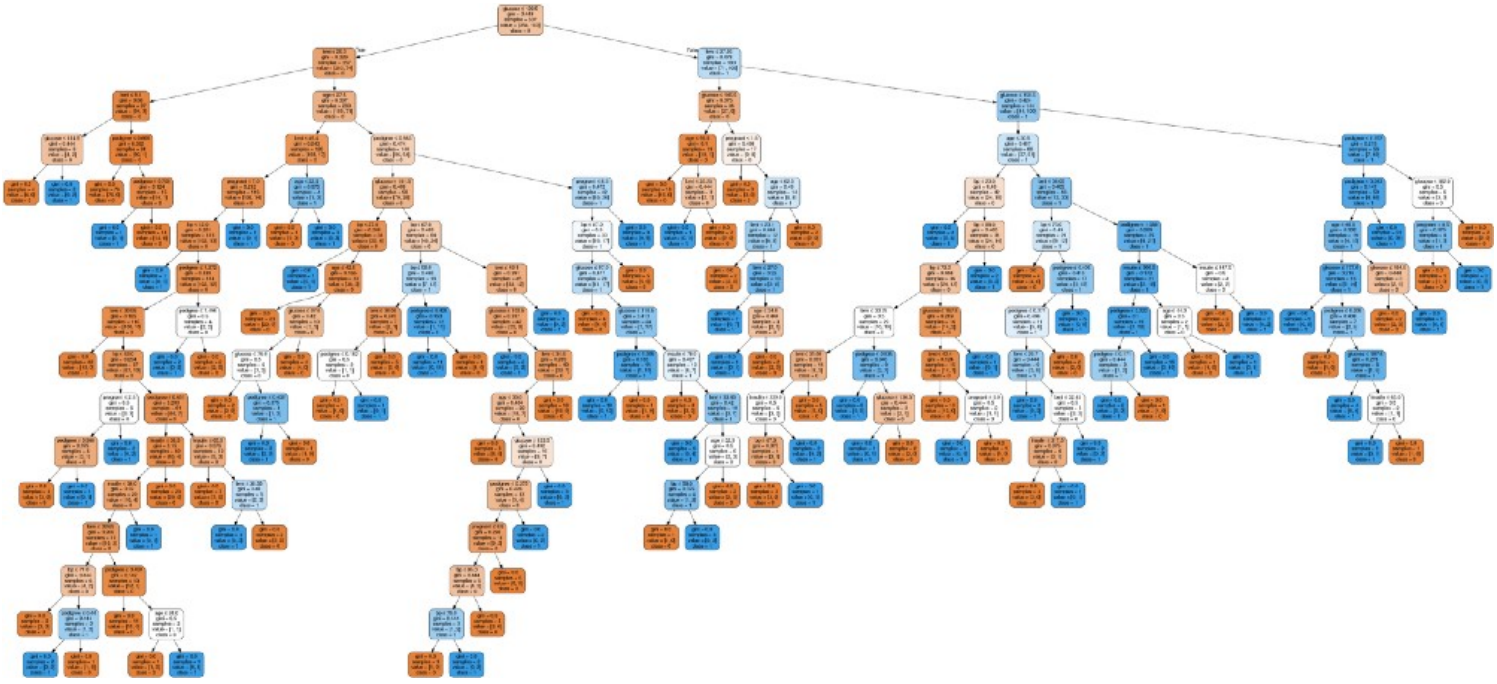
AND gate

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T



Source: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/

Decision tree using Pima Indian Diabetes dataset



Source: https://towardsdatascience.com/machine-learning-basics-descision-tree-from-scratch-part-ii-dee664d46831

References

1. Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106 https://link.springer.com/content/pdf/10.1007/BF00116251.pdf
2. Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies*, *27*(3), 221-234. https://dspace.mit.edu/bitstream/handle/1721.1/6453/aim-930.pdf?sequence=2
3. Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine learning*, *4*(2), 161-186. https://link.springer.com/content/pdf/10.1023/A:1022699900025.pdf
4. Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984). Classification and Regression Trees. Belmont: Wadsworth (hard to get the source)
5. Steinberg, D., & Colla, P. (2009). CART: classification and regression trees. *The top ten algorithms in data mining*, *9*, 179. https://www.researchgate.net/profile/Dan_Steinberg2/publication/265031802_Chapter_10_CART_Classification_and_Regression_Trees/links/567dcf8408ae051f9ae493fe.pdf
6. Loh, W. Y. (2014). Fifty years of classification and regression trees. *International Statistical Review*, *82*(3), 329-348. http://pages.stat.wisc.edu/~loh/treeprogs/guide/LohISI14.pdf slides: http://washstat.org/presentations/20150604/loh_slides.pdf

Learning decision trees (Classification)

Now that we have gone over the basics of Decision Trees, we can discuss how to build a decision tree from a training dataset. We consider a simple dataset with discrete values that considers the play tennis as an outcome given a set of features (attributes) *as shown below.*

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure: Training data set. Adapted from *Machine Learning* by T. Mitchell, 1997

The above dataset has14 training instances which feature 5 negative and 9 positive instances (No and Yes). We first select the ***best attribute*** that splits the given dataset into two or more subsets with '*less impurity*'.

We label a set is ***pure*** if all the instances are either positive or negative. The set is ***impure*** if some examples are positive and others are negative. A set has an *impurity* of 1 if it has 50% positive examples and 50% negative examples, and an *impurity* of 0 if all the examples are either positive or negative. We aim to build a decision tree that decreases impurity amongst all the sets at the leaf nodes.

Entropy

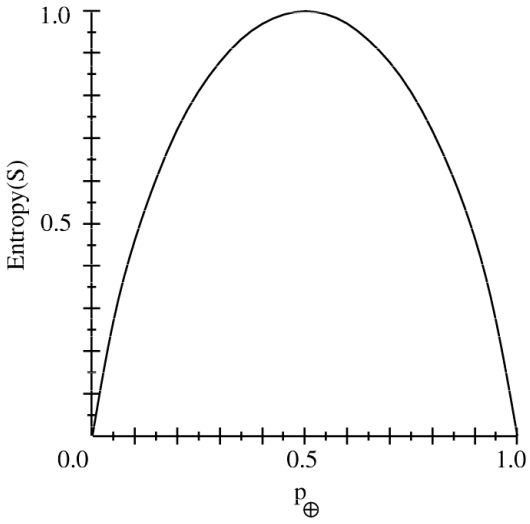
Entropy is a measure the impurity of the dataset which can also be seen as a measure of uncertainty. Our goal in machine learning is to reduce uncertainty or entropy as much as possible. Hence, from the perspective of information theory, given a data set *S, our Entropy(S) is given by*

$$Entropy\,(S) \equiv -p_{\oplus}\log_2 p_{\oplus} - p_{\ominus}\log_2 p_{\ominus}$$

where S $p_{\oplus}$ is the proportion (probability) of positive samples in S and $p_{\ominus}$ is the proportion of negative samples in S. We generalise this further:

$$Entropy(S) = -\sum_i p_i \log_2 p_i.$$

The below figure depicts the entropy function with respect to $p\oplus$.



- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

$$Entropy(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

Figure: Entropy function. Adapted from Machine Learning by T. Mitchell, 1997

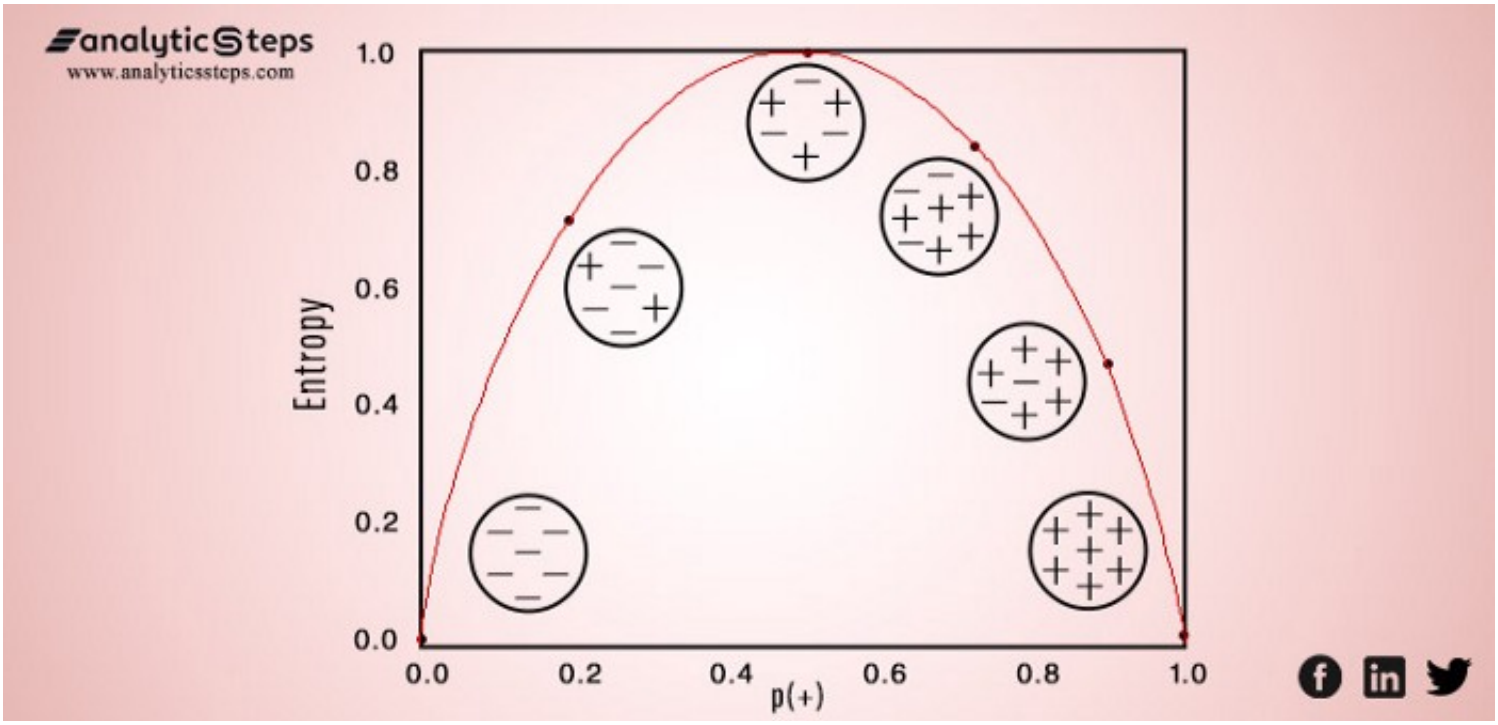


Figure Source: Variation of entropy against data points. https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8

A low entropy value means the data set is highly uniform and a high entropy value means the data set is highly varied. A *pure* set is one in which all examples are of the same class; that is, *yes* or *no* as in the above example.

Information gain

Information gain is the measure of how much information a feature gives about the classes. The major of the Decision Tree algorithm is to maximize information gain. The features that perfectly partitions the data give maximum information and hence with high Information gain, it would be used first for splitting the dataset.

The information gain represents an expected reduction in entropy (impurity) when a data set is split on a selected attribute. Our measure for information gain is given below.

$$Gain\,(S, A) \equiv Entropy\,(S) \; - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy\,(S_v)$$

Tan, M. (1993). Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, *13*(1), 7-33:https://link.springer.com/content/pdf/10.1007/BF00993101.pdf

Let's consider the training data set from the *PlayTennis* example.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Figure: Adapted from *Machine Learning* by T. Mitchell, 1997

We need to calculate the entropy first. Decision column consists of 14 instances and includes two labels: yes and no. There are 9 outcomes labelled as *Yes* and 5 outcomes labelled as *No.*

Entropy(Decision) = – p(Yes) .log_2p(Yes) – p(No) . log_2p(No)

Entropy(Decision) = – (9/14) . log_2(9/14) – (5/14) . log_2(5/14) = 0.940

Now, we need to find the most dominant factor to make a decision for the root note in the tree.

Below you can see a generalised code for Entropy calculation for any attribute given. Currently, on line 9, [-1] points to the last attribute called 'play', and you can change it to 0, 1, 2 etc so that you can see the Entropy calculated for different attributes.

Run PYTHON

```
1  #source: https://medium.com/@lope.ai/decision-trees-from-scratch-using-i
2  import numpy as np
3  import pandas as pd
4  eps = np.finfo(float).eps
5  from numpy import log2 as log
6
7
8  def find_entropy(df):
9      Class = df.keys()[-1]   #To make the code generic, changing target v
10     print(df.keys(), ' list of attributes')
11     print(Class, ' is Class')
12     entropy = 0
13     values = df[Class].unique()
14     for value in values:
```

Code Source: https://medium.com/@lope.ai/decision-trees-from-scratch-using-id3-python-coding-it-up-6b79e3458de4

Wind factor on the Decision outcome

Gain(Decision, Wind) = Entropy(Decision) – ∑ [p(Decision|Wind) . Entropy(Decision|Wind)

Wind attribute has two labels: weak and strong. We would reflect it to the formula.

Gain(Decision, Wind) = Entropy(Decision) – [p(Decision|Wind=Weak) . Entropy(Decision|Wind=Weak)] – [p(Decision|Wind=Strong) . Entropy(Decision|Wind=Strong)]

Now, we need to calculate (Decision|Wind=Weak) and (Decision|Wind=Strong) respectively.

1. Entropy(Decision|Wind=Weak) = – p(No) . log2p(No) – p(Yes) . log2p(Yes)
2. Entropy(Decision|Wind=Weak) = – (2/8) . log2(2/8) – (6/8) . log2(6/8) = 0.811

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
13	Overcast	Hot	Normal	Weak	Yes

1. Entropy(Decision|Wind=Strong) = – p(No) . log2p(No) – p(Yes) . log2p(Yes)
2. Entropy(Decision|Wind=Strong) = – (3/6) . log2(3/6) – (3/6) . log2(3/6) = 1

Day	Outlook	Temp.	Humidity	Wind	Decision
2	Sunny	Hot	High	Strong	No
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
14	Rain	Mild	High	Strong	No

Gain(Decision, Wind) = Entropy(Decision) – [p(Decision|Wind=Weak) . Entropy(Decision|Wind=Weak)] – [p(Decision|Wind=Strong) . Entropy(Decision|Wind=Strong)]

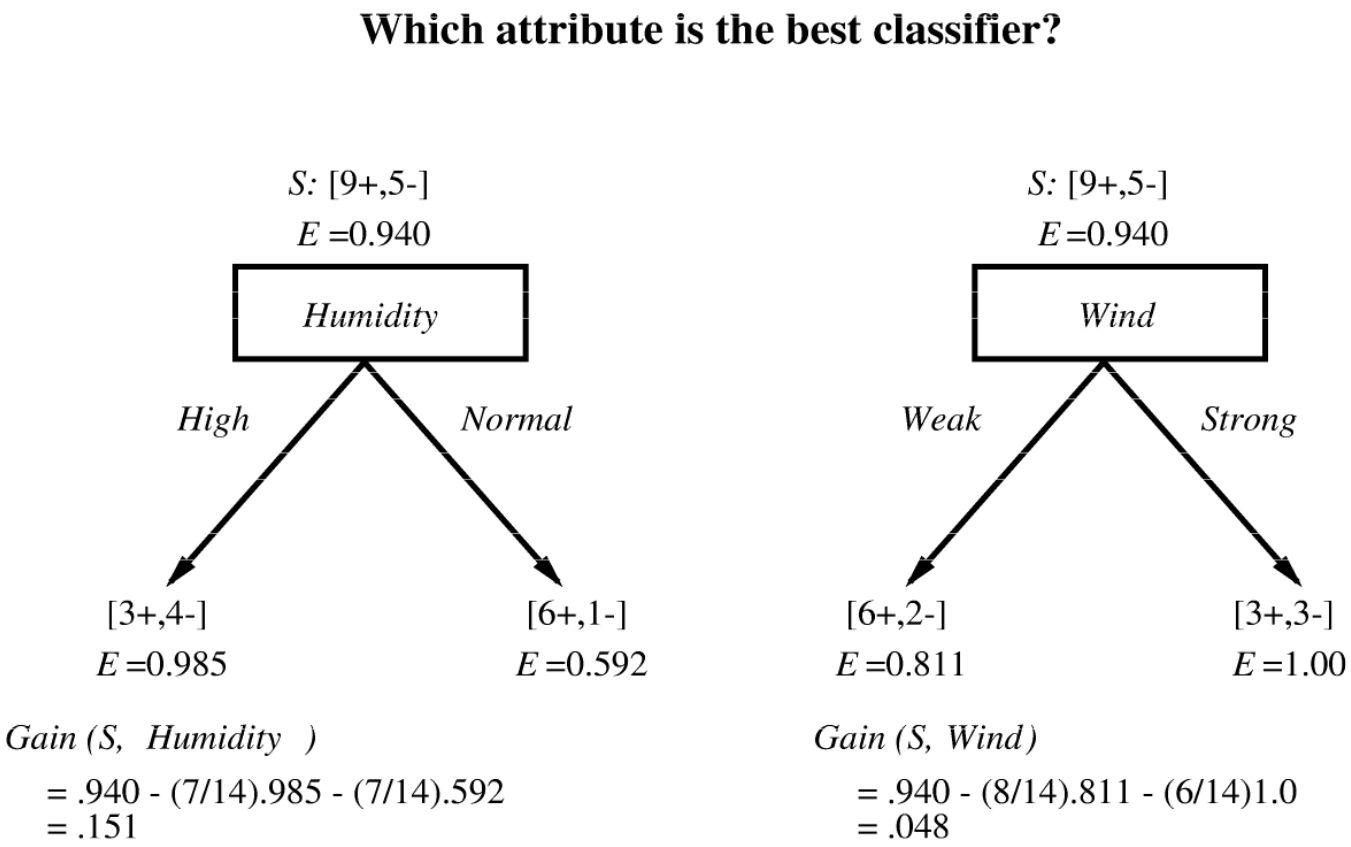
= 0.940 – [(8/14) . 0.811] – [(6/14). 1] = 0.048

Run PYTHON

```
#source: https://medium.com/@lope.ai/decision-trees-from-scratch-using-i
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log


def find_entropy(df):
    Class = df.keys()[-1]   #To make the code generic, changing target v
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
```

Code Source: https://medium.com/@lope.ai/decision-trees-from-scratch-using-id3-python-coding-it-up-6b79e3458de4

For the first split, information gain for the four attributes are shown below:

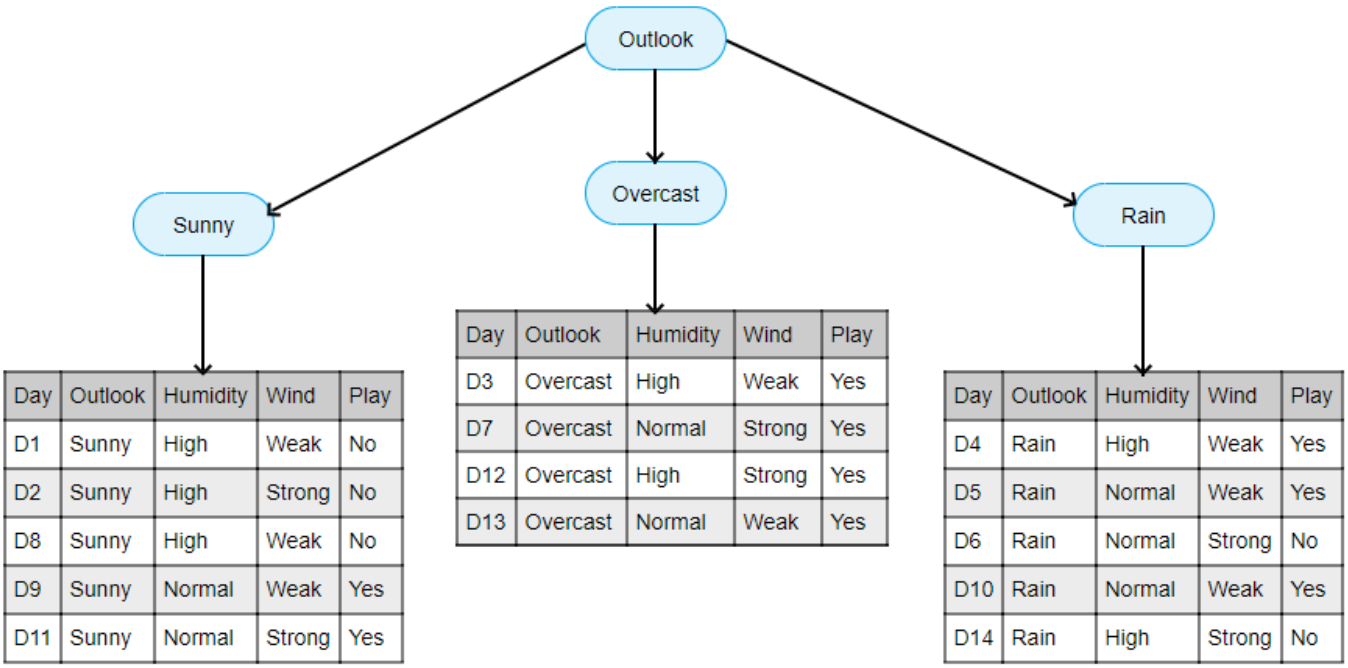


Attribute	Information Gain
Humidity	0.151
Wind	0.048
Outlook	**0.246**
Temp	0.029

Figure: Four attributes. Adapted from *Machine Learning* by T. Mitchell, 1997, Maidenhead; U.K: McGraw Hill.

We select *Outlook* as the first attribute to split the data set because it offers the highest information gain, resulting in the following tree.

For example, for the attribute *Outlook*, there are three branches representing *Sunny*, *Overcast,* and *Rain*. We start from the root of the tree and move downwards following the appropriate branches.



Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes

Day	Outlook	Humidity	Wind	Play
D3	Overcast	High	Weak	Yes
D7	Overcast	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes

Day	Outlook	Humidity	Wind	Play
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D10	Rain	Normal	Weak	Yes
D14	Rain	High	Strong	No

Figure Source: A Comprehensive Guide to Decision Tree Learning.
https://www.aitimejournal.com/@akshay.chavan/a-comprehensive-guide-to-decision-tree-learning

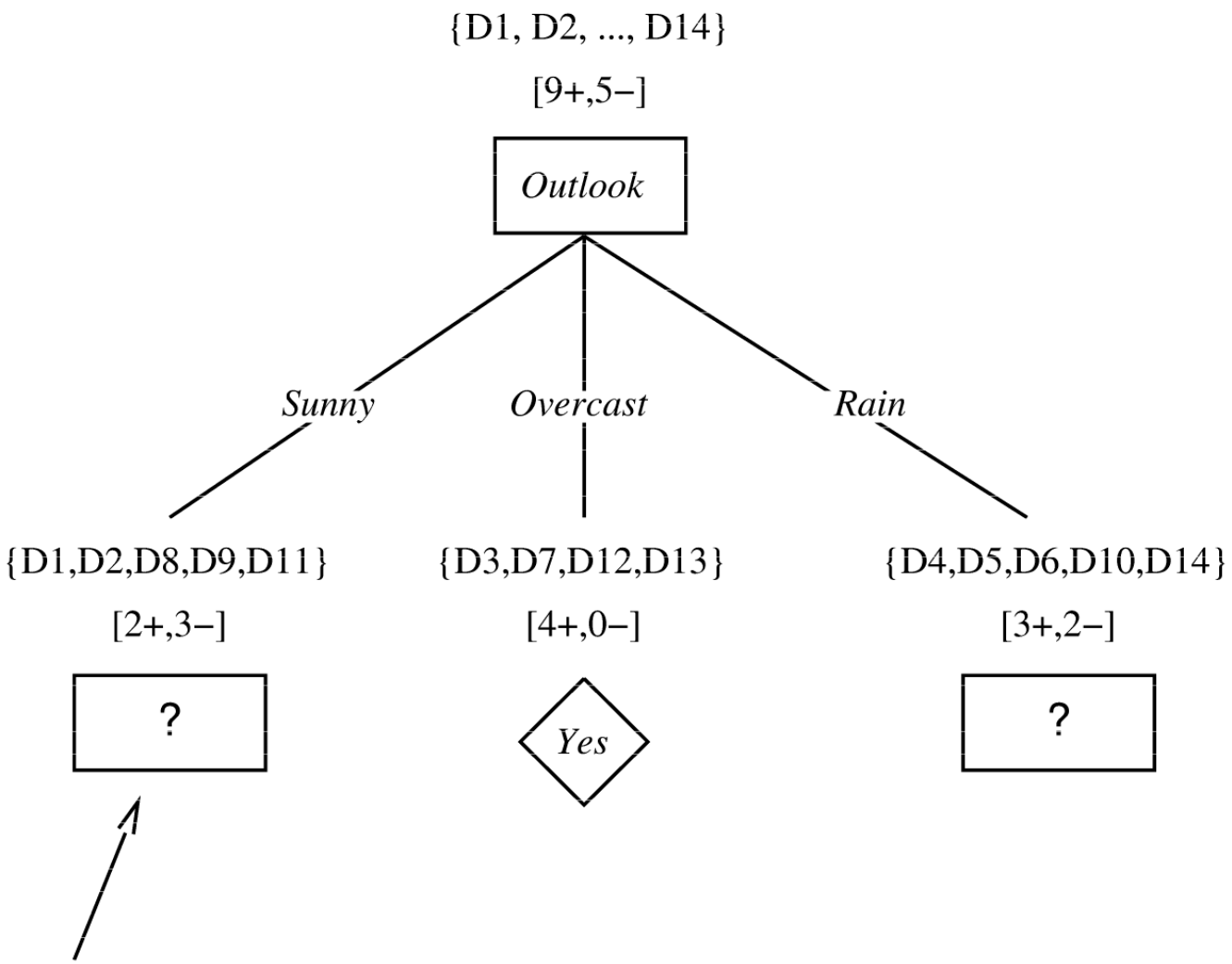
Note that the attribute temperature is not shown in the subsample datasets in above figure, but we can assume it is there.

Let's say we have the following attribute-value pairs representing a present condition:

{*Outlook*: *Sunny*, *Humidity*: *Normal*}

We need to first test the attribute at the root, which is *Outlook* in the following tree. Considering the value of *Outlook* is *Sunny*, we follow the corresponding branch and test the *Humidity* attribute. *Humidity* attribute is *Normal*, so our decision is *Yes (PlayTennis)*. In a decision tree, each leaf node represents a possible decision, i.e. *PlayTennis (yes)* or *PlayTennis (no)*.

Please note that in the branch *Sunny* we only consider cases with *Outlook* value of *Sunny,* that is days $D1, D2, D8, D9, and D11$. And similarly for the other two values *Overcast* and *Rain*.



S_{sunny} = {D1,D2,D8,D9,D11}

$Gain\ (S_{sunny}, Humidity) = .970 - (3/5)\ 0.0 - (2/5)\ 0.0 = .970$

$Gain\ (S_{sunny}, Temperature) = .970 - (2/5)\ 0.0 - (2/5)\ 1.0 - (1/5)\ 0.0 = .570$

$Gain\ (S_{sunny}, Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$

Figure: Branching of attributes. Adapted from *Machine Learning* by T. Mitchell, 1997

What do you think which attribute should be tested at the place pointed by an arrow?

The first option with value 0.970 should be tested.

```
Entropy of Sunny:
```

```
H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))
     = - (2/5) * log2(2/5) - (3/5) * log2(3/5)
     = 0.971
```

In the branch of attribute *Sunny*, we select *Humidity* as the second attribute because it offers the highest information gain for the days with *Sunny* outlook (for the days $D1, D2, D8, D9, and D11$). We only consider attributes that are not considered already on the path from the root node; hence, we do not consider attribute *Outlook* anymore.

```
Temperature Categorical values - hot, mild, cool
H(Sunny, Temperature=hot) = -0-(2/2)*log(2/2) = 0
H(Sunny, Temperature=cool) = -(1)*log(1)- 0 = 0
H(Sunny, Temperature=mild) = -(1/2)*log(1/2)-(1/2)*log(1/2) = 1
Average Entropy Information for Temperature -
I(Sunny, Temperature) = p(Sunny, hot)*H(Sunny, Temperature=hot)
+ p(Sunny, mild)*H(Sunny, Temperature=mild) + p(Sunny, cool)*H(Sunny,
Temperature=cool)
= (2/5)*0 + (1/5)*0 + (2/5)*1
= 0.4
Information Gain = H(Sunny) - I(Sunny, Temperature)
                 = 0.971 - 0.4
                 = 0.571
```

```
Humidity Categorical values - high, normal
H(Sunny, Humidity=high) = - 0 - (3/3)*log(3/3) = 0
H(Sunny, Humidity=normal) = -(2/2)*log(2/2)-0 = 0

Average Entropy Information for Humidity -
I(Sunny, Humidity) = p(Sunny, high)*H(Sunny, Humidity=high)
+ p(Sunny, normal)*H(Sunny, Humidity=normal)
= (3/5)*0 + (2/5)*0
= 0

Information Gain = H(Sunny) - I(Sunny, Humidity)
                 = 0.971 - 0
                 = 0.971
```

```
Wind Categorical values - weak, strong
H(Sunny, Wind=weak) = -(1/3)*log(1/3)-(2/3)*log(2/3) = 0.918
H(Sunny, Wind=strong) = -(1/2)*log(1/2)-(1/2)*log(1/2) = 1

Average Entropy Information for Wind -
I(Sunny, Wind) = p(Sunny, weak)*H(Sunny, Wind=weak) + p(Sunny, strong)*H(Sunny,
Wind=strong)
= (3/5)*0.918 + (2/5)*1
= 0.9508
```

```
Information Gain = H(Sunny) - I(Sunny, Wind)
                 = 0.971 - 0.9508
                 = 0.0202
```

Source of details of calculation: https://iq.opengenus.org/id3-algorithm/

We do not need to split the branch *Overcast,* because it's a pure set. We label a set is ***pure*** if all the instances are either positive or negative.

If you calculate information gains for the branch *Rain*, you will find the attribute with the highest information gain is *Wind*.

```
Complete entropy of Rain is -
H(S) = - p(yes) * log2(p(yes)) - p(no) * log2(p(no))
     = - (3/5) * log(3/5) - (2/5) * log(2/5)
     = 0.971
```

Then we move into the leaves:

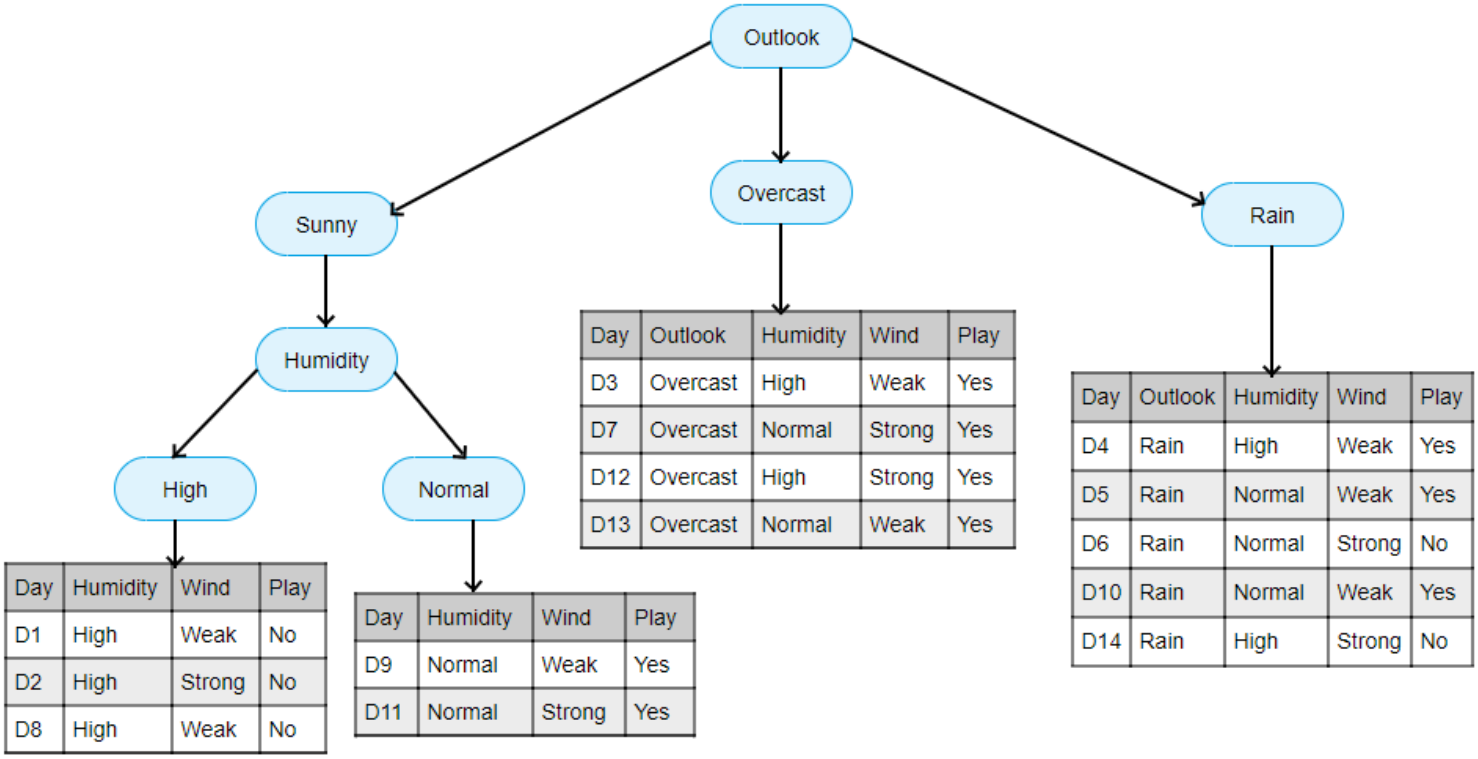
```
Tempertaure Categorical values - mild, cool
H(Rain, Temperature=cool) = -(1/2)*log(1/2)- (1/2)*log(1/2) = 1
H(Rain, Temperature=mild) = -(2/3)*log(2/3)-(1/3)*log(1/3) = 0.918
Average Entropy Information for Temperature -
I(Rain, Temperature) = p(Rain, mild)*H(Rain, Temperature=mild)
 + p(Rain, cool)*H(Rain, Temperature=cool)
= (2/5)*1 + (3/5)*0.918
= 0.9508

Information Gain = H(Rain) - I(Rain, Temperature)
                 = 0.971 - 0.9508
                 = 0.0202
```

```
Wind Categorical values - weak, strong
H(Wind=weak) = -(3/3)*log(3/3)-0 = 0
H(Wind=strong) = 0-(2/2)*log(2/2) = 0

Average Entropy Information for Wind -
I(Wind) = p(Rain, weak)*H(Rain, Wind=weak)
 + p(Rain, strong)*H(Rain, Wind=strong)
= (3/5)*0 + (2/5)*0
= 0

Information Gain = H(Rain) - I(Rain, Wind)
```

```
                = 0.971 - 0
                = 0.971
```

Source of details of calculation: https://iq.opengenus.org/id3-algorithm/

Note: You will notice that the attribute temperature is eliminated as at decision for every note, it was not selected as having higher information gain and then the leaves of those nodes were pure. We can conclude that the temperature does not have a major effect in the outcome of play tennis and was eliminated due to the greedy nature of the tree.



Day	Outlook	Humidity	Wind	Play
D3	Overcast	High	Weak	Yes
D7	Overcast	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes

Day	Outlook	Humidity	Wind	Play
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D10	Rain	Normal	Weak	Yes
D14	Rain	High	Strong	No

Day	Humidity	Wind	Play
D1	High	Weak	No
D2	High	Strong	No
D8	High	Weak	No

Day	Humidity	Wind	Play
D9	Normal	Weak	Yes
D11	Normal	Strong	Yes

Figure Source: A Comprehensive Guide to Decision Tree Learning.
https://www.aitimejournal.com/@akshay.chavan/a-comprehensive-guide-to-decision-tree-learning

Note that the attribute temperature is not shown in the subsample datasets in above figure, but we can assume it is there.

A tree structure offers an easy and intuitive way to explain a decision making process, and it's widely used in describing many "How To" manuals and complex decision-making processes.

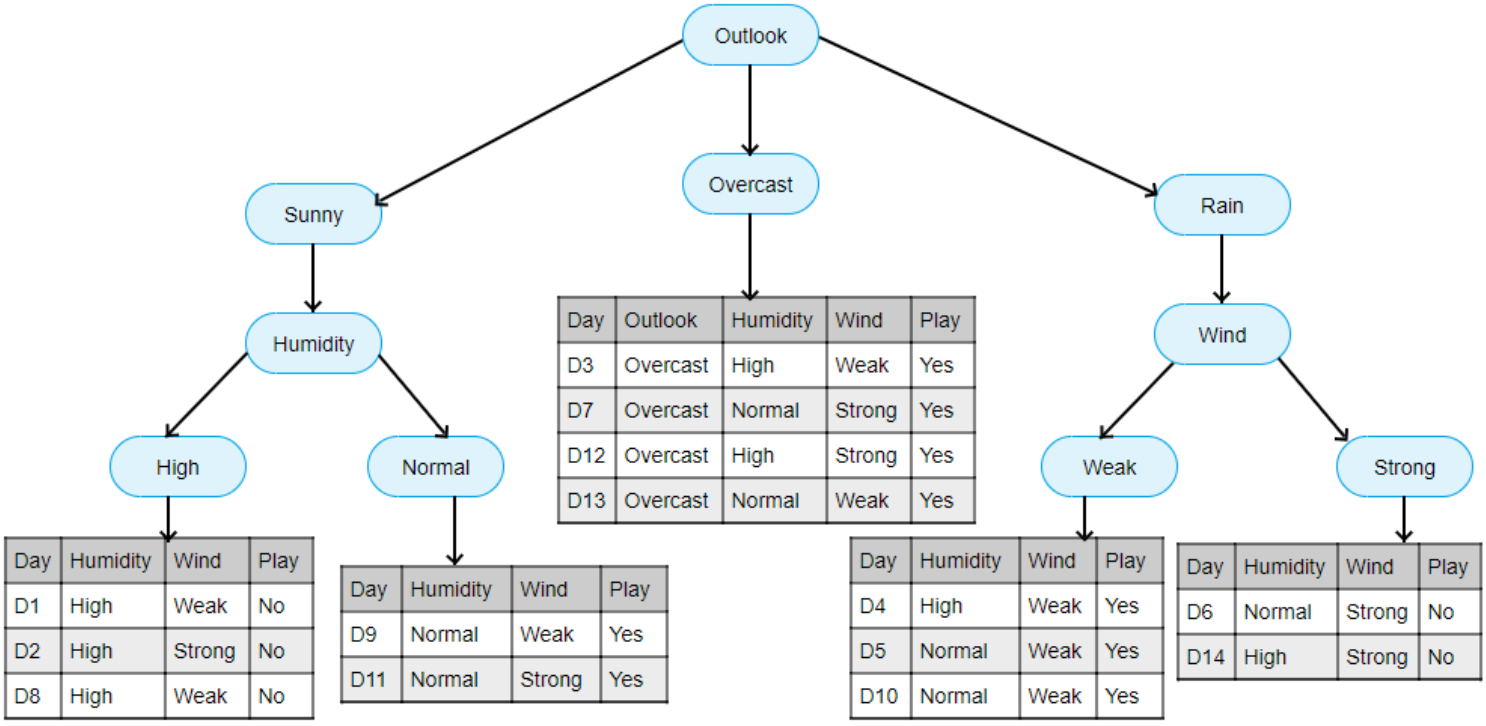


Figure Source: A Comprehensive Guide to Decision Tree Learning.
https://www.aitimejournal.com/@akshay.chavan/a-comprehensive-guide-to-decision-tree-learning

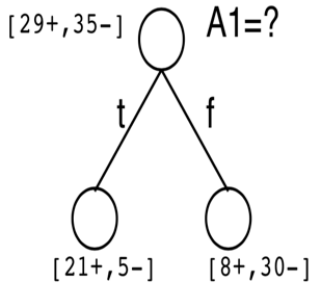
Note that the attribute temperature is not shown in the subsample datasets in above figure, but we can assume it is there.

We can also represent the above decision tree as the following set of rules. Each leaf node (decision) is represented by a rule, and the condition of a rule is a conjunction of all the tests in the path from the root to that leaf node.

```
IF (Outlook = Sunny AND Humidity = High) THEN PlayTennis(No)
IF (Outlook = Sunny AND Humidity = Normal) THEN PlayTennis(Yes)
IF (Outlook = Overcast) THEN PlayTennis(Yes)
IF (Outlook = Rain AND Wind = Strong) THEN PlayTennis(No)
IF (Outlook = Rain AND Wind = Weak) THEN PlayTennis(Yes)
```

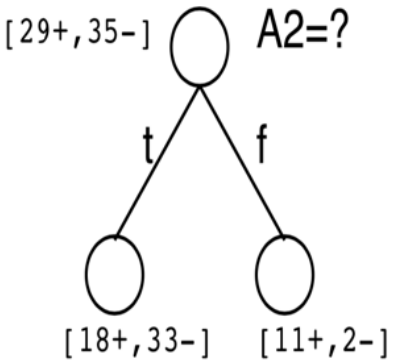
Another Example

Given that we have a training set with 29 positive and 35 negative examples and we have two attributes (features) $A1$ and $A2$. If we split this data set on attribute $A1$, it results in two subsets (for two leaves) as shown below. We have one data set with 21 positive and 5 negative examples, and another with 8 positive and 30 negative examples and hence, the information gain for $A1$ is given below.



$$
\begin{aligned}
Gain(S, A1) &= Entropy(S) - \left(\frac{|S_t|}{|S|}Entropy(S_t) + \frac{|S_f|}{|S|}Entropy(S_f)\right) \\
&= ((\frac{26}{64}(-\frac{21}{26}\log_2(\frac{21}{26}) - \frac{5}{26}\log_2(\frac{5}{26}))) + \\
&\quad (\frac{38}{64}(-\frac{8}{38}\log_2(\frac{8}{38}) - \frac{30}{38}\log_2(\frac{30}{38})))) \\
&= 0.9936 - (\ 0.2869 + 0.4408\ ) \\
&= 0.2658
\end{aligned}
$$

In the case of attribute $A2$, the split and the corresponding information gain are given below.



$$
\begin{aligned}
Gain(S, A2) &= 0.9936 - (\ 0.7464 + 0.0828\ ) \\
&= 0.1643
\end{aligned}
$$

Figure: Splitting dataset. Adapted from *Machine Learning* by T. Mitchell, 1997

Information gain for $A1$ (0.2658) is greater than $A2$ (0.1643), meaning $A1$ offers a larger expected reduction in entropy (impurity), so we choose $A1$ for the split.

ID3 Decision Tree Algorithm

The general idea behind decision tree induction is as follows:

1. **Step 1:** Select the feature (predictor variable) that best classifies the data via *Information Gain* and assign that feature to the root node.
2. **Step 2:** Traverse down from the root node, and make the relevant decision at each internal node such that it best classifies the data.
3. **Step 3:** Return back to step 1 (usually via recursion) and repeat until you assign a class to the input data.

We note that the example shown in this lesson considers **ID3** (Iterative Dichotomiser 3) Decision Tree

as the designated algorithm for developing the decision tree.

- **if** data-set is pure, **then** (e.g. all 'no')
    - construct a leaf having the name of the class (e.g. ‘no')
- **else**
    1. choose the feature with the highest information gain (e.g. ‘colour’)
    2. for each value of that feature (e.g. ‘red’, ‘brown’, ‘green’)

        i) take the subset of the data-set having that feature value

        ii) construct a child node having the name of that feature value (e.g. ‘red’)

        ii) call the function **recursively** on the child node and the subset

Below you can see how the Decision Tree traverses in the hypothesis space.

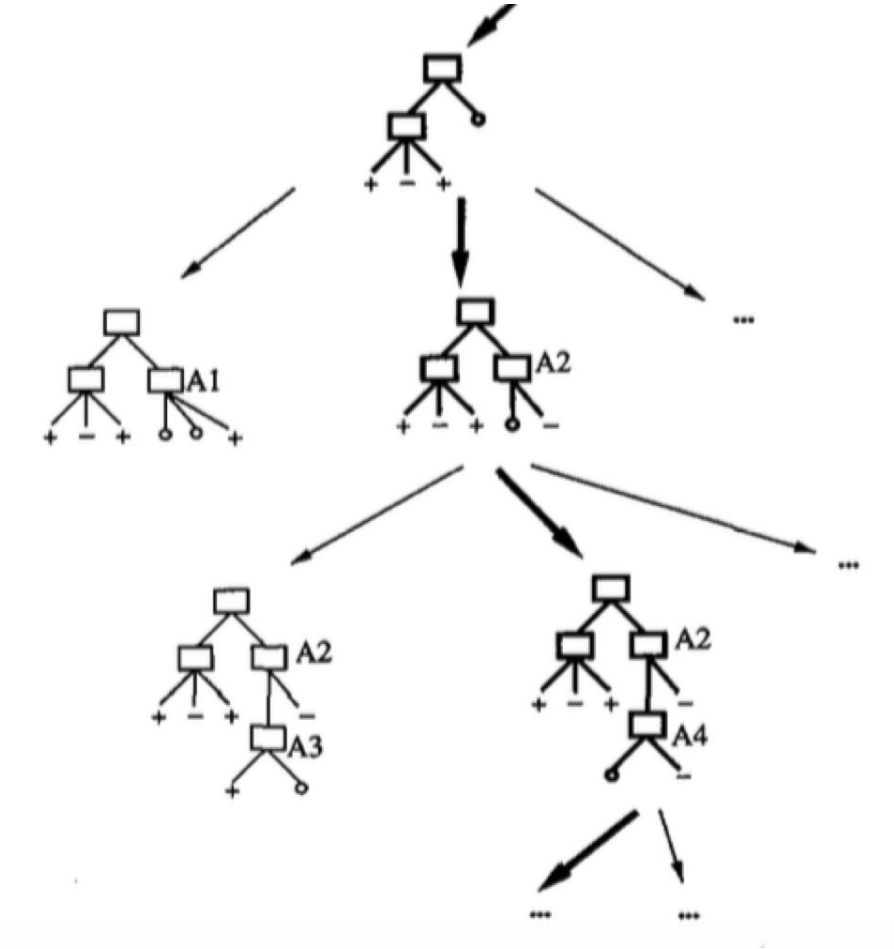


Figure: Decision Tree hypothesis space. Adapted from *Machine Learning* by T. Mitchell, 1997

Now we see code in Python and R that implements the recursive part.

Run PYTHON

```python
#source: https://medium.com/@lope.ai/decision-trees-from-scratch-using-i
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log


def find_entropy(df):
    Class = df.keys()[-1]   #To make the code generic, changing target v
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
```

Code Source: https://medium.com/@lope.ai/decision-trees-from-scratch-using-id3-python-coding-it-up-6b79e3458de4

We see the implementation of Entropy and Information Gain in R in the code below.

Run R

```r
#Source:https://www.r-bloggers.com/2015/04/id3-classification-using-dat

IsPure <- function(data) {
  length(unique(data[,ncol(data)])) == 1
}
Entropy <- function( vls ) {
  res <- vls/sum(vls) * log2(vls/sum(vls))
  res[vls == 0] <- 0
  -sum(res) }


InformationGain <- function( tble ) {
  tble <- as.data.frame.matrix(tble)
  entropyBefore <- Entropy(colSums(tble))
```

Code Source: https://www.r-bloggers.com/2015/04/id3-classification-using-data-tree/

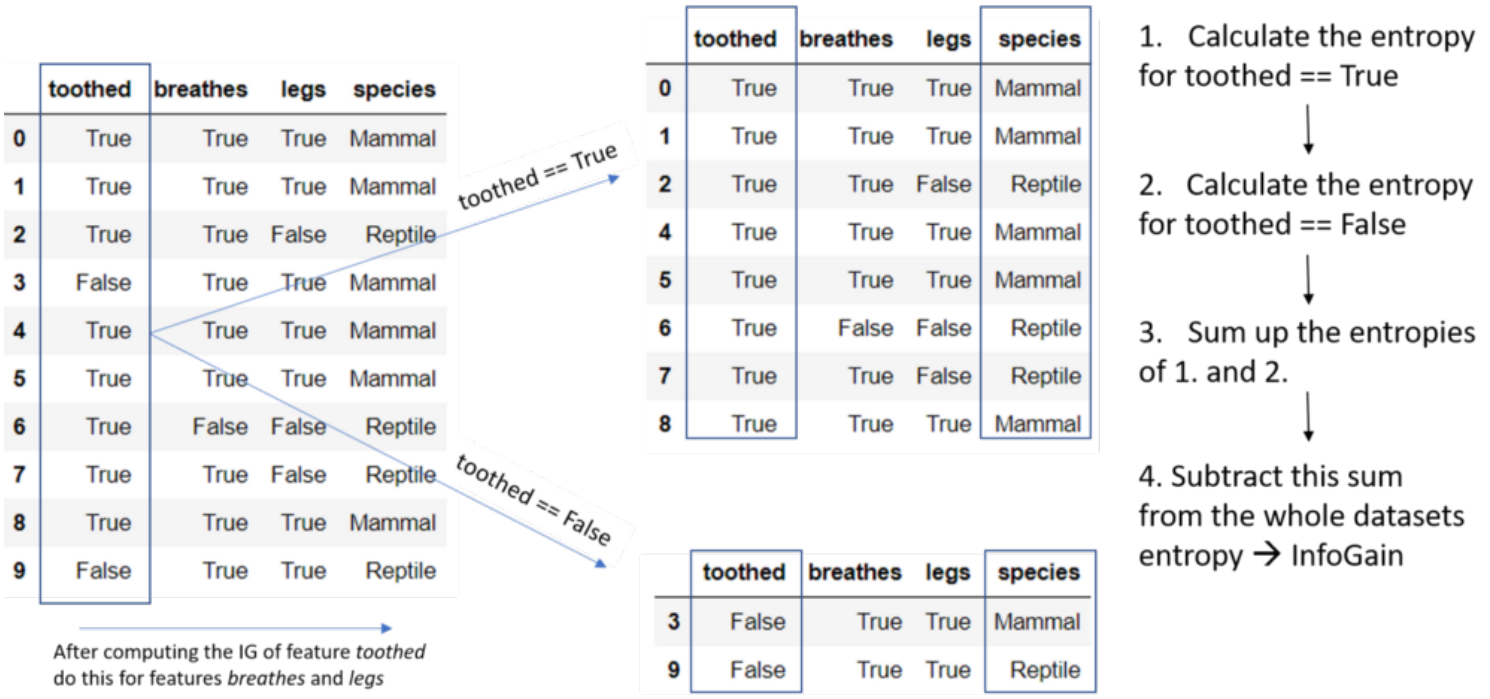
Another example: classification of species

	toothed	hair	breathes	legs	species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
2	True	False	True	False	Reptile
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
6	True	False	False	False	Reptile
7	True	False	True	False	Reptile
8	True	True	True	True	Mammal
9	False	False	True	True	Reptile

Figure Source. Machine Learning Basics: Decision Tree From Scratch: https://towardsdatascience.com/machine-learning-basics-descision-tree-from-scratch-part-ii-dee664d46831

	toothed	hair	breathes	legs	species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
8	True	True	True	True	Mammal

	toothed	hair	breathes	legs	species
2	True	False	True	False	Reptile
6	True	False	False	False	Reptile
7	True	False	True	False	Reptile
9	False	False	True	True	Reptile



	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
3	False	True	True	Mammal
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal
9	False	True	True	Reptile

	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal

	toothed	breathes	legs	species
3	False	True	True	Mammal
9	False	True	True	Reptile

Figure Source. Machine Learning Basics: Decision Tree From Scratch: https://towardsdatascience.com/machine-learning-basics-descision-tree-from-scratch-part-ii-dee664d46831

Hence the entropy of our dataset regarding the target feature is calculated with:

Entropy of parent node $= -(6/10) * \log2(6/10) - (4/10) * \log2(4/10) =$ **0.971**

Entropy of (toothed == True) = -(5/8) log2 (5/8) — (3/8) log2 (3/8) = 0.95 Entropy of (toothed == False) = — (1/2) log2 (1/2) — (1/2) log2 (1/2) = 1 Entropy for split on toothed = 8/10(0.95) + 2/10(1) = **0.96**

Information Gain = 0.971- 0.96 = ***0.011***

We'll denote entropy by "H ( )"

breathes:

$H(\text{breathes}) = (9/10 * -((6/9 * \log2(6/9)) + (3/9 * \log2(3/9))) + 1/10 * -((0) + (1 * \log2(1)))) =$ **0.82647**

InfoGain(breathes)=0.971−0.82647= **0.1445**

legs:

$H(\text{legs}) = 7/10 * -((6/7 * \log2(6/7)) + (1/7 * \log2(1/7))) + 3/10 * -((0) + (1 * \log2(1))) =$ **0.41417**

InfoGain(legs)=0.971−0.41417= **0.5568**

Hence the splitting the dataset along the feature *legs* results in the largest information gain and we should use this feature for our root node. Hence, the decision tree model looks like:

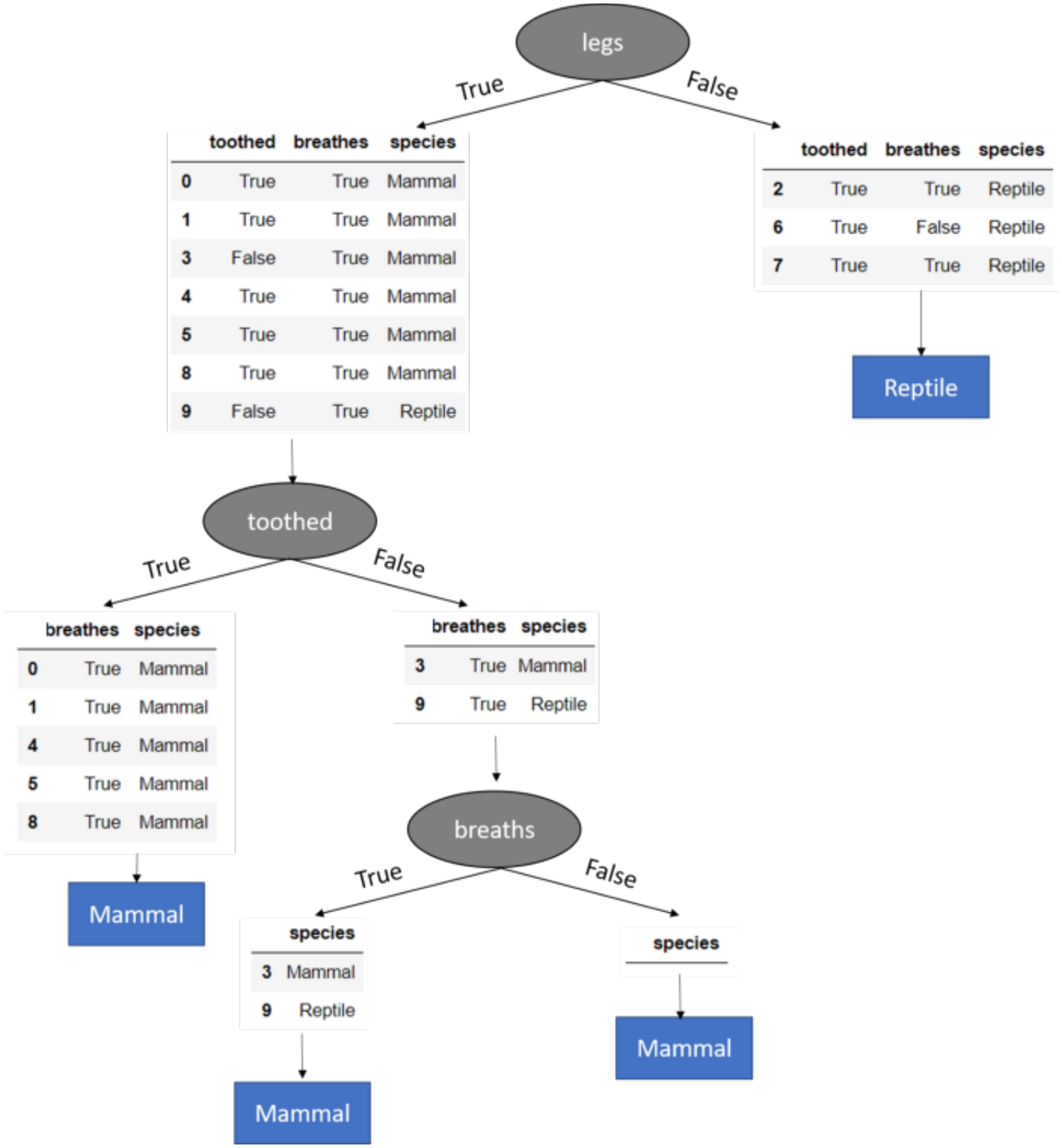


Figure Source. Machine Learning Basics: Decision Tree From Scratch: https://towardsdatascience.com/machine-learning-basics-descision-tree-from-scratch-part-ii-dee664d46831

Information gain calculation for the features toothed and breathes for the remaining dataset legs == True:

The Entropy of the (new) sub data set after the first split:

$H(D) = -((67 * \log2(67)) + (17 * \log2(17))) =$ **0.5917**

toothed:

$H(\text{toothed}) = 5/7 * -((1 * \log2(1)) + (0)) + 2/7 * -((1/2 * \log2(1/2)) + (1/2 * \log2(1/2))) =$ **0.285**

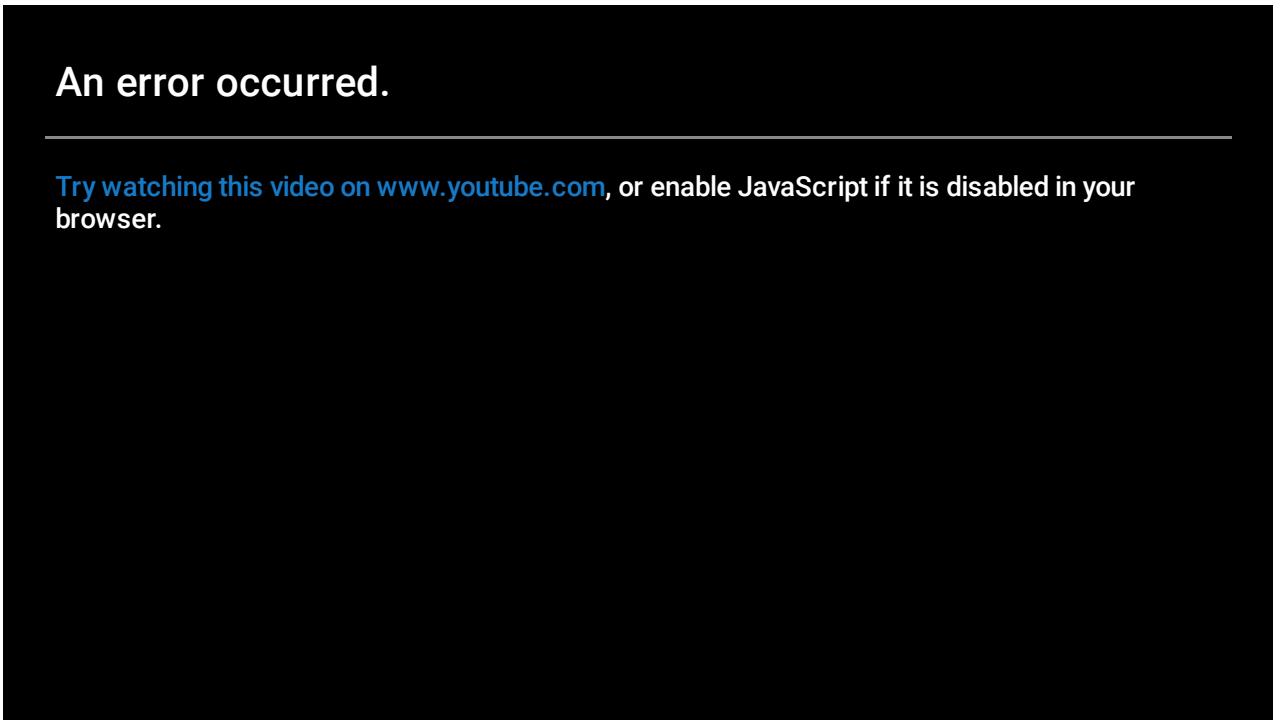
✅ InfoGain(toothed)=0.5917−0.285= **0.3067**

breathes:

$H(\text{breathes}) = 77 * -((67 * \log2(67)) + (17 * \log2(17))) + 0 =$ **5917**

✅ InfoGain(toothed)= 0.5917−0.5917 = **0**

The dataset for toothed == False still contains a mixture of different target feature values why we proceed partitioning on the last left feature (== breathes)



References

1. Decision Tree Lecture Notes by T Mitchell (Machine Learning): http://www2.cs.uh.edu/~ceick/ai/dectree.pdf https://www.cs.princeton.edu/courses/archive/spr07/cos424/papers/mitchell-dectrees.pdf
2. Loh, W. Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*(1), 14-23. https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.8

3. Another Example: https://towardsdatascience.com/machine-learning-basics-descision-tree-from-scratch-part-ii-dee664d46831

CART

Impurity measure: *Entropy* or *Gini*?

So far, we used the *entropy* impurity measure. The *classification and regression tree* (CART) decision tree algorithm given in the textbook uses the *Gini* impurity measure.

Loh, W. Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*(1), 14-23. https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.8

In general, either impurity measure will do a good job. The ***Gini* measure** is a bit faster (since there is no need to calculate log values); however, the ***entropy* measure** produces more balanced decision trees.

Training Algorithm	**CART** (Classification and Regression Trees)	**ID3** (Iterative Dichotomiser 3)
Target(s)	Classification and Regression	Classification
Metric	Gini Index	Entropy function and Infomation Gain
Cost function (Based on what to split?)	Select its splits to achieve the subsets that minimize Gini Impurity	Yield the largest Information Gain for categorical targets

Figure Source. Decision Tree Intuition: From Concept to Application: https://www.kdnuggets.com/2020/02/decision-tree-intuition.html

The CART Decision Tree algorithm will choose the attribute with the largest Gini Gain as the Root Node. A branch with Gini of 0 is a leaf node, while a branch with Gini more than 0 needs further splitting. Similar to ID3, the nodes are grown recursively until all data is classified.

The Gini measure is given by

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2$$

where p_i is the probability of an object being classified to a particular class.

		Yes	No	Total
Feature 2: Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	3	2	5
	Total	10	4	

Gini (PlayTennis, Outlook=Sunny)
= 1-(⅖)^2 - (⅗)^2 =0.48

Gini (PlayTennis, Outlook=Overcast)
= 1-(4/4)^2 - (0/4)^2=0

Gini (PlayTennis, Outlook=Rainy)
= 1-(⅗)^2 - (⅖)^2 =0.48

The Gigi Index of Outlook (children node)
= 5/14 x 0.48 + 4/14 x 0 + 5/14 x 0.48 = 0.3429

Gini Gain = Gini (parent node) - Gini (children node)

= [1- (10/14)^2 -(4/14)^2] - 0.3429
= 0.4082 - 0.3429
= 0.065

Figure Source. Decision Tree Intuition: From Concept to Application:
https://www.kdnuggets.com/2020/02/decision-tree-intuition.html

Below we provide further details for each attribute in the dataset.

```
Gini(Outlook=Sunny) = 1 – (2/5)^2 – (3/5)^2 = 1 – 0.16 – 0.36 = 0.48
Gini(Outlook=Overcast) = 1 – (4/4)^2 – (0/4)^2 = 0
Gini(Outlook=Rain) = 1 – (3/5)^2 – (2/5)^2 = 1 – 0.36 – 0.16 = 0.48

Gini(Outlook) = (5/14) x 0.48 + (4/14) x 0 + (5/14) x 0.48 = 0.171 + 0 + 0.171 = 0.342
```

```
Gini(Temp=Hot) = 1 – (2/4)^2 – (2/4)^2 = 0.5
Gini(Temp=Cool) = 1 – (3/4)^2 – (1/4)^2 = 1 – 0.5625 – 0.0625 = 0.375
Gini(Temp=Mild) = 1 – (4/6)^2 – (2/6)^2 = 1 – 0.444 – 0.111 = 0.445

Gini(Temp) = (4/14) x 0.5 + (4/14) x 0.375 + (6/14) x 0.445 = 0.142 + 0.107 + 0.190 =
0.439
```

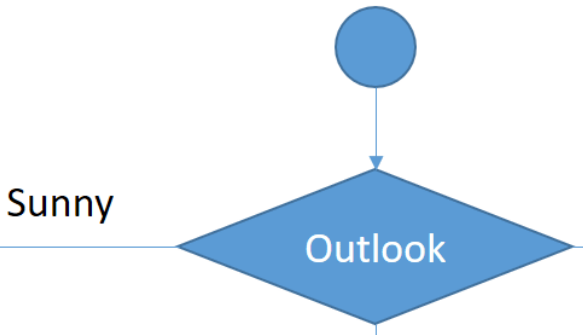
```
Gini(Humidity=High) = 1 - (3/7)^2 - (4/7)^2 = 1 - 0.183 - 0.326 = 0.489
Gini(Humidity=Normal) = 1 - (6/7)^2 - (1/7)^2 = 1 - 0.734 - 0.02 = 0.244

Gini(Humidity) = (7/14) x 0.489 + (7/14) x 0.244 = 0.367
```

```
Gini(Wind=Weak) = 1 - (6/8)^2 - (2/8)^2 = 1 - 0.5625 - 0.062 = 0.375
Gini(Wind=Strong) = 1 - (3/6)^2 - (3/6)^2 = 1 - 0.25 - 0.25 = 0.5

Gini(Wind) = (8/14) x 0.375 + (6/14) x 0.5 = 0.428
```

Finally, we need to make a decision by selecting the lowest Gini index node, which is Outlook in our case.

```
Feature     Gini index
---------------------
Outlook         0.342
Temperature     0.439
Humidity        0.367
Wind            0.428
```

Figure Source. A Step by Step CART Decision Tree Example: https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/



Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Figure Source. A Step by Step CART Decision Tree Example: https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/

Next, we look at Sunny leaf taking into account Outlook is the root node:

```
Gini(Outlook=Sunny and Temp.=Hot) = 1 - (0/2)^2 - (2/2)^2 = 0
Gini(Outlook=Sunny and Temp.=Cool) = 1 - (1/1)^2 - (0/1)^2 = 0
Gini(Outlook=Sunny and Temp.=Mild) = 1 - (1/2)^2 - (1/2)^2 = 1 - 0.25 - 0.25 = 0.5

Gini(Outlook=Sunny and Temp.) = (2/5)x0 + (1/5)x0 + (2/5)x0.5 = 0.2
```

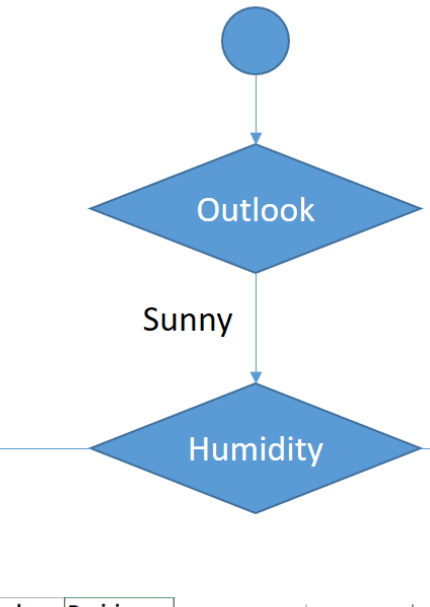
```
Gini(Outlook=Sunny and Humidity=High) = 1 - (0/3)^2 - (3/3)^2 = 0
Gini(Outlook=Sunny and Humidity=Normal) = 1 - (2/2)^2 - (0/2)^2 = 0

Gini(Outlook=Sunny and Humidity) = (3/5)x0 + (2/5)x0 = 0
```

```
Gini(Outlook=Sunny and Wind=Weak) = 1 - (1/3)^2 - (2/3)^2 = 0.266
Gini(Outlook=Sunny and Wind=Strong) = 1- (1/2)^2 - (1/2)^2 = 0.2

Gini(Outlook=Sunny and Wind) = (3/5)x0.266 + (2/5)x0.2 = 0.466
```

Then we make a decision where Humidity is chosen:

```
Feature    Gini index
--------------------
```

```
Temperature    0.2
Humidity         0
Wind         0.466
```



Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No

Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Figure Source. A Step by Step CART Decision Tree Example: https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/

Since the leaves are pure, then the labels "no" and "yes" will be placed on right and left, respectively.

Next, we need to recurse back and return to Outlook where we look at the next leaf, which is Overcast. We know it is pure, and hence the label "yes" is placed.

Can you figure out the rest?

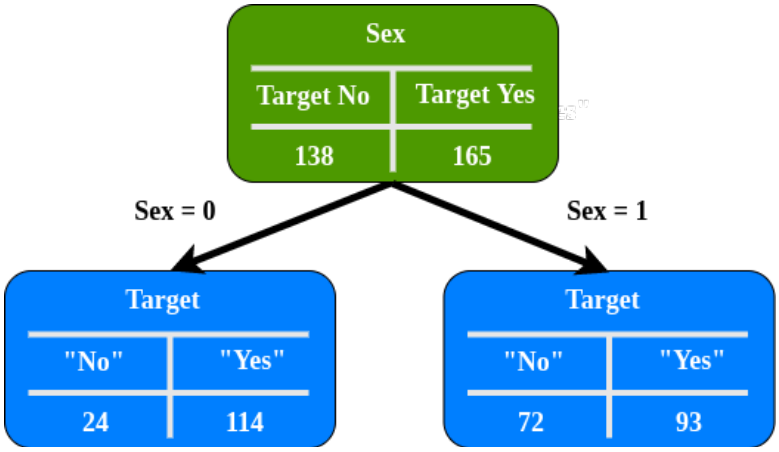
In this case, as well, the attribute Temperature will not become a node.

Another Example

We consider using Heart Disease Data set with 303 rows and has 13 attributes where the Target consists of 138 negative values (0) and 165 positive values (1)

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	Yes
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	Yes
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	Yes
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	Yes
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	Yes

Figure Source. The Basics of Decision Trees: https://medium.com/datadriveninvestor/the-basics-of-decision-trees-e5837cc2aba7



Gini Impurity — Left Node

$$I_{Left-Sex} = 1 - \left(\frac{24}{24+114}\right)^2 - \left(\frac{114}{24+114}\right)^2$$

$$I_{Left-Sex} = 0.29$$

Gini Impurity — Right Node

$$I_{Right-Sex} = 1 - \left(\frac{72}{72+93}\right)^2 - \left(\frac{93}{72+93}\right)^2$$

$$I_{Right-Sex} = 0.49$$

Figure Source. The Basics of Decision Trees: https://medium.com/datadriveninvestor/the-basics-of-decision-trees-e5837cc2aba7

Next, we can calculate the total Gini Impurity with weight average. The Left Node represents 138 patients while Right Node represents 165 patients.

Total Gini Impurity — Leaf Node

$$I_{Sex} = \textit{weight average of the leaf node impurities}$$

$$I_{Sex} = \left(\frac{138}{138+165}\right) I_{Left-Sex} + \left(\frac{165}{138+165}\right) I_{Right-Sex}$$

$$I_{Sex} = \left(\frac{138}{138+165}\right) 0.29 + \left(\frac{165}{138+165}\right) 0.49$$

$$I_{Sex} = 0.399$$

Figure Source. The Basics of Decision Trees: https://medium.com/datadriveninvestor/the-basics-of-decision-trees-e5837cc2aba7

Further example using same dataset.

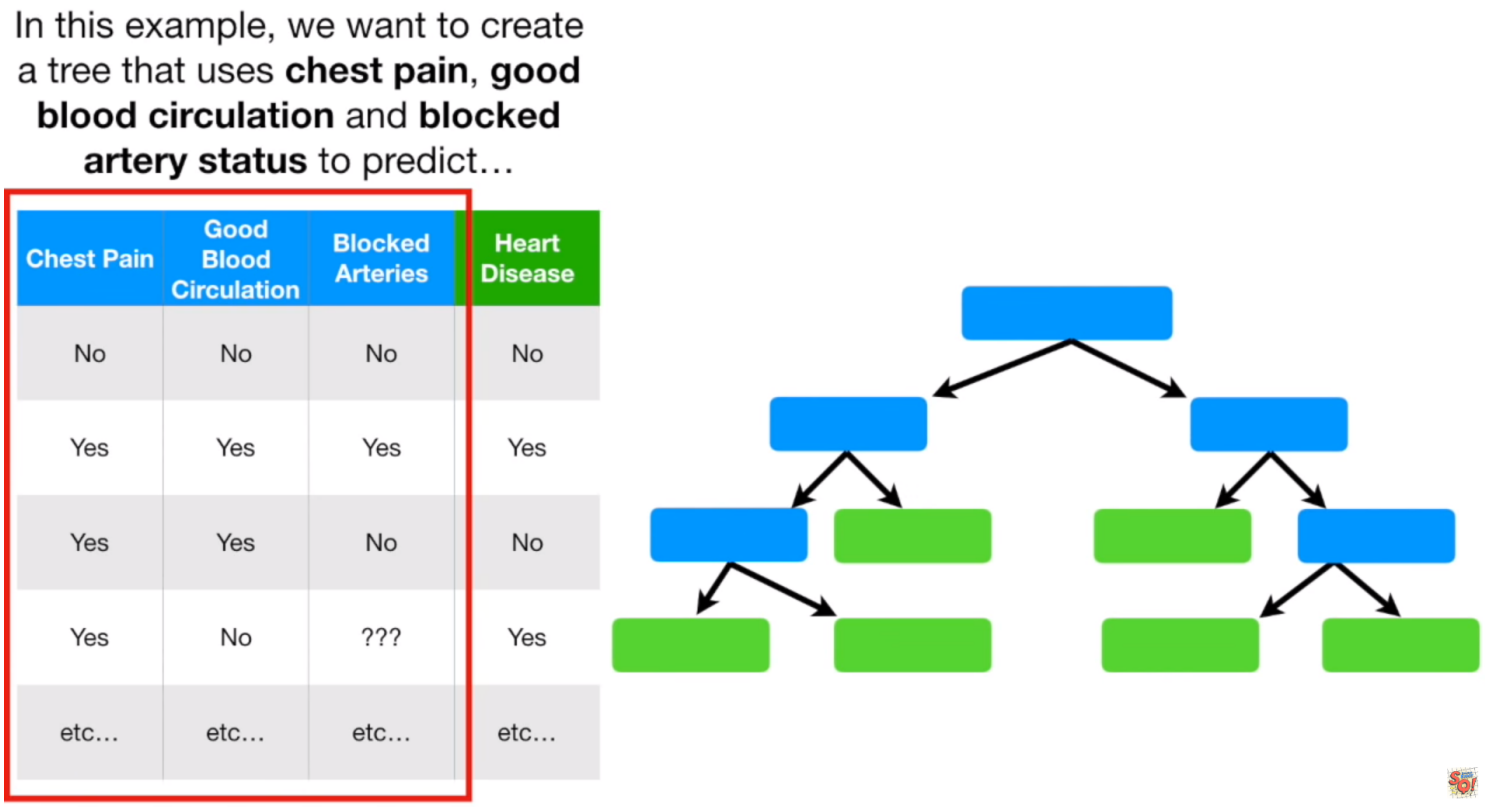
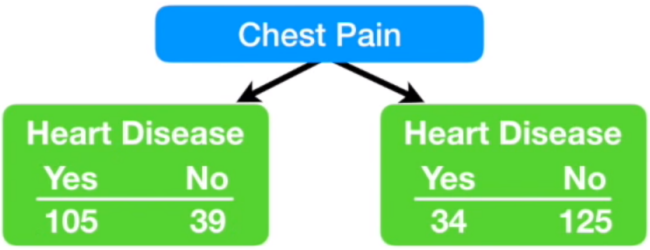


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

We first calculate the Gini impurity of the left leaf.



For this leaf, the Gini impurity = 1 - (the probability of "yes")^2 - (the probability of "no")^2

$$= 1 - \left(\frac{105}{105 + 39}\right)^2 - \left(\frac{39}{105 + 39}\right)^2$$

$$= 0.395$$

Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

Then in a similar way, the Gini impurity for the right leaf is calculated and then the Gini impurity for the node Chest Pain is calculated as shown below.

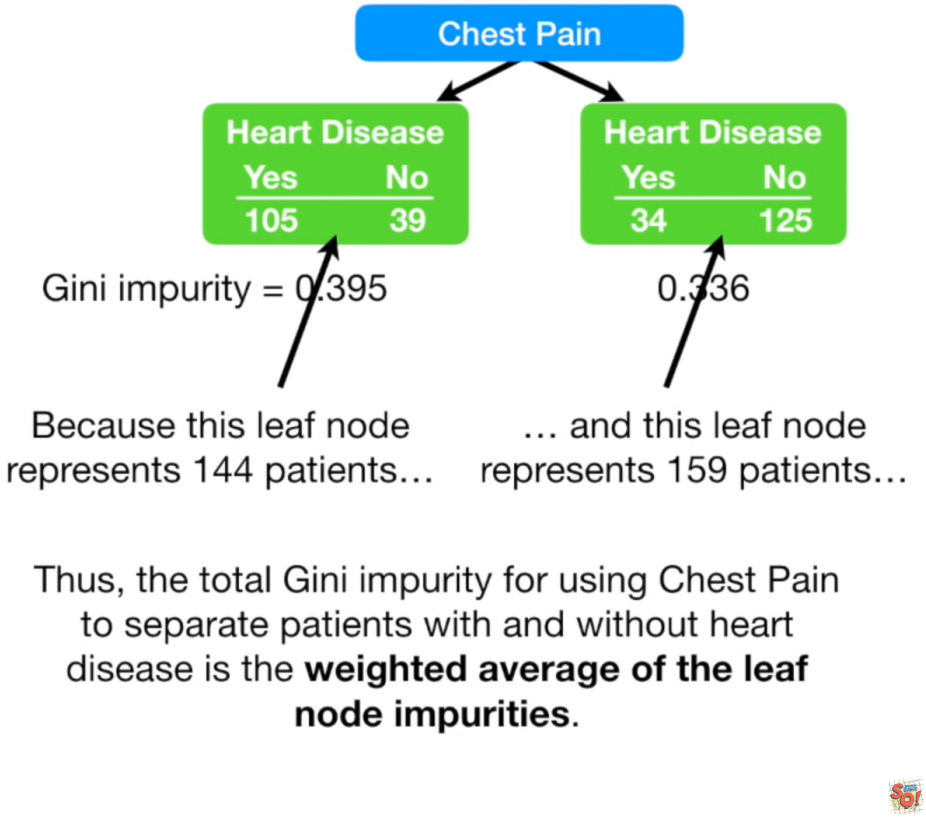
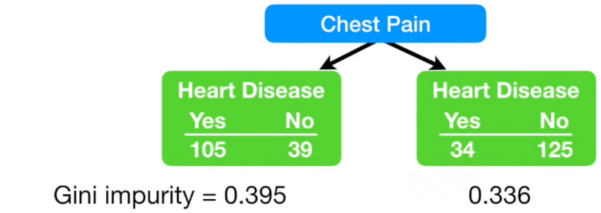


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

Let us calculate the Gini impurity for Chest Pain:



Gini impurity for Chest Pain = weighted average of Gini impurities for the leaf nodes

$$= \left(\frac{144}{144 + 159}\right) 0.395 + \left(\frac{159}{144 + 159}\right) 0.336$$

$$= 0.364$$

Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

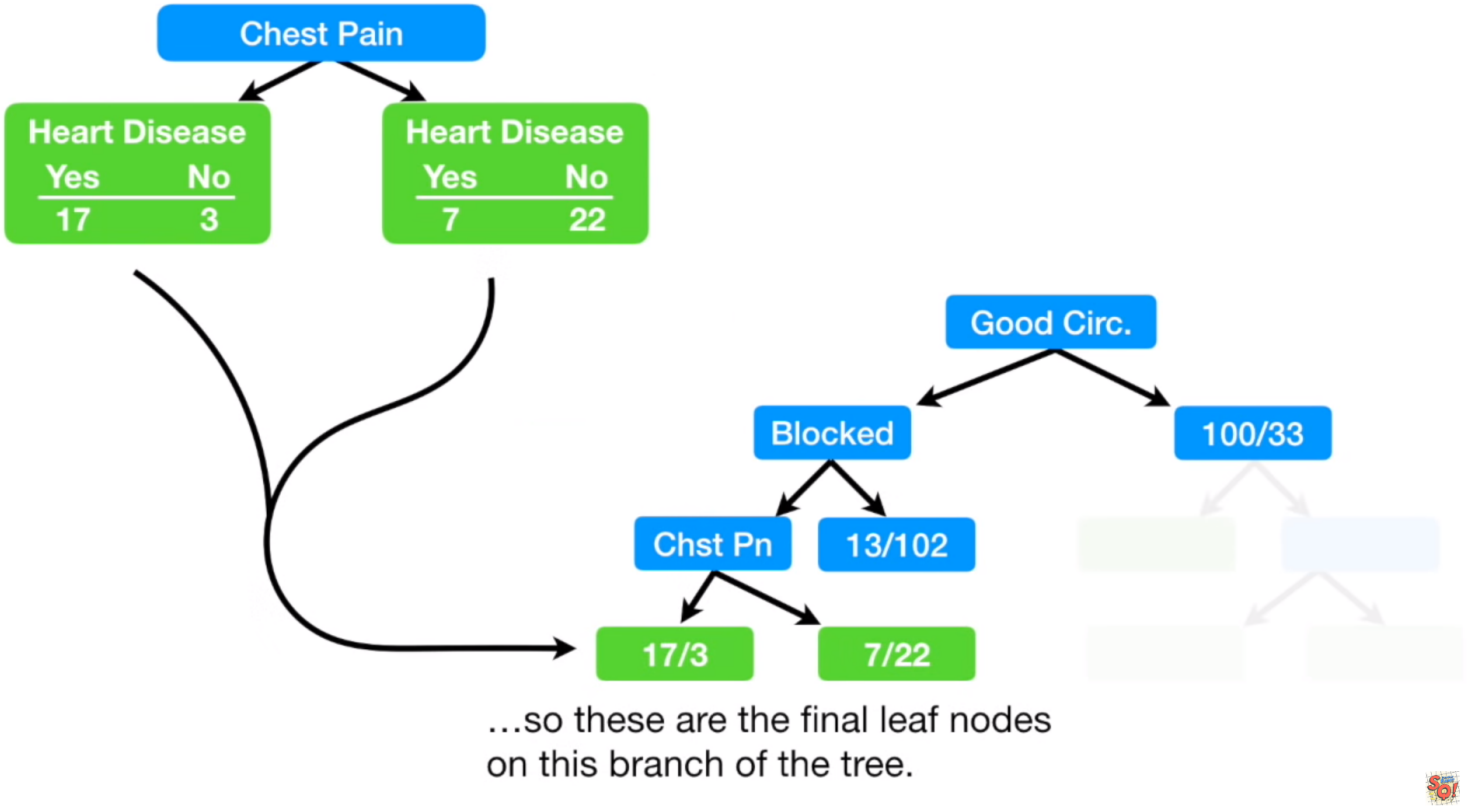


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

The good news is that we follow the exact same steps as we did on the left side:

1) Calculate all of the Gini impurity scores.

2) If the node itself has the lowest score, than there is no point in separating the patients any more and it becomes a leaf node.

3) If separating the data results in an improvement, than pick the separation with the lowest impurity value.

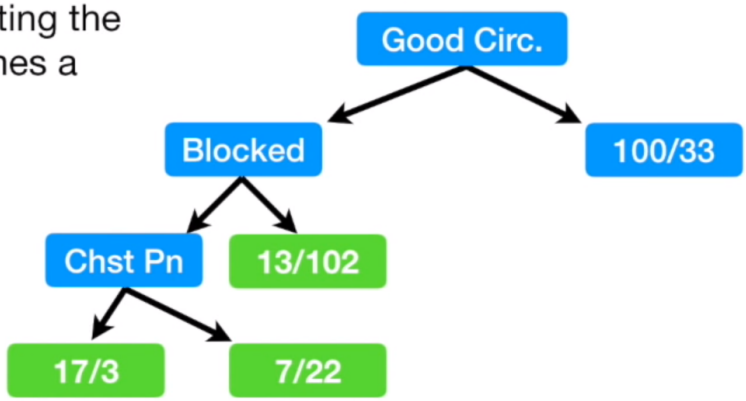


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

Next, we compare Gini with Information Gain for Iris problem.

Run

```r
#Source: http://www.learnbymarketing.com/481/decision-tree-flavors-gini-

gini_process <-function(classes,splitvar = NULL){
  #Assumes Splitvar is a logical vector
  if (is.null(splitvar)){
    base_prob <-table(classes)/length(classes)
    return(1-sum(base_prob**2))
  }
  base_prob <-table(splitvar)/length(splitvar)
  crosstab <- table(classes,splitvar)
  crossprob <- prop.table(crosstab,2)
  No_Node_Gini <- 1-sum(crossprob[,1]**2)
  Yes_Node_Gini <- 1-sum(crossprob[,2]**2)
  return(sum(base_prob * c(No_Node_Gini,Yes_Node_Gini)))
```

Code Source: http://www.learnbymarketing.com/481/decision-tree-flavors-gini-info-gain/

CART can also handle the regression problem using a different splitting criterion such as Mean Squared Error (MSE) to determine the splitting points. The output variable of a Regression Tree is numerical, and the input variables allow a mixture of continuous and categorical variables. We will cover details in next lesson.

CART implementation in Python

Now we see CART Tree implementation from scratch in python that works both with Gini and Entropy.

Note data below

```
Title: Balance Scale Weight & Distance Database

Number of Instances: 625 (49 balanced, 288 left, 288 right)
Number of Attributes: 4 (numeric) + class name = 5

Attribute Information:
Class Name (Target variable): 3
L [balance scale tip to the left]
B [balance scale be balanced]
R [balance scale tip to the right]
Left-Weight: 5 (1, 2, 3, 4, 5)
Left-Distance: 5 (1, 2, 3, 4, 5)
Right-Weight: 5 (1, 2, 3, 4, 5)
Right-Distance: 5 (1, 2, 3, 4, 5)

Missing Attribute Values: None
Class Distribution:
46.08 percent are L
07.84 percent are B
46.08 percent are R
```

Source: https://archive.ics.uci.edu/ml/datasets/Balance+Scale

and code:

Run PYTHON

```
#Source: https://www.geeksforgeeks.org/decision-tree-implementation-pyt
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
    balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
```

Code Source: https://www.geeksforgeeks.org/decision-tree-implementation-python/

Decision Tree Algorithms

There are several algorithms for creating decision trees and we discuss some of the prominent ones.

- **ID3** Decision Tree Algorithm creates a multiway tree where each node is developed in a greedy manner to yield the largest information gain for categorical targets. The trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data. (Developed in 1986 by Ross Quinlan)

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106
https://link.springer.com/content/pdf/10.1007/BF00116251.pdf

- **C4.5** Decision Tree Algorithm is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. The accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it. (Developed in 1993 by Ross Quinlan)

Quinlan, J. R. (1993). C4. 5: Programs for Machine Learning.
Morgan Kaufmann Publishers Inc.: https://dl.acm.org/doi/book/10.5555/583200

- **C5.0** is a version that improves C4.5 by using less memory and builds smaller rulesets than C4.5 while being more accurate. (Developed by Ross Quinlan)

Quinlan, J. R. (1996). Improved use of continuous attributes in C4. 5. *Journal of artificial intelligence research*, *4*, 77-90: http://home.engineering.iastate.edu/~julied/classes/ee547/Handouts/quinlan96a.pdf

- **CART (Classification and Regression trees)** is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984). Classification and Regression Trees. Belmont: Wadsworth (hard to get the real source)

Steinberg, D., & Colla, P. (2009). CART: classification and regression trees. *The top ten algorithms in data mining*, *9*, 179. https://www.researchgate.net/profile/Dan_Steinberg2/publication/265031802_Chapter_10_CART_Classification_and_Regression_Trees/links/567dcf8408ae051f9ae493fe.pdf

Learning with a continuous value attribute

So far we have used discrete value attributes only for inducing a decision tree. However, if we have a continuous value attribute, like say *Temperature* as shown below, we first need to create a corresponding discrete value attribute and use that discrete value attribute in building a decision tree. One popular approach is to sort training examples on continuous attribute value and find possible split points (midway boundaries) where class value changes (Fayyad, 1991).

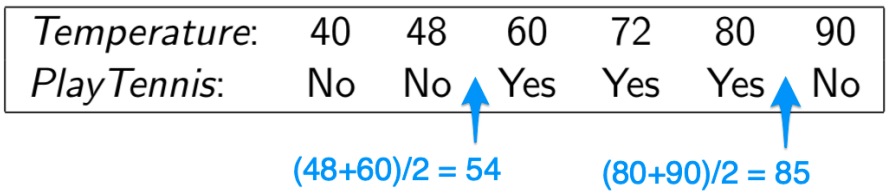


Figure: Values for Temperature and Play tennis. Adapted from *Machine Learning* by T. Mitchell, 1997

In the above example, the Play Tennis outcome (Yes / No) changes between [48 and 60] and [80 and 90]. Therefore, two split points (48+60)/2 = 54 and (80+90)/2 = 85 will enable two possible boolean splits (Temperature > 54) and (Temperature > 85). The possible splits can be evaluated for information gain, and the best split could be selected. Hence, dealing with a continuous attribute is computationally more expensive; however, normally proven methods (from dynamic programming) are used to quickly find a good split point.

More information:

Chickering, David Maxwell, Christopher Meek, and Robert Rounthwaite. "Efficient determination of dynamic split points in a decision tree." *Proceedings 2001 IEEE international conference on data mining*. IEEE, 2001: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.5468&rep=rep1&type=pdf

Let's look at the heart disease dataset again and consider continuous values.

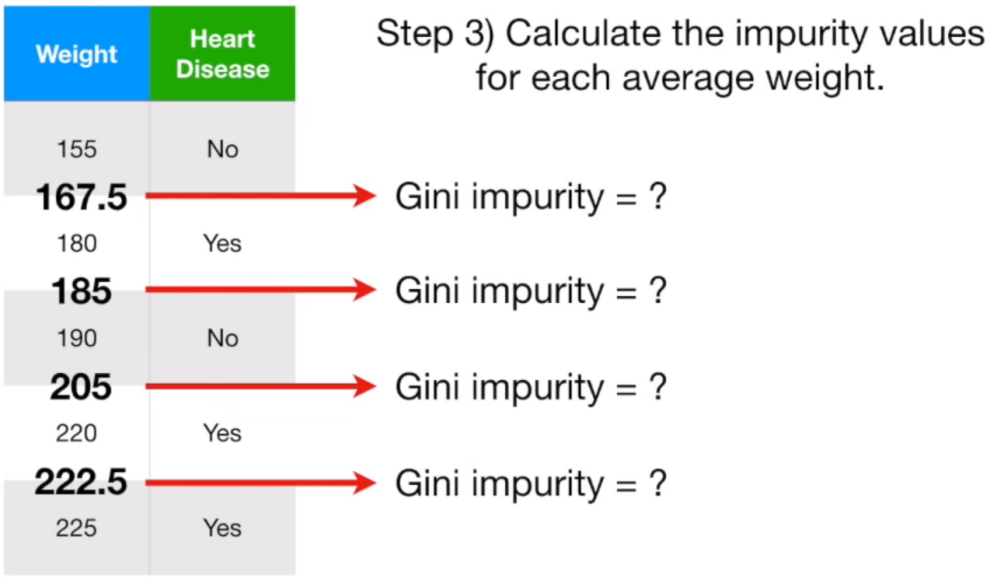


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

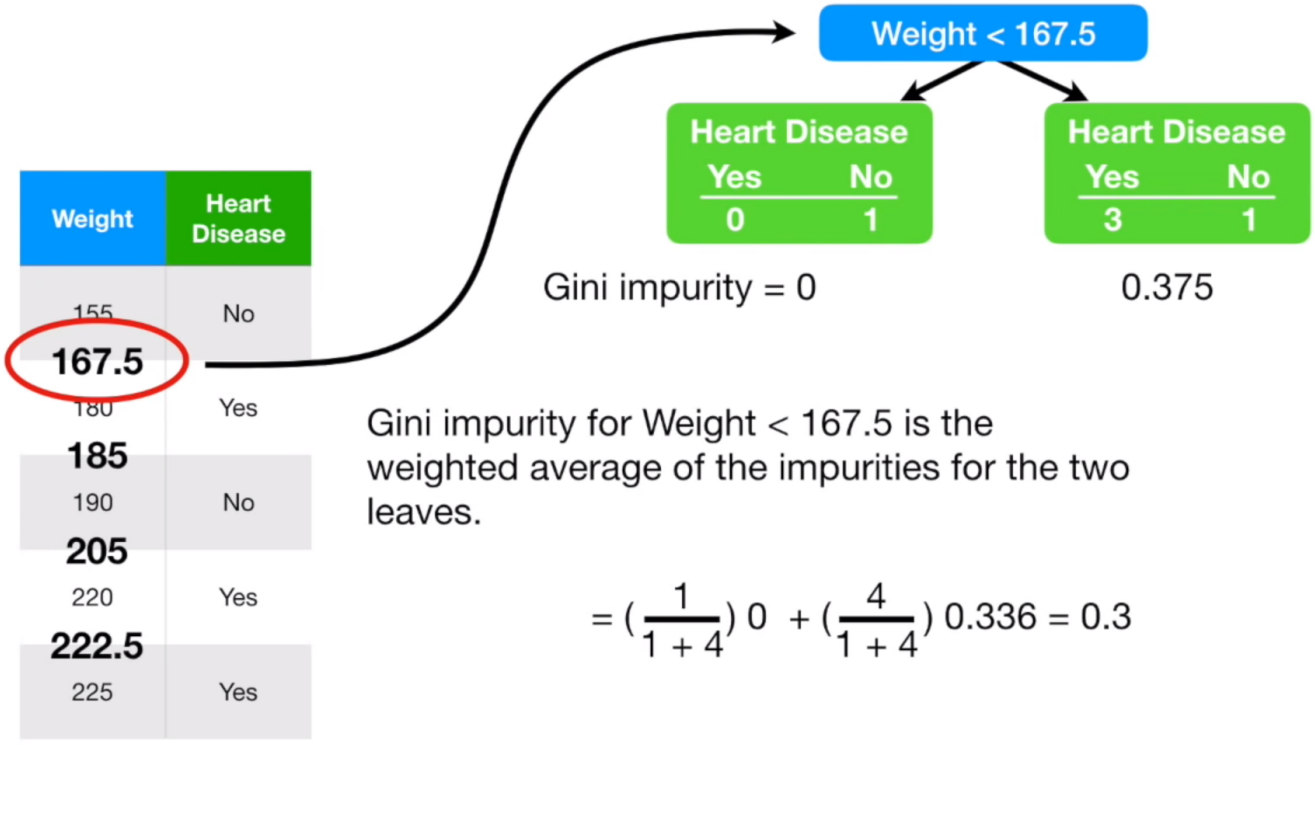


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

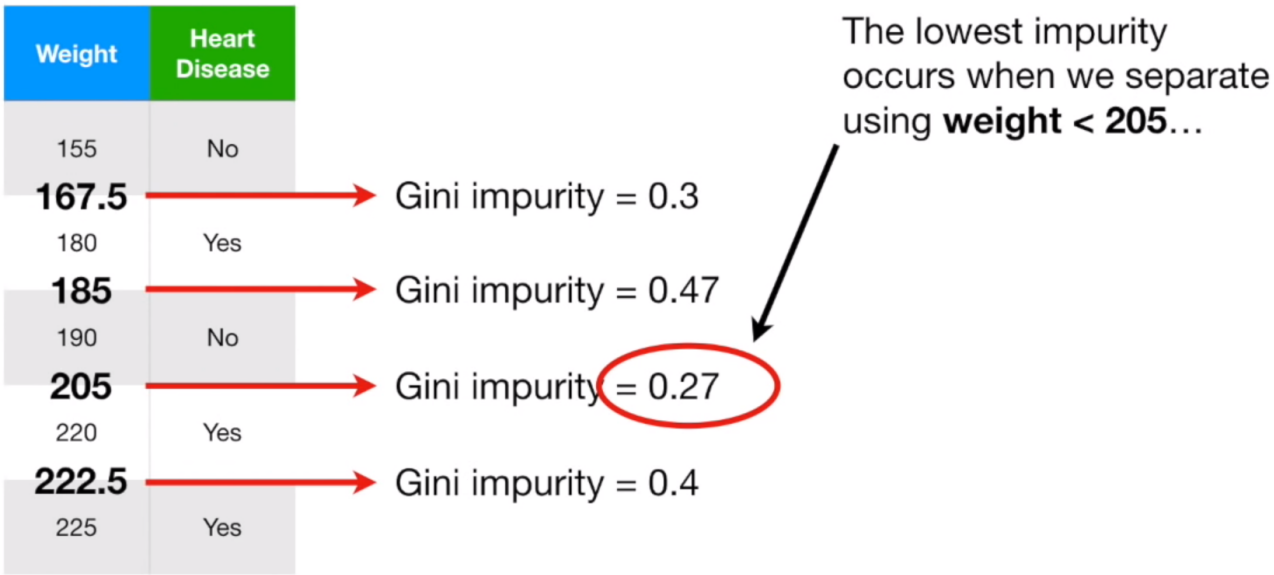


Figure Source. Decision Trees by StatsQuest: https://www.youtube.com/watch?v=7VeUPuFGJHk&feature=emb_logo

Handling missing values in training data

Often training or a test data set may have some attributes with missing values. This may be due to the unavailability of data or they might be very expensive to acquire. Normally one of the following three approaches is used:

- if node n tests A, assign most common value of A among other examples sorted to node n,
- assign most common value of A among other examples with same target value,
- assign probability p_i to each possible value v_i of A, and assign fraction p_i of examples to each branch (representing each v_i) in tree.

We also need to use the same approach to classify a test (an unseen) case.

Attributes with differing costs

All attributes are not the same! Some may cost significantly more than the others. For example, a blood test may cost $150 as opposed to an attribute representing the height of the patient. We normally like to avoid expensive attributes from our decision-making process, as much as possible. Often the following revised information gain measure (Gain from ID3) is used to account for attribute cost:

$$\frac{Gain^2\left(S, A\right)}{Cost\left(A\right)}$$

Tan, M. (1993). Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, *13*(1), 7-33: https://link.springer.com/content/pdf/10.1007/BF00993101.pdf

Another approach is to associate a *cost matrix* with false positive and false negative classifications. For example, the cost of falsely classifying *no-cancer* may result in death. However, falsely classifying *cancer* may result in further tests costing a few dollars, but no deaths. Such a cost matrix forces a different tree structure to be learned to minimise *asymmetric* misclassification costs.

Advantages/Limitations

Black-box models are those that are not explainable especially how they come to a decision, such as neural networks and related methods ensemble methods such as Random Forests which we will cover this week. In Black-box models, is not easy to get insights into which attributes were used in a decision-making process. The major advantages of Decision Trees generally address problems with black-box methods:

1. A major advantage of decision tree easy to interpret and where when required, it is easy to generate human-comprehensible classification rules.

2. The decision-making process is visible and often comprehensible, therefore such models are called white-box models.
3. Decision trees can be displayed graphically, and are easily interpreted even by a non-expert, especially if they are small.
4. Decision trees are seen as more closely mirror human decision-making than typical regression and classification approaches.
5. Trees can easily handle qualitative predictors without the need to create dummy variables.

The major limitations or Decision trees are:

1. Black-box models generally provide better accuracy than decision trees and hence, the question is if accuracy is needed or information about the decision making process.
2. There are challenges when it comes to robustness of decision trees. A small change in training data can at times change the major structure of the tree.
3. Decision-tree learners often create over-complex trees that do not generalize well from the training data.
4. Challenges in big data problems where a large number of attributes are present.
5. For data including categorical variables with different numbers of levels, information gain in decision trees is biased in favour of attributes with more levels.

Deng, H., Runger, G., & Tuv, E. (2011, June). Bias of importance measures for multi-valued attributes and solutions. In *International conference on artificial neural networks* (pp. 293-300). Springer, Berlin, Heidelberg. https://www.researchgate.net/publication/221079908_Bias_of_Importance_Measures_for_Multi-valued_Attributes_and_Solutions

Scitkit-Learn Example

Next, we use Scikit-learn that implements CART for Decision Tree for Iris Dataset.

Run PYTHON

```
#Source: https://heartbeat.fritz.ai/understanding-the-mathematics-behind

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree.export import export_text
iris = load_iris()
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
decision_tree = decision_tree.fit(iris.data, iris.target)
r = export_text(decision_tree, feature_names=iris['feature_names'])
print(r)
```

Run PYTHON

```
#Source: https://www.hackerearth.com/practice/machine-learning/machine-l

#Importing required libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

#Loading the iris data
data = load_iris()
print('Classes to predict: ', data.target_names)
#There are three classes of iris plants: 'setosa', 'versicolor' and 'vir
#Extracting data attributes
```

I know it is important to visualise your tree and hence the code below visualise the notes and leaves.

Run PYTHON

```python
#source: https://www.kaggle.com/willkoehrsen/visualize-a-decision-tree-w

from sklearn.datasets import load_iris
iris = load_iris()

# Model (can also use single decision tree)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=10)

# Train
model.fit(iris.data, iris.target)
# Extract single tree
estimator = model.estimators_[5]

```

We now look at R example using Caret:

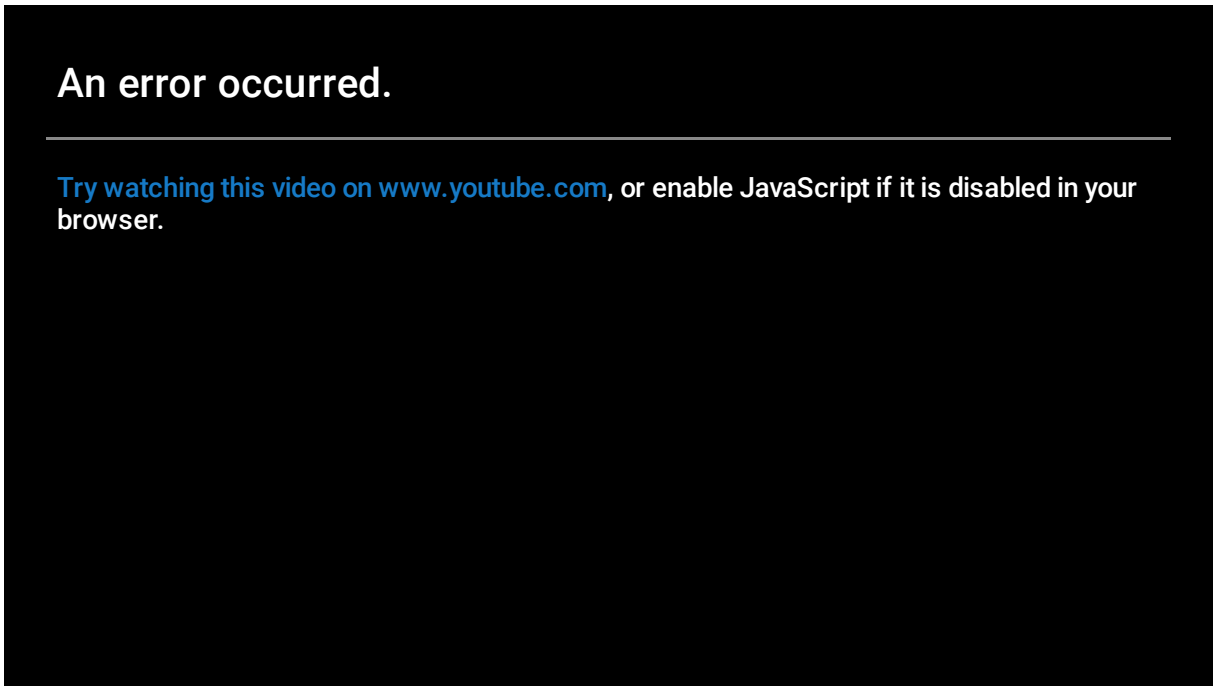
http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/141-cart-model-decision-tree-essentials/

Run R

```r

#Source: https://rpubs.com/maulikpatel/229337

data("iris")
names(iris) = tolower(names(iris))
table(iris$species)

suppressMessages(library(caret))
index = createDataPartition(y=iris$species, p=0.7, list=FALSE)

train.set = iris[index,]
test.set = iris[-index,]

dim(train.set)
```

Further information for R and Caret: http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/141-cart-model-decision-tree-essentials/

Below is a video showing details of CART with another example.



References

1. Loh, W. Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*(1), 14-23. https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.8
2. Chickering, David Maxwell, Christopher Meek, and Robert Rounthwaite. "Efficient determination of dynamic split points in a decision tree." *Proceedings 2001 IEEE international conference on data mining*. IEEE, 2001: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.5468&rep=rep1&type=pdf
3. Scikit-learn Decision Tree: https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart
4. Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery*, *2*(4), 345-389:https://cursa.ihmc.us/rid=1MYWPSJ6G-2CTVWGP-30RY/Murthy%201998%20DMKD%20Automatic%20Construction%20of%20Decision%20Trees.pdf
5. Rudin, C., & Radin, J. (2019). Why are we using black box models in AI when we don't need to? A lesson from an explainable AI competition. *Harvard Data Science Review*, *1*(2): https://hdsr.mitpress.mit.edu/pub/f9kuryi8/release/5?source=techstories.org

Regression Trees

Now that you have mastered decision trees, we can move to regression trees. **Regression trees** *partition* a data set into subgroups and fit a simple *constant* for each instance in the subgroup. The partitioning is achieved by successive binary partitions (*recursive partitioning*) based on the different predictors. The constant to predict is based on the average response values for all observations that fall in the particular subgroup.

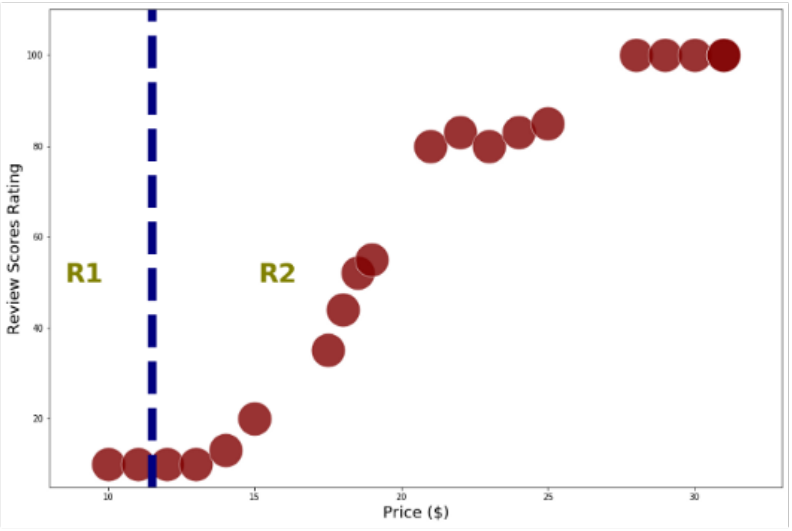
Hence, a bit of change in decision trees from the previous lesson can enable them to perform regression tasks. The overall algorithm for building a regression tree is similar to building a classification decision tree certain important differences.

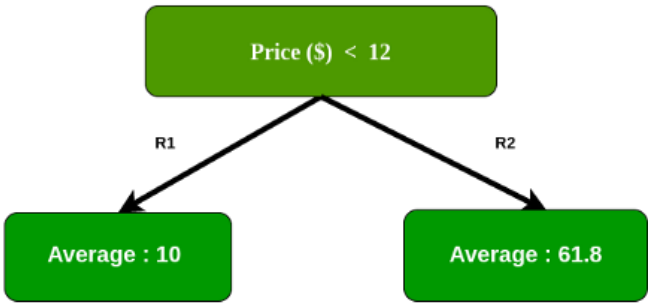
1. each leaf node predicts a continuous value, not a discrete class value
2. a prediction is based on the training cases associated with the corresponding leaf node (i.e. an average value of the target attribute)
3. a data split tries to minimise the mean-squared-error (MSE) resulting in a leaf as close as possible to the predicted value.

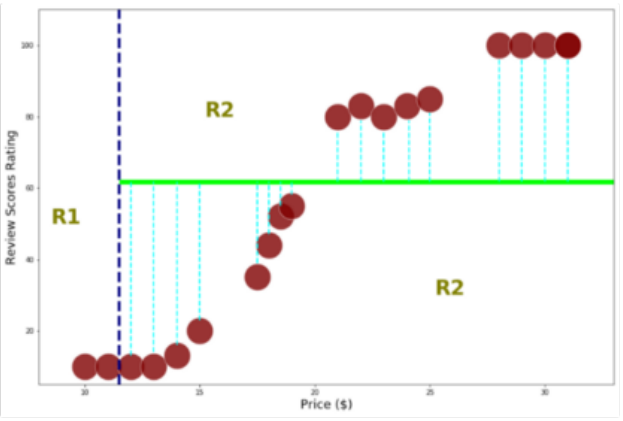
The CART regression tree algorithm begins with the entire data set S and searches every distinct value of every input variable to find the predictor and split value that partitions the data into two regions (R_1 and R_2) such that the overall SSE is minimized for average values (c_1 and c_2) in the respective regions as shown below.

$$\texttt{minimize}\left\{ SSE = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2 \right\}$$

Hence, CART in regression cases uses least-squares which intuitively splits are chosen to minimize the residual sum of squares (RSS) between the observation and the mean in each node as shown below.







$$\varepsilon = \sum_{i=1}^{n} (actual\ value - average\ value\ in\ each\ region)^2$$

$$RSS = \varepsilon_1^2 + \varepsilon_2^2 + .. + \varepsilon_n^2$$

$$RSS = (10 - 10)^2 + (10 - 10)^2 + (10 - 58.9)^2 \ldots (10 - 58.9)^2 = 18609.08$$

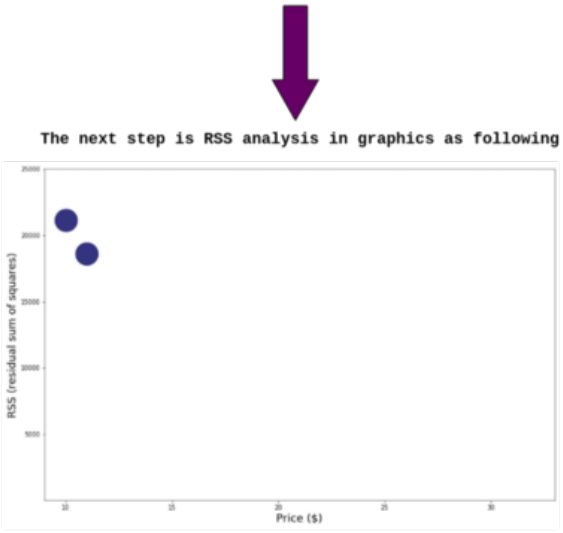


Figure Source. Regression in Decision Tree: https://medium.com/@arifromadhan19/regrssion-in-decision-tree-a-step-by-step-cart-classification-and-regression-tree-196c6ac9711e

After finding the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. This process is continued until some stopping criterion is reached. What results is, typically, a very deep, complex tree that may produce good predictions on the training set, but is likely to overfit the data, leading to poor performance on unseen data.

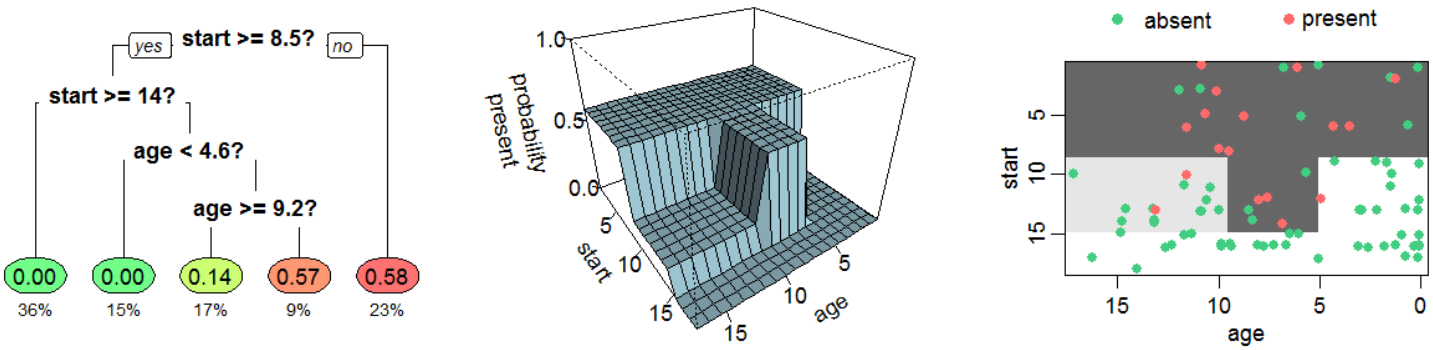


Figure Source. Regression Tree that estimates the probability of kyphosis after surgery, given the age of the patient and the vertebra at which surgery was started. The same tree is shown in three different ways. **Left** The coloured leaves show the probability of kyphosis after surgery and the percentage of patients in the leaf. **Middle** The tree as a perspective plot. **Right** Aerial view of the middle plot. The probability of kyphosis after surgery is higher in darker areas.": https://en.wikipedia.org/wiki/Decision_tree_learning

Further visualization for another regression problem:

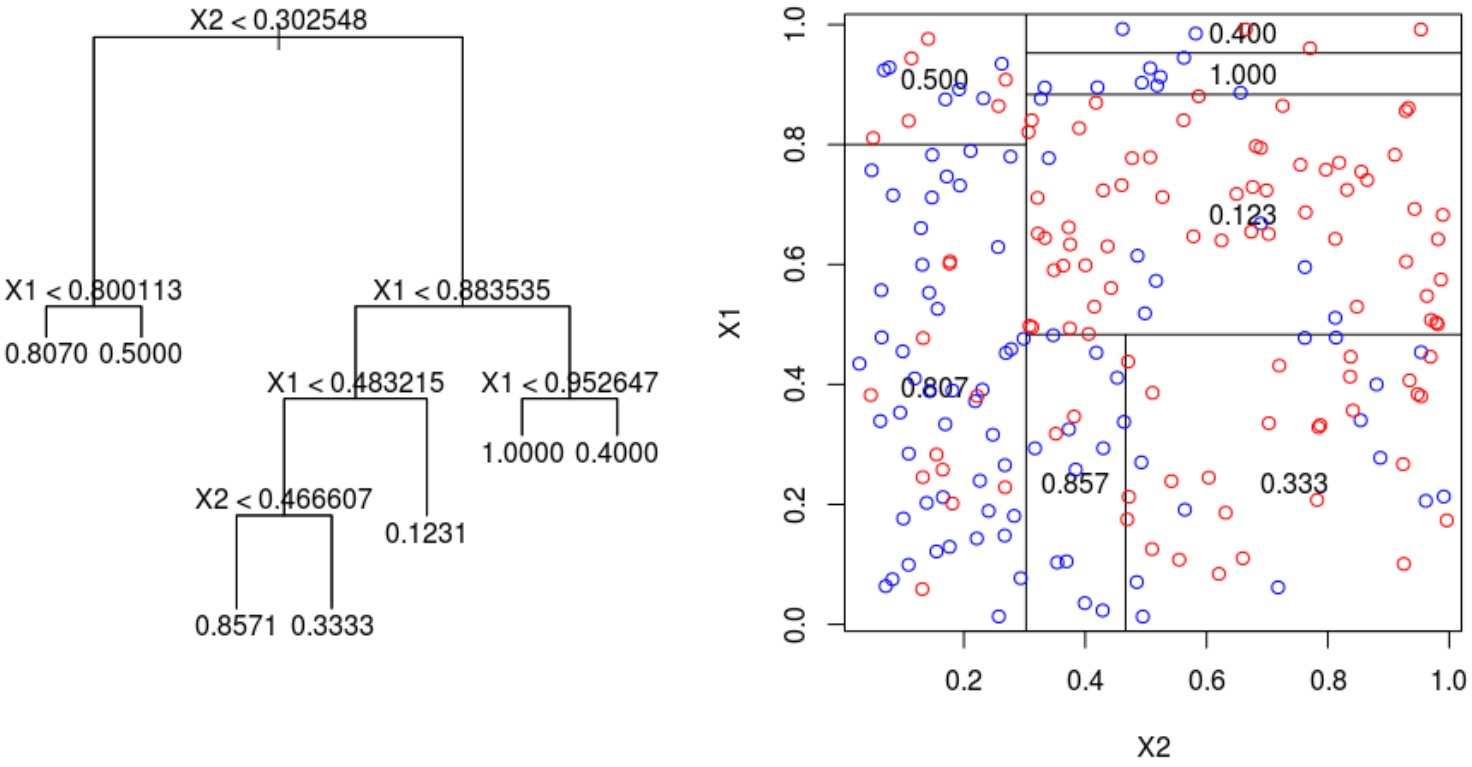
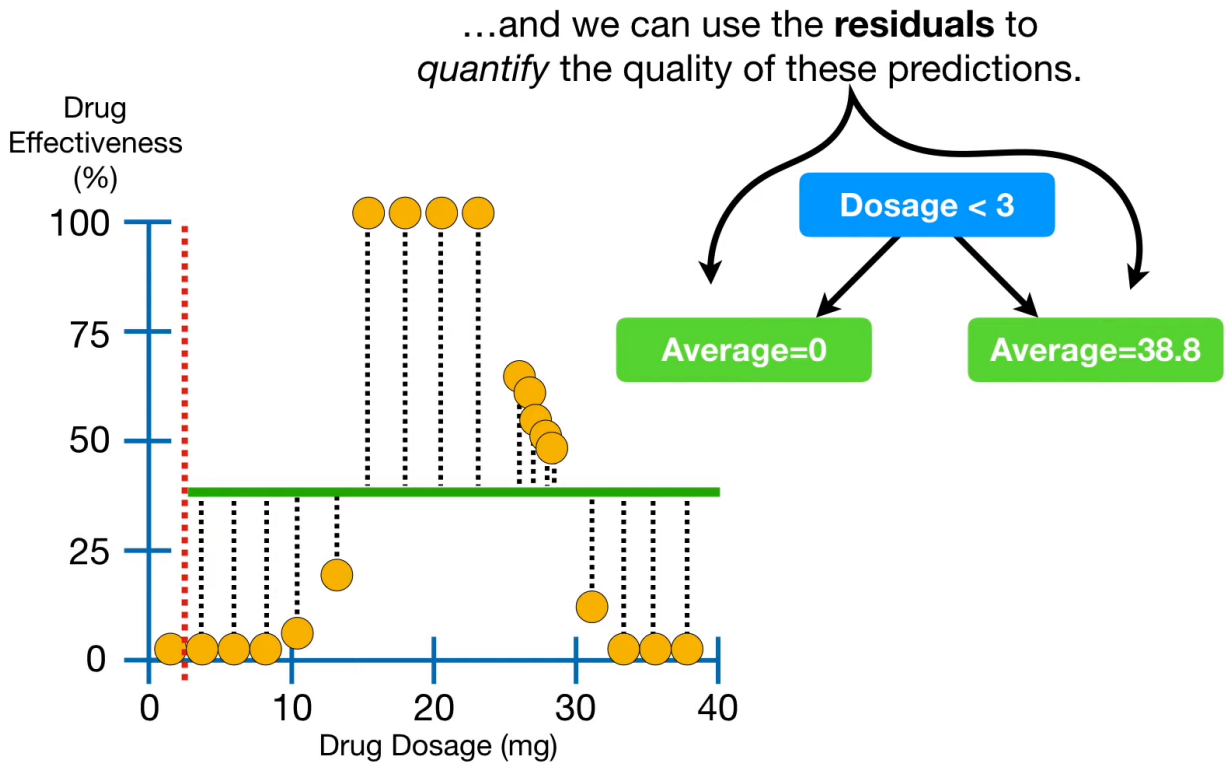
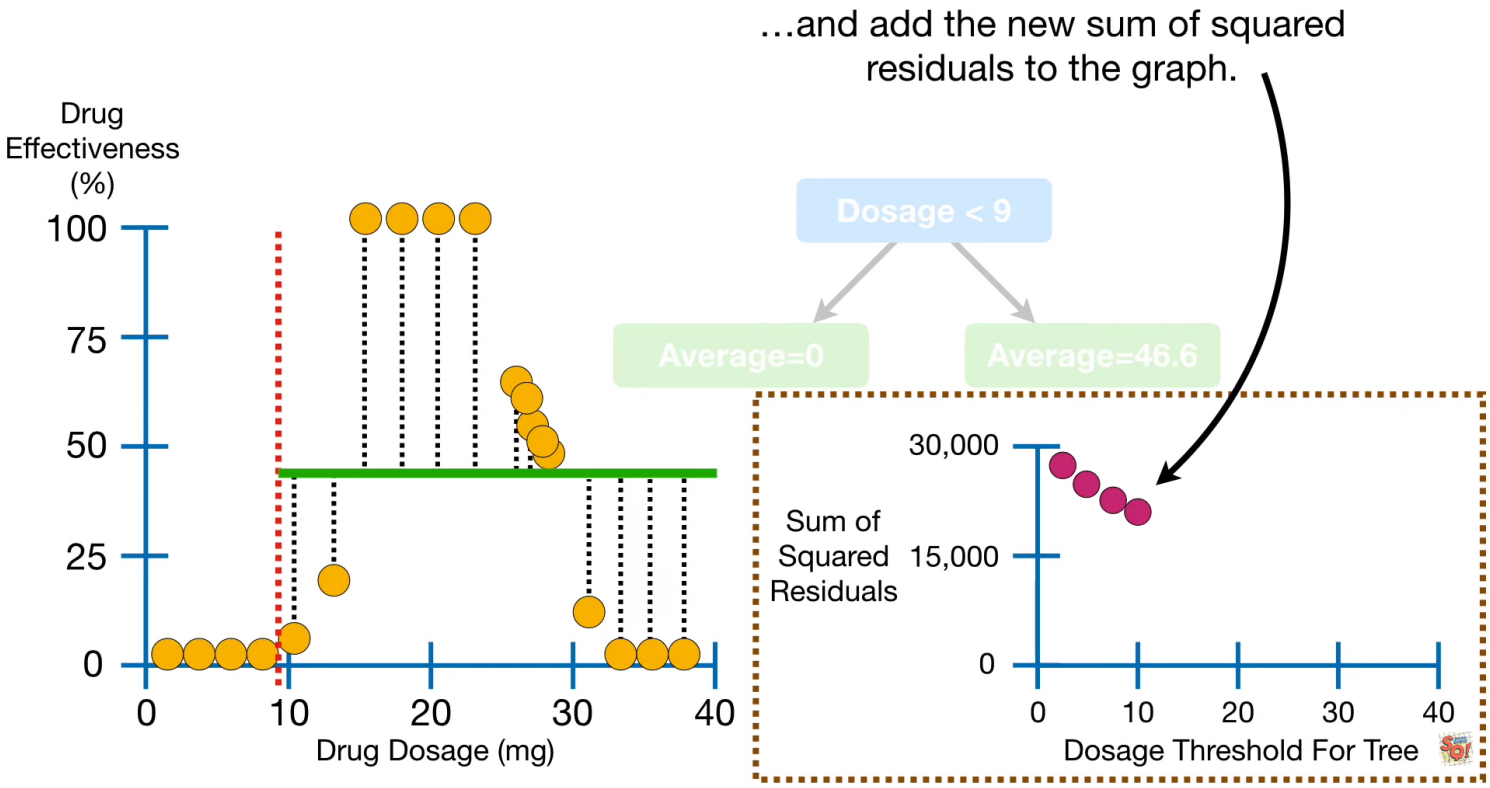


Figure Source. Decision Trees in R: https://www.datacamp.com/community/tutorials/decision-trees-R

A simple example

Let's consider a univariate case of drug effectiveness vs drug dosage. As shown in the figure below, we need to evaluate the sum of squared error or residuals (SSE or SSR) for different splits given by x eg. Dosage < x where x is 3.





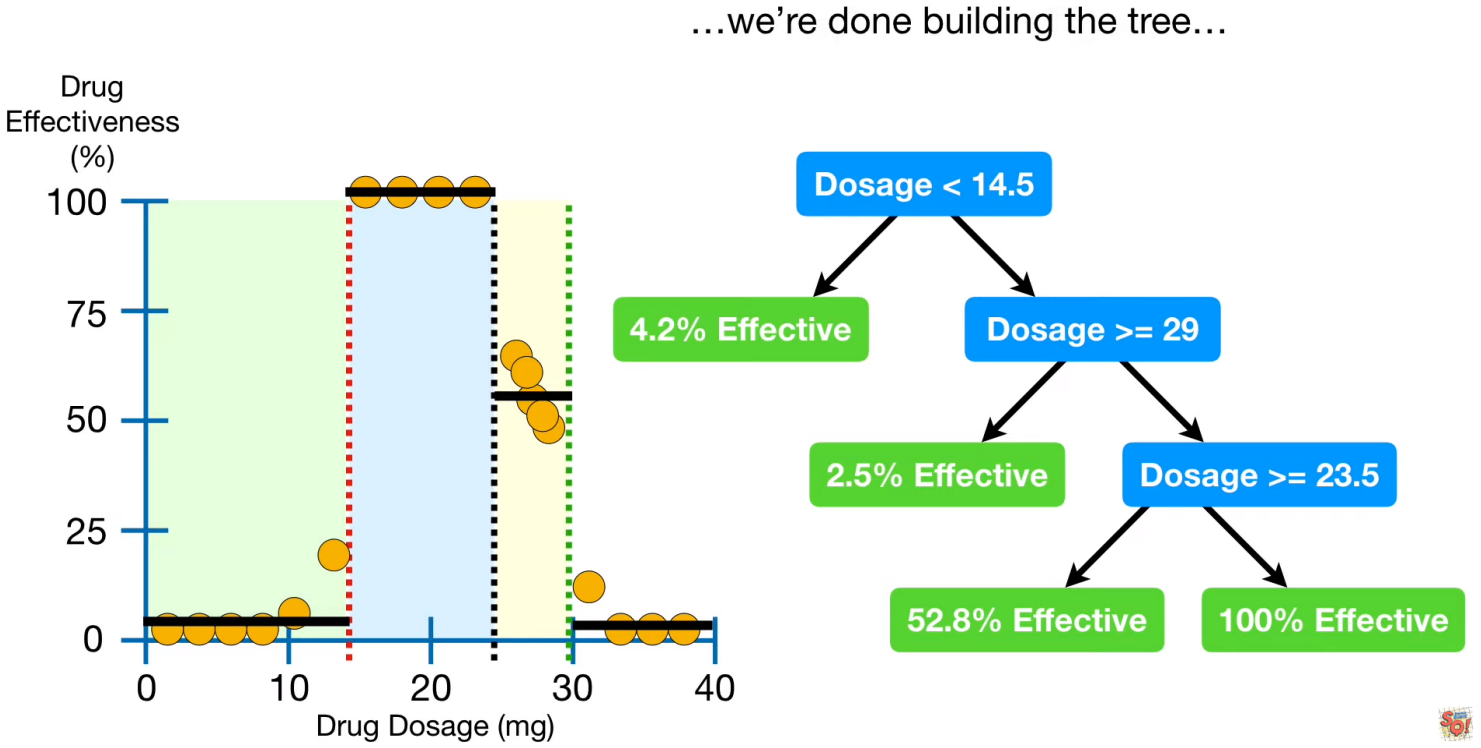


Figure Source. Regression Trees, Clearly Explained!!! by StatQuest: https://www.youtube.com/watch?v=g9c66TUylZ4&feature=emb_logo

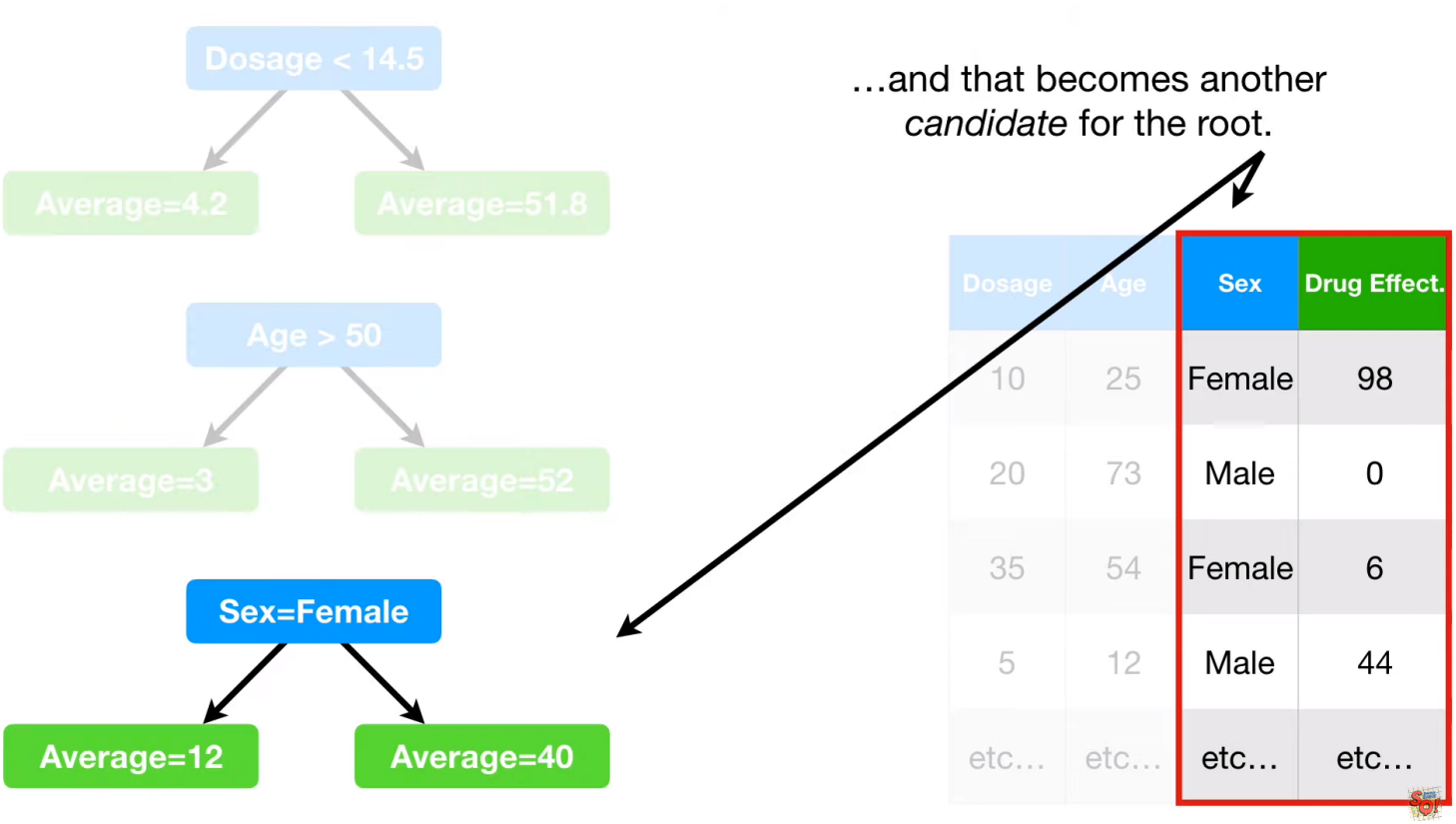
We also need to consider the number of instances (data points), in order to ensure we do not overfit the dataset. Hence we need to ensure that our regression tree is not over-complicated or does not

have a large depth that overfits the data.

We eventually find that the split of 14.5 best has the best error taking into account that a maximum of 7 data points is part of the split. Therefore, we use Dosage < 14.5 as the root and then move on from there to see how the leaves can be split. Hence, Dosage < 14.5 is 4.2 % effective.

On the right leaf, we place another note that looks at the case when Dosage >=29. Hence, Dosage >= 29 is 2.5 % effective. Now we are left with the case where data points cover Dosage >=14.5 and < 29. We can see that there are two chunks of points, and hence it is important to split the data further. This is why we have the last node where Dosage >= 23.5 and two leaves. Note that the leaves give the average values.

Looking into multiple attributes we need to revisit the idea of selecting the root node of the decision tree for classification problems. Similar to calculating the best Gini or Information Gain for different combinations of the attributes with the outcome, we will consider the SSE or SSR for them and pick the best one.



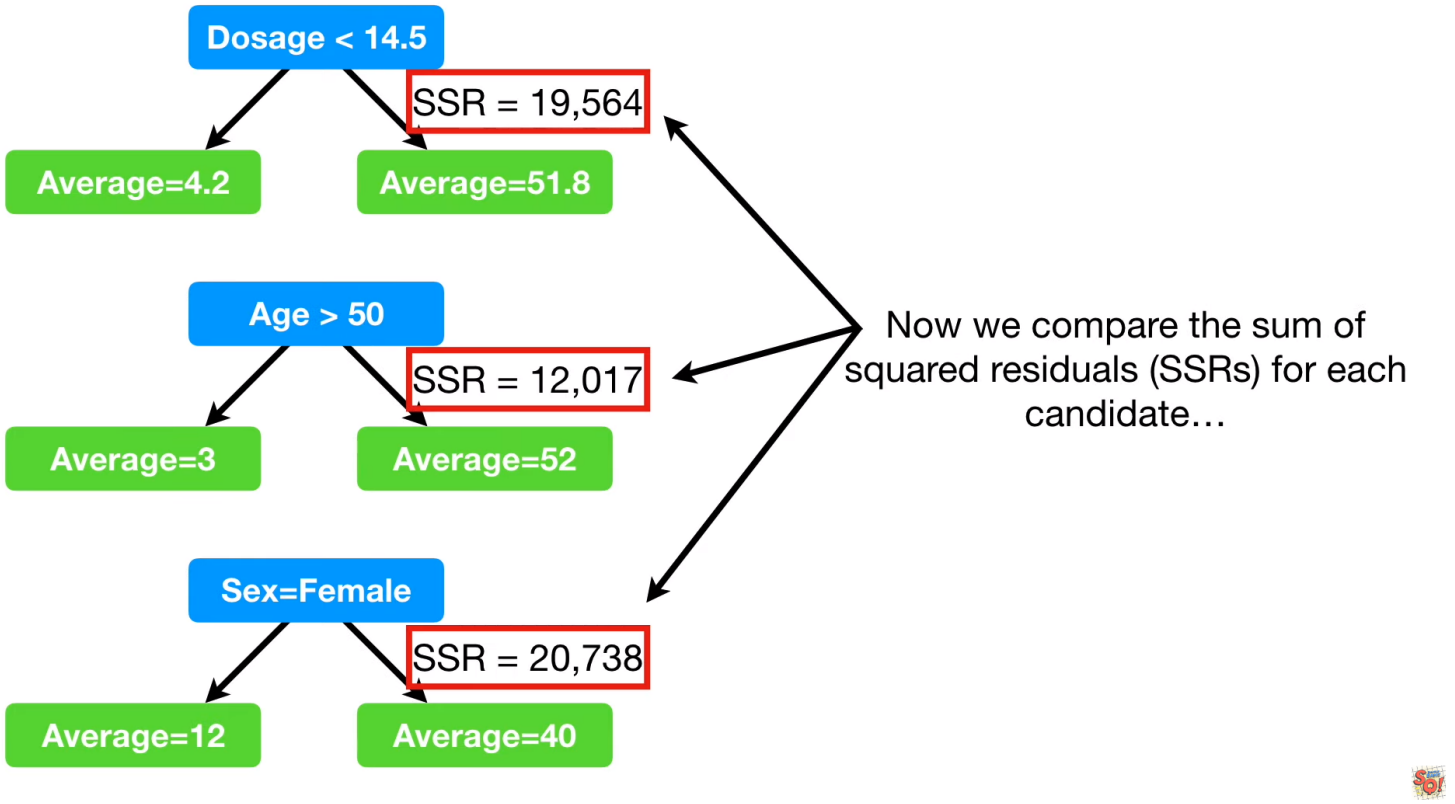


Figure Source. Regression Trees, Clearly Explained!!! by StatQuest: https://www.youtube.com/watch?v=g9c66TUylZ4&feature=emb_logo

Effect of Depth

The goal of the depth parameter in the CART algorithm is to ensure that you do not build complicated trees that overfit the data.

The following figure plots the above model's predictions when the maximum tree depth is restricted to 2 (on the left) and 3 (on the right). The red horizontal lines represent the average target values (y value) for the cases (blue dots) in the respective regions. The CART algorithm splits regions such that the target values of the training cases (blue dots) are as close as possible to the respective predicted values (red lines).

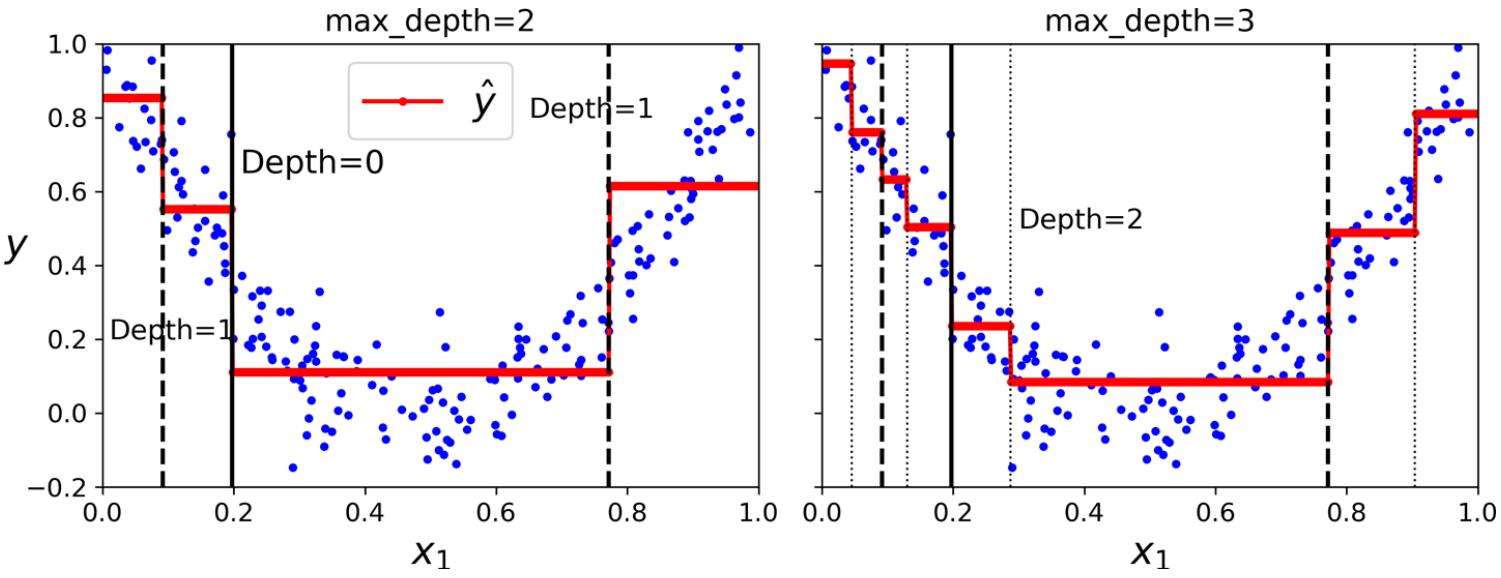


Figure: Plot of CART algorithm. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by A.* Géron, 2019, Sebastopol; CA: O'Reilly Media.

Play Tennis Example Revisited

Lets' look at another example of a regression tree that builds from a regression data adapted from the play tennis example.

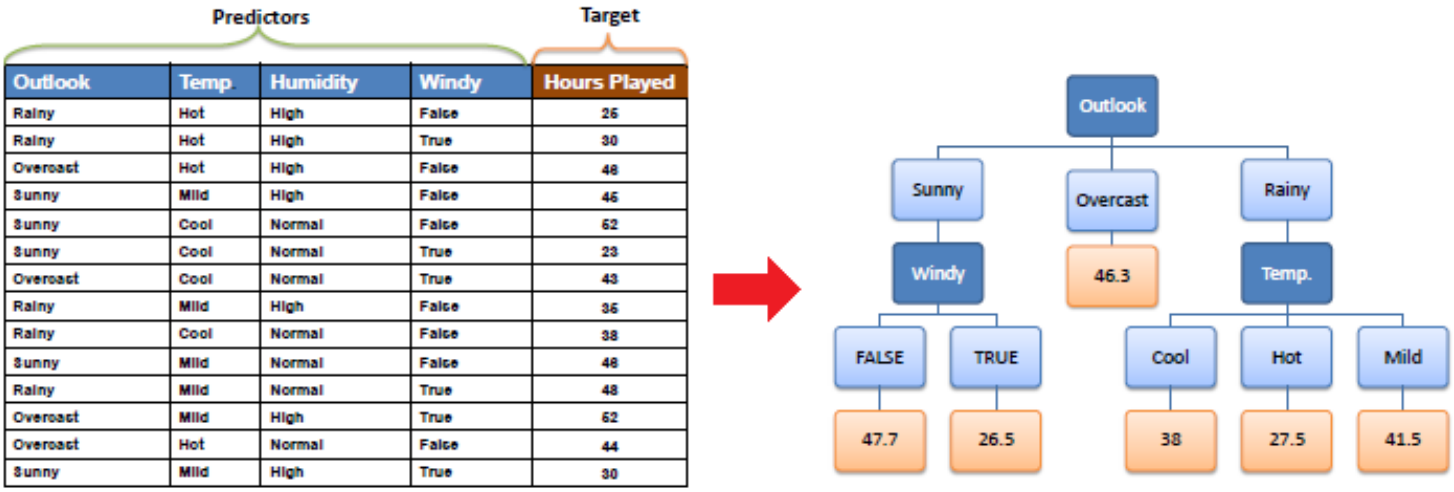


Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

We now return look at one attribute (outcome) for the dataset as shown below.

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$\boldsymbol{Count} = n = 14$$

$$\boldsymbol{Average} = \bar{x} = \frac{\sum x}{n} = 39.8$$

$$\boldsymbol{Standard\ Deviation} = S = \sqrt{\frac{\sum (x - \bar{x})^2}{n}} = 9.32$$

$$\boldsymbol{Coeffeicient\ of\ Variation} = CV = \frac{S}{\bar{x}} * 100\% = 23\%$$

1. Standard Deviation (**S**) is for tree building (branching).
2. Coefficient of Deviation (**CV**) is used to decide when to stop branching. We can use Count (**n**) as well.
3. Average (**Avg**) is the value in the leaf nodes.

Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Next, we look at the standard deviation S for **two** attributes (outcome T and attribute X) with probability P as shown below.

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)*__S__(Sunny) + **P**(Overcast)*__S__(Overcast) + **P**(Rainy)*__S__(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Standard Deviation Reduction

The standard deviation reduction is based on the decrease in standard deviation after a dataset is split on an attribute. Constructing a regression tree is all about finding attribute that returns the highest standard deviation reduction (i.e., the most homogeneous branches).

Step 1: The standard deviation of the target is calculated.

Standard deviation (Hours Played) = 9.32

Step 2: The dataset is then split on the different attributes. and t standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR=0.17		

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

$$SDR(T,X) = S(T) - S(T,X)$$

SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

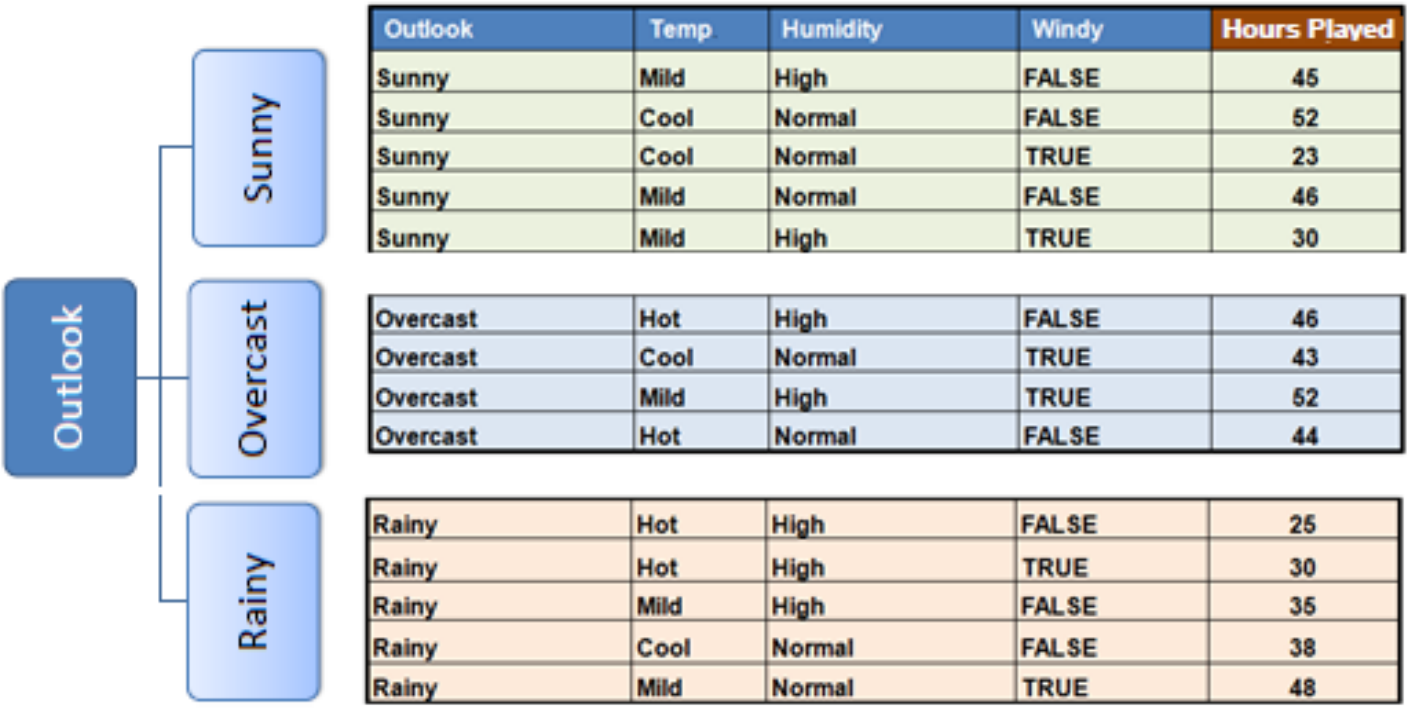
= 9.32 – 7.66 = 1.66

Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Step 3: The attribute with the largest standard deviation reduction is chosen for the decision node.

★		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

Step 4a: The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches until all data is processed.



Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48

Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Our termination criteria is when the coefficient of deviation (**CV**) for a branch becomes smaller than a certain threshold (e.g., 10%) and/or when too few instances (**n**) remain in the branch (e.g., 3). ***Step 4b***: "Overcast" subset does not need any further splitting because its CV (8%) is less than the threshold (10%). The related leaf node gets the average of the "Overcast" subset.

Outlook - Overcast

		Hours Played (StDev)	Hours Played (AVG)	Hours Played (CV)	Count
Outlook	Overcast	3.49	46.3	8%	4
	Rainy	7.78	35.2	22%	5
	Sunny	10.87	39.2	28%	5

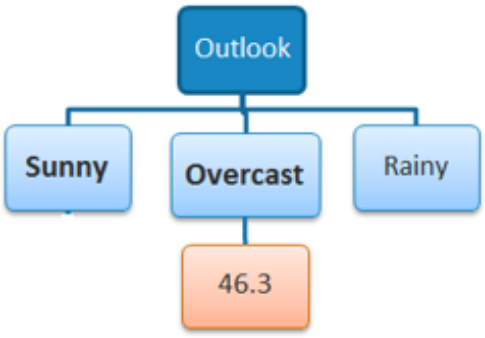


Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Step 4c: However, the "Sunny" branch has a CV (28%) more than the threshold (10%) which needs further splitting. We select "Windy" as the best node after "Outlook" because it has the largest SDR.

Outlook - Sunny

Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Cool	Normal	TRUE	23
Mild	Normal	FALSE	46
Mild	High	TRUE	30
			S = 10.87
			AVG = 39.2
			CV = 28%

		Hours Played (StDev)	Count
Temp	Cool	14.50	2
	Mild	7.32	3

SDR = 10.87-((2/5)*14.5 + (3/5)*7.32) = **0.678**

		Hours Played (StDev)	Count
Humidity	High	7.50	2
	Normal	12.50	3

SDR = 10.87-((2/5)*7.5 + (3/5)*12.5) = **0.370**

		Hours Played (StDev)	Count
Windy	False	3.09	3
	True	3.50	2

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5) = **7.62**

Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Since the number of data points for both branches (FALSE and TRUE) is equal or less than 3, we stop further branching and assign the average of each branch to the related leaf node.

Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Mild	Normal	FALSE	46
Cool	Normal	TRUE	23
Mild	High	TRUE	30

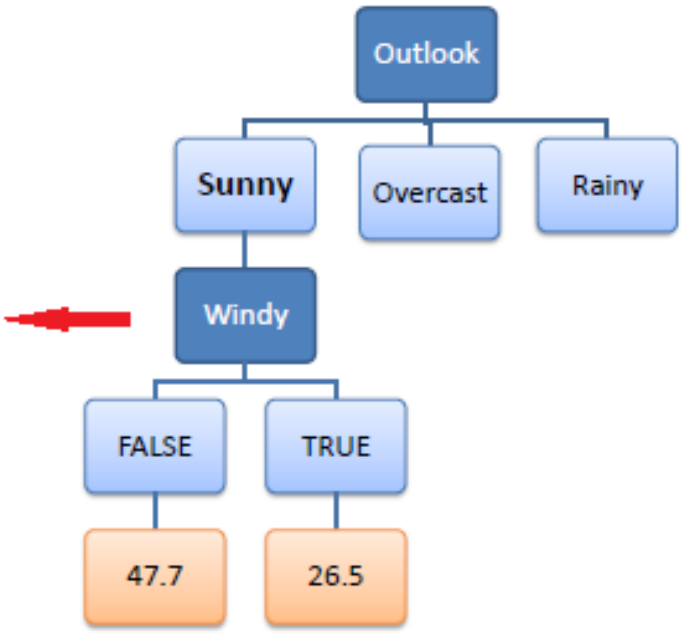


Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Step 4d: Furthermore, the "rainy" branch has a CV (22%) which is more than the threshold (10%). This branch needs further splitting. We select "Windy" as the best node because it has the largest SDR.

Outlook - Rainy

Temp	Humidity	Windy	Hours Played
Hot	High	FALSE	**25**
Hot	High	TRUE	**30**
Mild	High	FALSE	**35**
Cool	Normal	FALSE	**38**
Mild	Normal	TRUE	**48**
			S = 7.78
			AVG = 35.2
			CV = 22%

		Hours Played (StDev)	Count
Temp	Cool	0	1
	Hot	2.5	2
	Mild	6.5	2

SDR = 7.78 - ((1/5)*0+(2/5)*2.5 + (2/5)*6.5) = **4.18**

		Hours Played (StDev)	Count
Humidity	High	4.1	3
	Normal	5.0	2

SDR = 7.78 - ((3/5)*4.1 + (2/5)*5.0) = **3.32**

		Hours Played (StDev)	Count
Windy	False	5.6	3
	True	9.0	2

SDR = 7.78 - ((3/5)*5.6 + (2/5)*9.0) = **0.82**

Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Because the number of data points for all three branches (Cool, Hot and Mild) is equal or less than 3 we stop further branching and assign the average of each branch to the related leaf node.

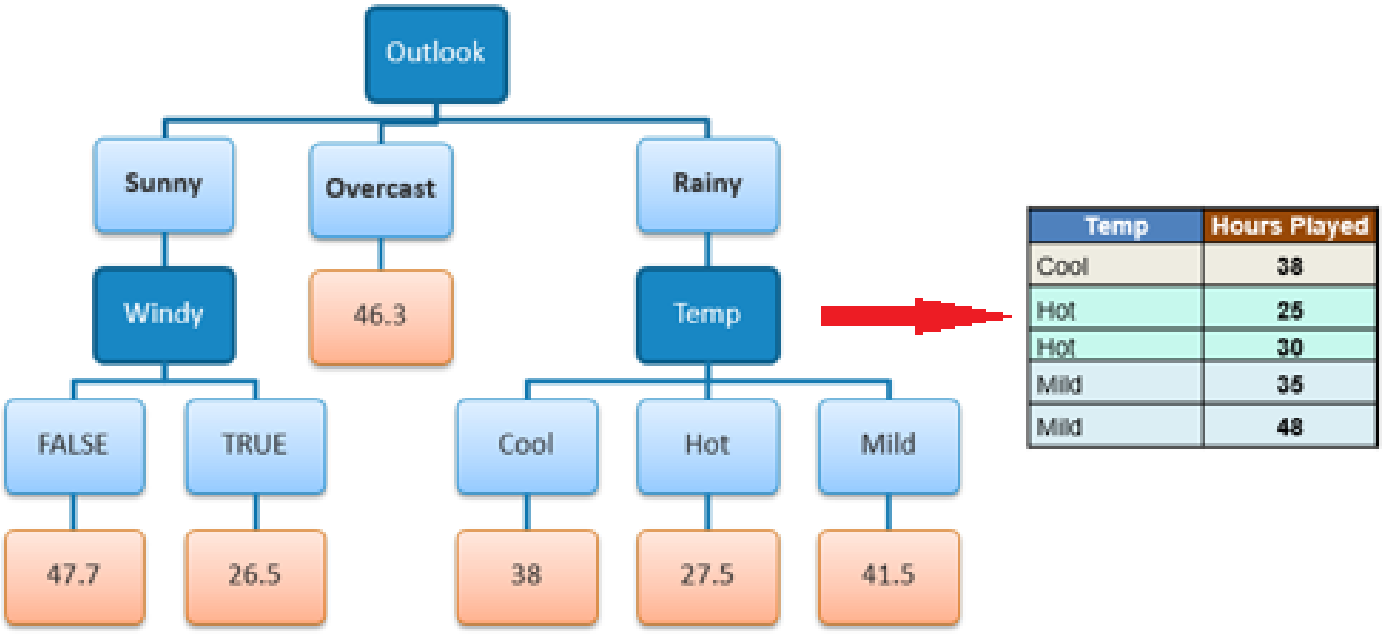


Figure Source. Decision Tree - Regression: https://www.saedsayad.com/decision_tree_reg.htm

Run

PYTHON

```
1
2 import numpy as np
3 import pandas as pd
4 import seaborn as sns
5 from pylab import rcParams
6 import matplotlib.pyplot as plt
7 import matplotlib.animation as animation
8 from matplotlib import rc
9 import unittest
10 import math
11 from sklearn import metrics
12 from sklearn.tree import export_graphviz
13 import IPython, graphviz, re
14
```

Regression Tree with Scikit-learn

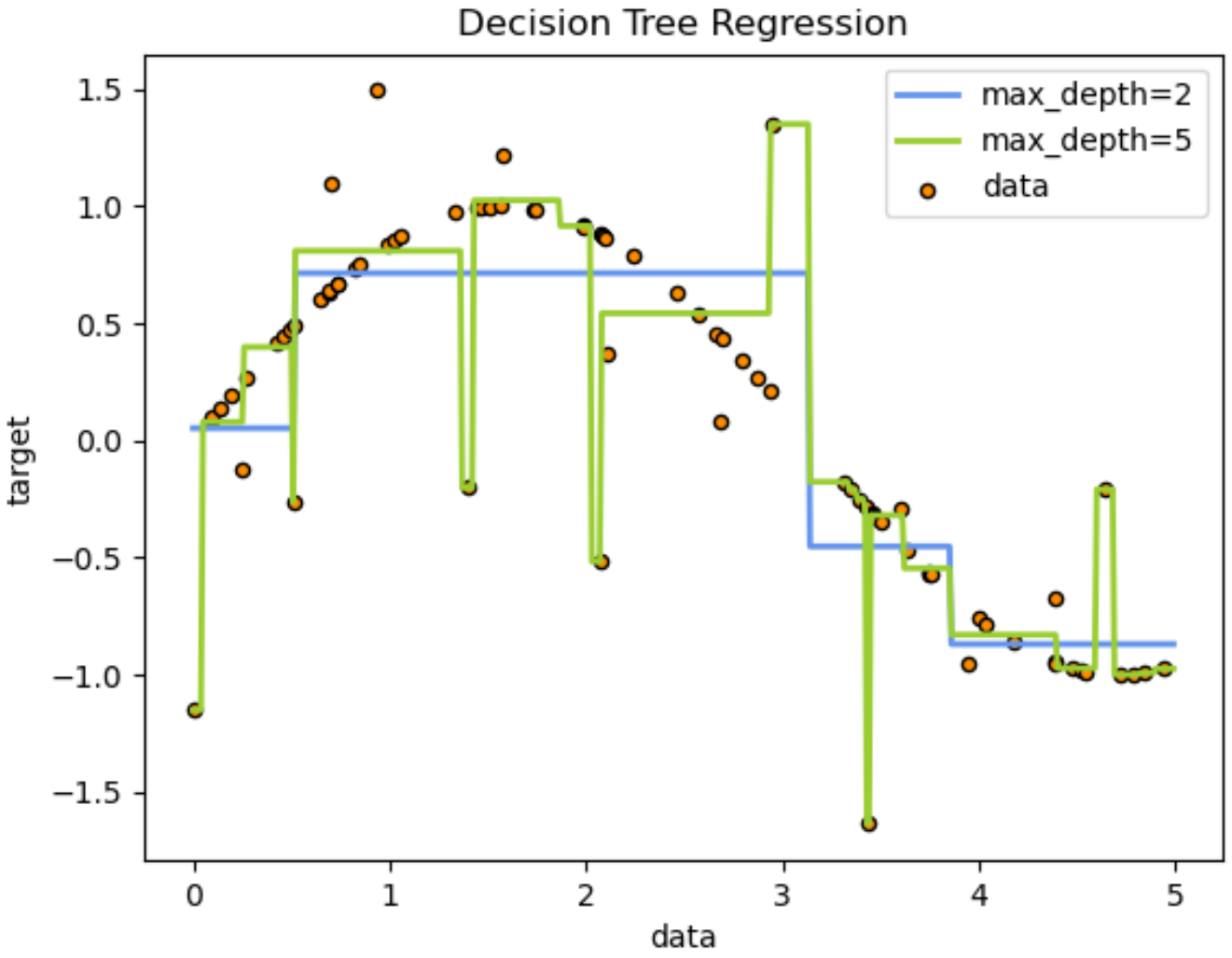
Univariate regression with regression tree to fit a sine curve with noise. We notice that if the maximum depth of the tree is set too high, the decision trees overfits the data. See code below.

Run PYTHON

```
#source: https://scikit-learn.org/stable/auto_examples/tree/plot_tree_re

import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))

# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=2)
```

Code Source: https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html#sphx-glr-auto-examples-tree-plot-tree-regression-py

The output from above code:



Next, we look at an example to show multi-output regression with regression tree which is used to predict simultaneously the noisy x and y observations of a circle given a single underlying feature.

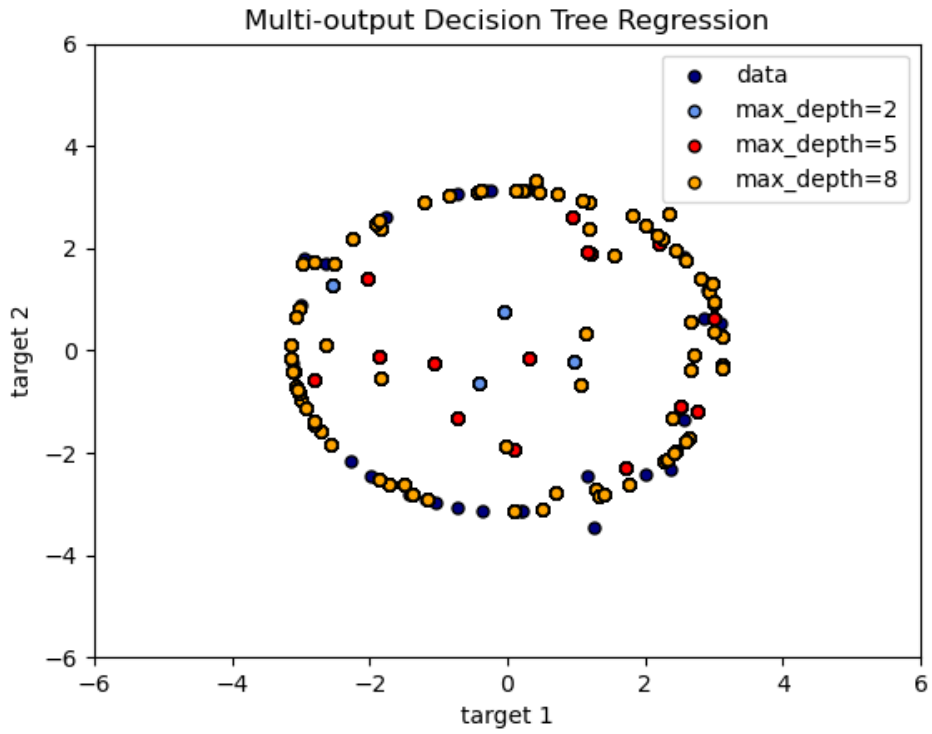
We notice that if the maximum depth of the tree is set too high, the regression tree overfits by learning too fine details of the training data.

Run PYTHON

```
1
2 #Source: https://scikit-learn.org/stable/auto_examples/tree/plot_tree_re
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from sklearn.tree import DecisionTreeRegressor
7
8 # Create a random dataset
9 rng = np.random.RandomState(1)
10 X = np.sort(200 * rng.rand(100, 1) - 100, axis=0)
11 y = np.array([np.pi * np.sin(X).ravel(), np.pi * np.cos(X).ravel()]).T
12 y[::5, :] += (0.5 - rng.rand(20, 2))
13
14 # Fit regression model
```

Code Source: https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression_multioutput.html#sphx-glr-auto-examples-tree-plot-tree-regression-multioutput-py

The output from the above code:



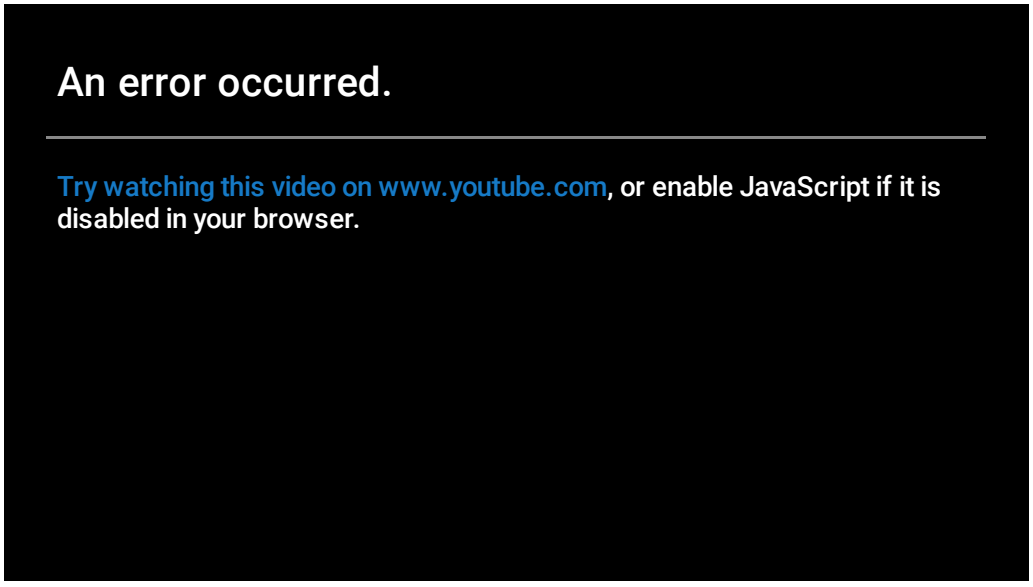
Regression Tree in R

Run R

```r
#https://www.datacamp.com/community/tutorials/decision-trees-R

library(ISLR)
data(package="ISLR")
carseats<-Carseats

require(tree)

names(carseats)
#Let's take a look at the histogram of car sales:

hist(carseats$Sales)


```

Code Source: https://www.datacamp.com/community/tutorials/decision-trees-R

Finally, all of this is summarised in the video below.



References

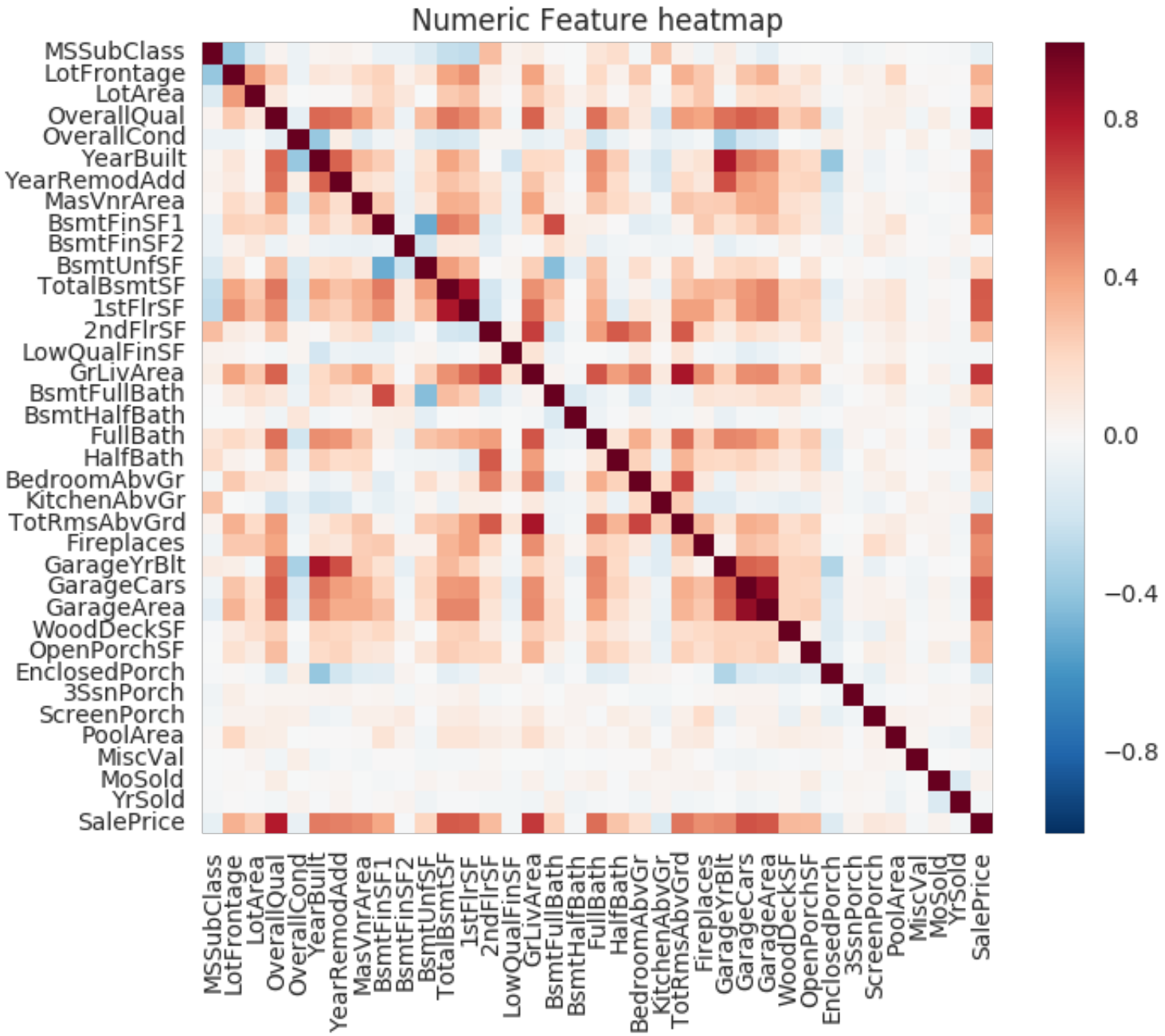
1. Elith, J., Leathwick, J. R., & Hastie, T. (2008). A working guide to boosted regression trees. *Journal of Animal Ecology*, *77*(4), 802-813: https://besjournals.onlinelibrary.wiley.com/doi/epdf/10.1111/j.1365-2656.2008.01390.x

2. Implementing Regression Tree from Scratch: https://colab.research.google.com/drive/11T18ZELpOWuP7UtM7EKOut5juhJa9cHb#scrollTo=ulPEgkKLZPLx

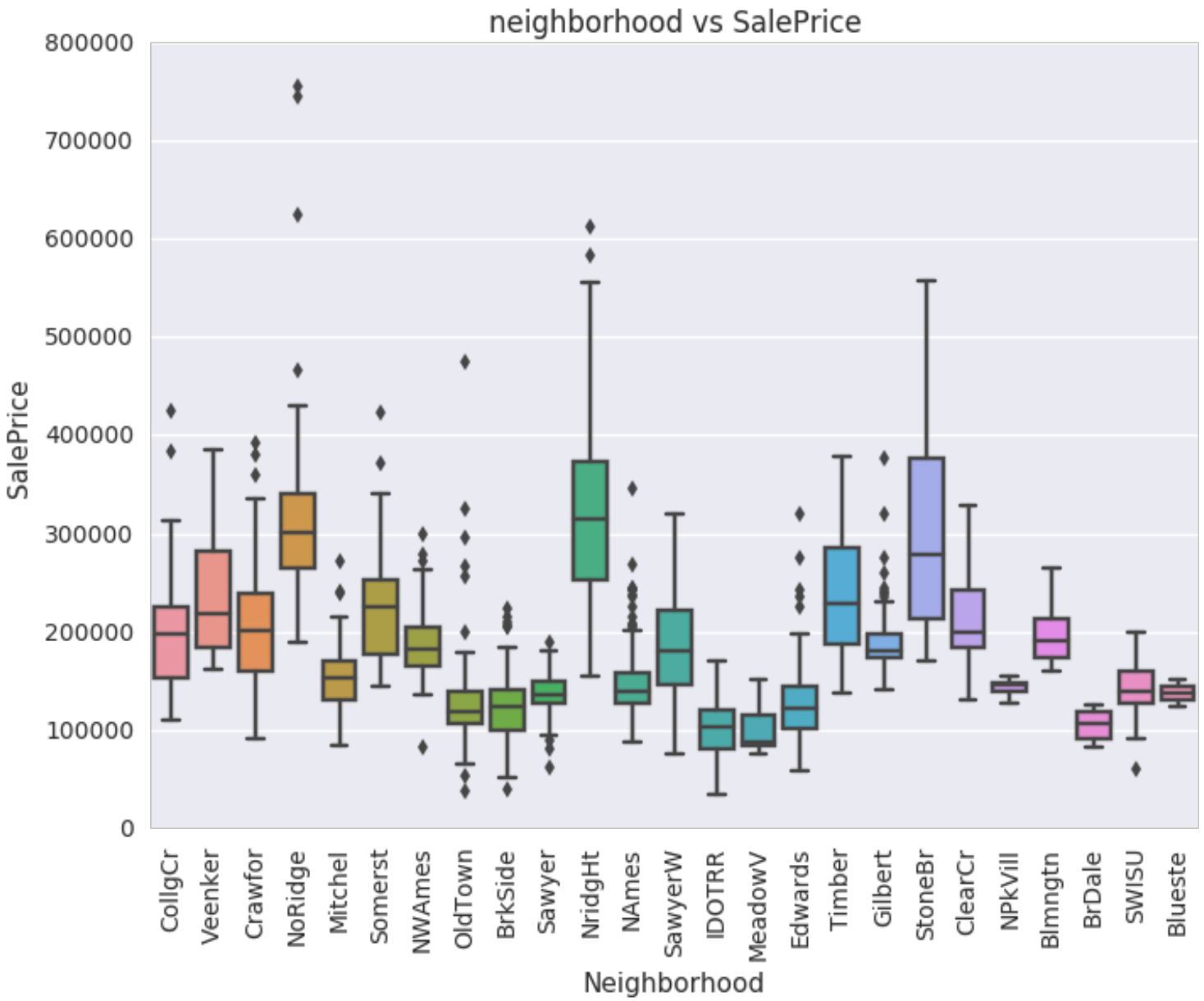
Regression Tree from Scratch

Source: https://curiously.com/posts/build-a-decision-tree-from-scratch-in-python/

We use the Kaggle data: "House Prices: Advanced Regression Techniques" which contains *1460* training data points and 80 features for the price of a house. Visualisation of the data is given below.



Source: https://www.kaggle.com/xingobar/house-price-data-visualization



Source: https://www.kaggle.com/xingobar/house-price-data-visualization

We only select specific features:

```python
X = df_train[['OverallQual', 'GrLivArea', 'GarageCars']]
y = df_train['SalePrice']
```

We show the code for implementing Regression Trees from scratch and use R2 and RMSE scores to test the performance. As an exercise, you can compare the results with Decision Tree in Scikit-learn

and R Caret.

Exercise 1

Use either Python or R

1. Select a classification dataset from UCI Machine Learning repository and create a decision tree, report training and test performance before and after pruning. (you can use the Dataset from Project I)
2. Do the same for a regression dataset.
3. Compare results with Multilayer perceptron with either Adam/SGD.

Reference

1. Boston dataset for regression: http://lib.stat.cmu.edu/datasets/boston

Overfitting vs Underfitting the data

It is very important to make sure that a model is neither overfitting nor underfitting the data. There are many approaches to identify whether a model is overfitting or underfitting the data. Below we discuss two of them from the decision tree perspective.

In **cross-validation testing**, if a model performs well on the training data set but poorly on the test data set, then we say that model is overfitting. However, if it performs poorly on both training and test datasets, then we say that model is underfitting. Often models that overfit a training data set are too complex and, on the other hand, models that underfit a training data set are too simple.

Learning curves can also be used to identify overfitting or underfitting models. A learning curve shows the performance (accuracy) of a model on both training and test data sets over a varying number of training cases. In other words, a learning curve demonstrates how error rates change over different training data set sizes.

If the validation error is consistently higher than the training error, we can say the model is overfitting the data set. See the figure below:

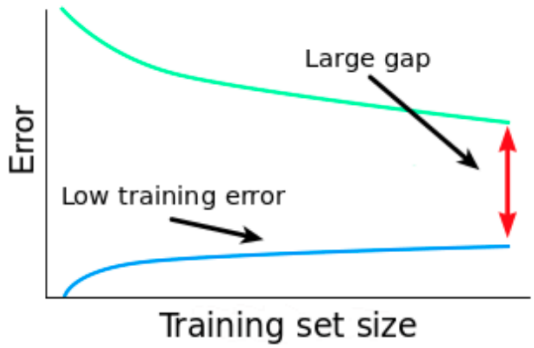


Figure: Overfitting of the data set. Adapted from "Learning curves machine learning" by Dataquest, n.d. Retrieved from https://www.dataquest.io/blog/learning-curves-machine-learning/.

However, if both the training and validation errors are close to each other and high, we say the model is underfitting the data set. See the figure below:

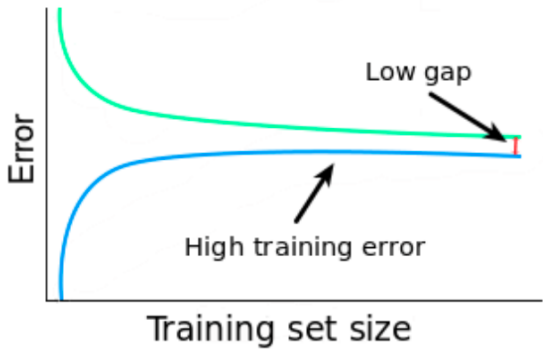


Figure: Underfitting of the data set. Adapted from "Learning curves machine learning" by Dataquest, n.d.

Retrieved from https://www.dataquest.io/blog/learning-curves-machine-learning/.

Overfitting in decision trees

Read the section titled "Regularization Hyperparameters" in Chapter 6 of Géron, 2019.

The chapter explains in detail how to avoid overfitting while learning a decision tree. Below we summarise some of the important points discussed in the section.

One of the advantages of decision trees is that they don't impose many restrictions (data should be linear, distribution shapes, etc.) on types of training data required for building a model. We don't even need to specify which attributes to consider while building a model. A decision tree learning algorithm selects a set of suitable attributes for building a model and ignores the rest of the attributes. This is a very useful feature particularly when we have many attributes (say thousands!) and it's not easy to pick suitable attributes for building a model. Such models are called **nonparametric models** because the attributes used for building a model are not determined prior to training a model. However, on the downside, this could allow a decision tree algorithm to closely fit a model to the training data set by selecting only a subset of the attributes, resulting in overfitting.

Regularisation in decision tree learning

Regularisation in machine learning refers to the process of avoiding overfitting. For decision tree learning, this means we need to restrict the degree of freedom while building a decision tree. Below are some of the popular approaches to avoid overfitting.

- the maximum depth of a decision tree (max_depth)
- the minimum number of cases per node, before it can be split (min_samples_split)
- the minimum number of cases in a leaf node (min_samples_leaf)
- the maximum number of leaf nodes (max_leaf_nodes)
- the maximum number of features that are evaluated for splitting at each node (max_features)

In scikit-learn, increasing min_* parameters or reducing max_* parameters will regularise the model.

The following figure shows two decision trees for the same data set where the tree on the left is not restricted (no parameters are changed), and the tree on the right is restricted with "min_samples_leaf=4". The tree on the right is not fitting data as closely as the tree on the left, resulting in better performance on the test data set.

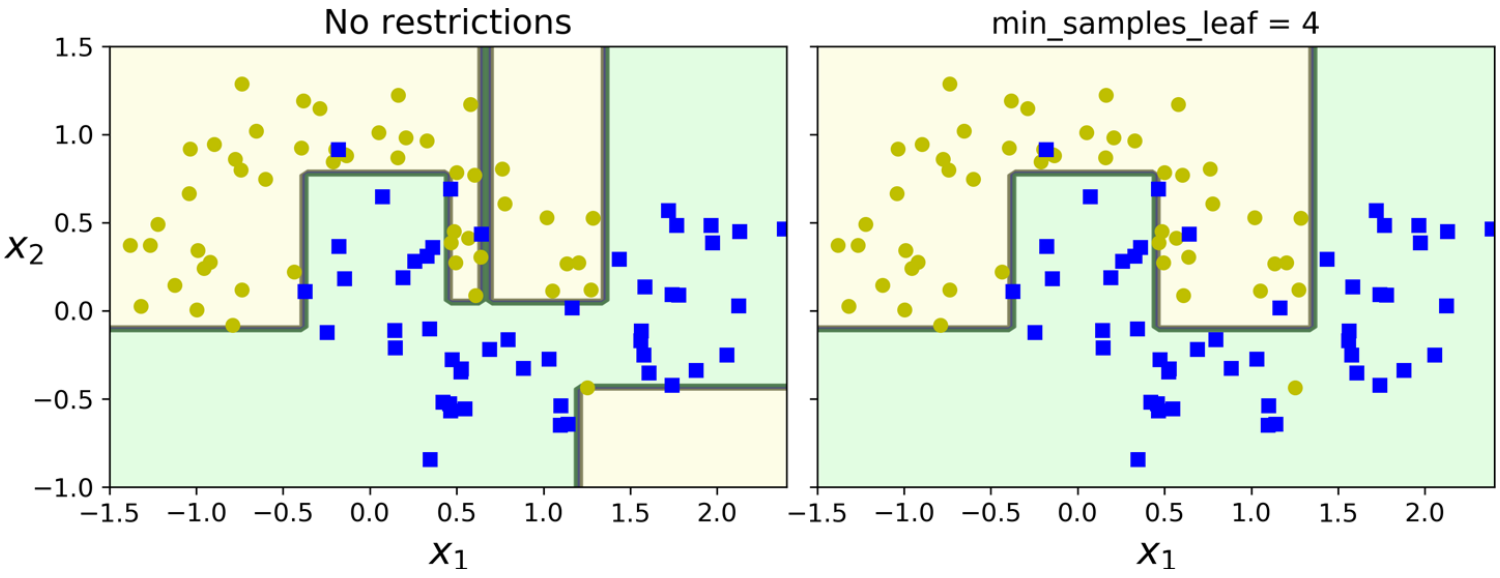


Figure: Fitting of the data set. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media. (Chapter 6)

Pruning a decision tree

At times, the size of the decision tree can be very large and complex which makes it hard to understand and such trees also may overfit the training data. It is important to simplify or prune them and there are methods do so without making a large difference in accuracy and to help improve generalisation performance.

In this approach, we first build a decision tree without any restrictions. However, later we remove nodes that are not (statistically) significantly improving prediction accuracy. For example, if all children of a node are leaf nodes and the split at that node offers only minor information gain (not statistically significant), we may consider that split unnecessary and remove it. Normally standard statistical tests, like the Chi-squared test, are used in determining whether a split is significant or not. If it is not statistically significant, that split and its children are removed. Normally pruning starts from lower levels and continues upwards until all unnecessary nodes are pruned.

Please note that there are many different approaches for pruning a given tree and often we can set the required parameter values for the required pruning for a given algorithm.

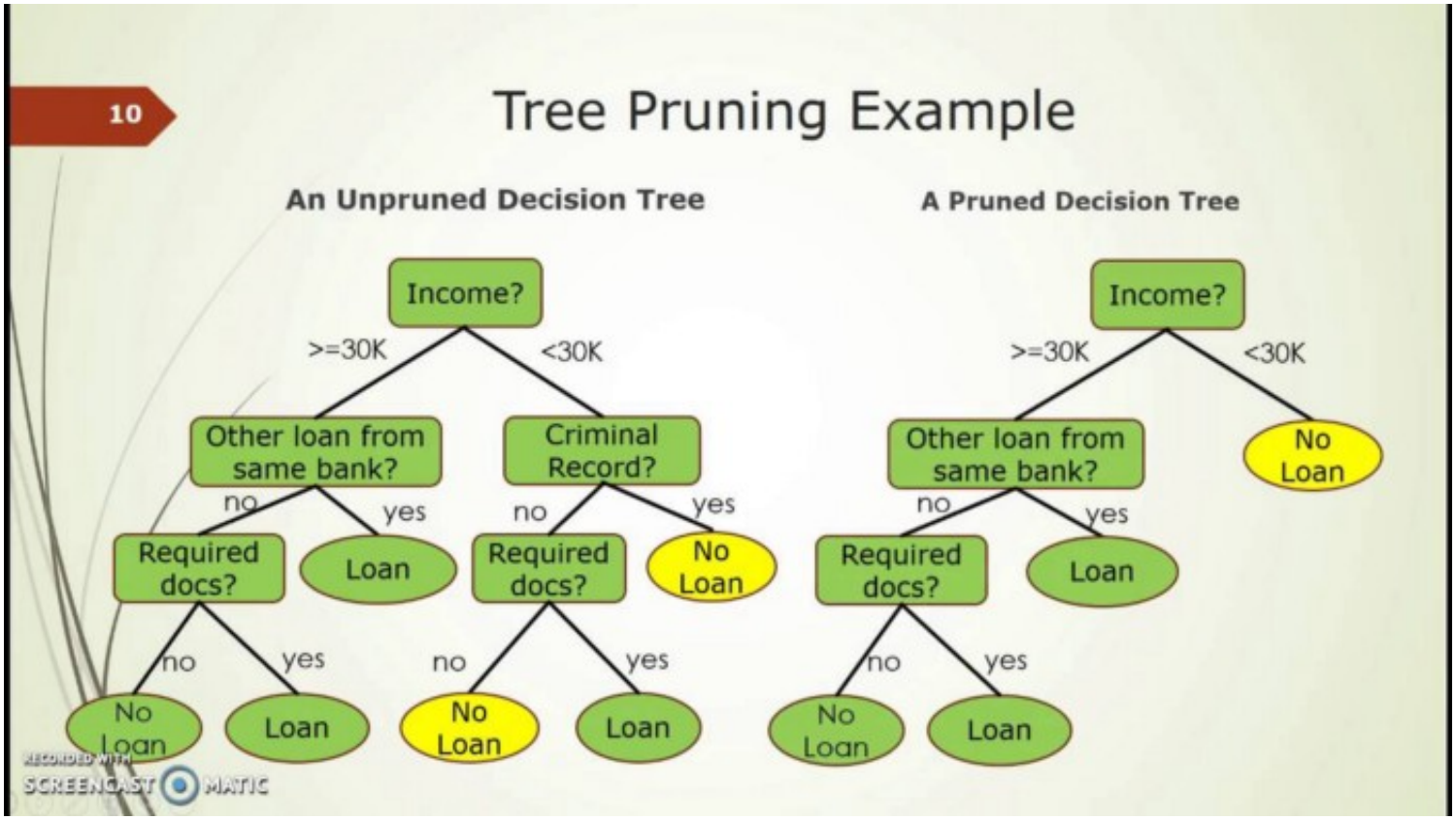


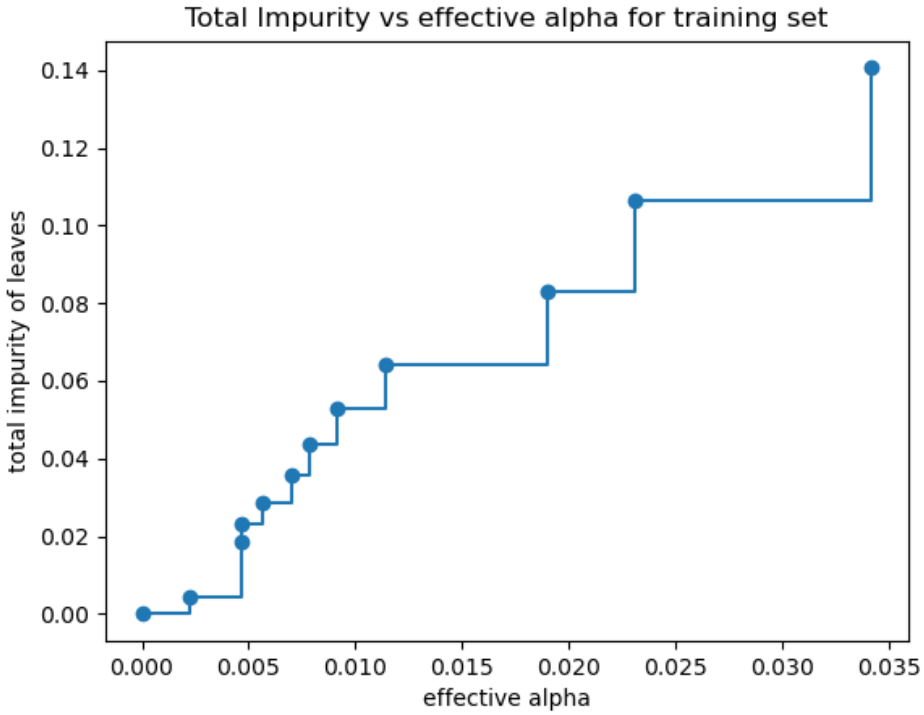
Figure Source. Decision Trees Explained Easily: https://medium.com/@chiragsehra42/decision-trees-explained-easily-28f23241248

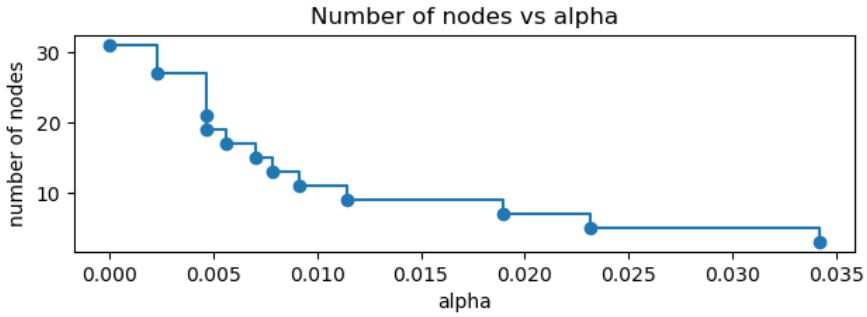
Below is an example in Scikit-learn

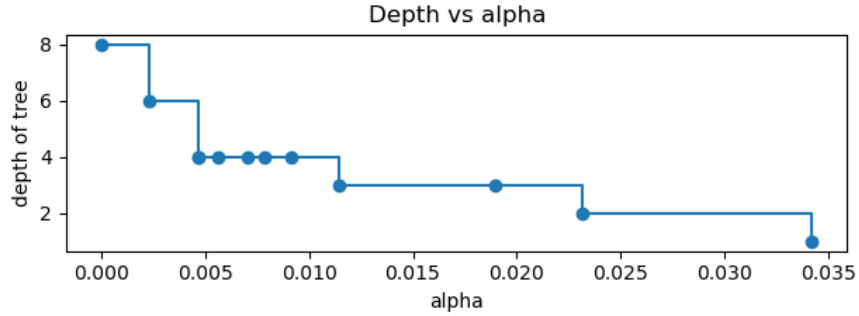
Run PYTHON

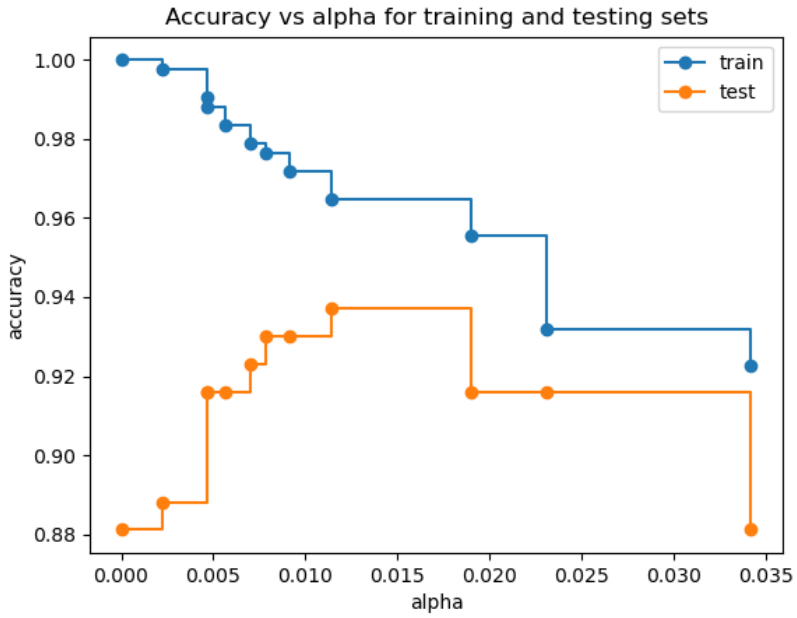
```
#Source: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_co
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0

clf = DecisionTreeClassifier(random_state=0)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots()
```

The outputs are given below


Total Impurity vs effective alpha for training set
total impurity of leaves
effective alpha
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035


Number of nodes vs alpha
number of nodes
alpha
10
20
30
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035

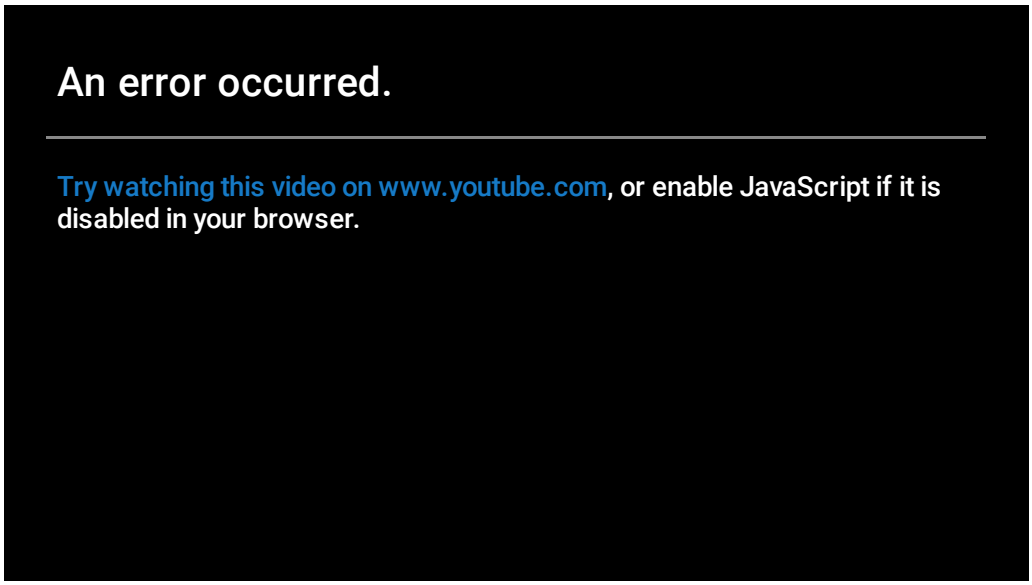

Depth vs alpha
depth of tree
alpha
2
4
6
8
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035



Code and Figure Source: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html

Below is an example in R with a detailed tutorial.

http://www.socr.umich.edu/people/dinov/courses/DSPA_notes/08_DecisionTreeClass.html



References

1. Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies*, *27*(3), 221-234. https://dspace.mit.edu/bitstream/handle/1721.1/6453/aim-930.pdf?sequence=2

Ensemble learning

Ensemble learning tries to exploit **collective wisdom** believing that collective wisdom derived from many decision-makers is often smarter than a single decision-maker. We use a similar approach in our day-to-day activities, like reading reviews before deciding which product to buy, consulting multiple medical experts before deciding on a critical surgery, or conducting multiple interviews (by different people) of a candidate before deciding whether to recruit them, etc.

Ensemble learning refers to the process of generating multiple predictive models and strategically combining their predictions to derive a final prediction. Here, an ensemble refers to a group of predictors. The technique is called ensemble learning and an algorithm is called an **ensemble method**.

In essence, ensemble methods in machine learning have the following two things in common:

- they construct multiple, diverse predictive models from adapted versions of the training data (most often reweighed or resampled)
- they combine the predictions of these models in some way, often by simple averaging or voting (possibly weighted).

The basic idea of ensembles or “multi-level” learning schemes is to build different “experts” and let them vote.

Advantage: improves predictive performance.

Disadvantage: produces output that is very hard to interpret.

Diversity of the models

Diversity of the models is often a key to improving overall performance (accuracy) of an ensemble method. Interestingly, combining diverse algorithms often outperforms combining similar algorithms. Ensemble methods are good at learning from different types of errors resulting from diverse algorithms, resulting in improved predictions.

As discussed in chapter 7 of Géron, 2019, we use many different approaches during a typical project and eventually combine selected multiple predictors using a suitable ensemble method to derive a better predictor. For example, we can build multiple decision tree models by selecting the following:

- random subset of the attributes
- random subset of the training set.

For a new case, we derive predictions from all the above models and then predict based on the majority of votes (prediction with majority occurrence). This approach, called a Random Forest, is widely used and one of the most effective ensemble learning methods.

Voting classifiers

As discussed in the section "Voting Classifiers" in Chapter 7 of Géron, 2019, let's suppose we use different learning algorithms to build multiple classifiers. A simple way to combine these diverse predictors is to predict the class that gets a majority of votes. Such an ensemble method is known as **hard voting classifier**.

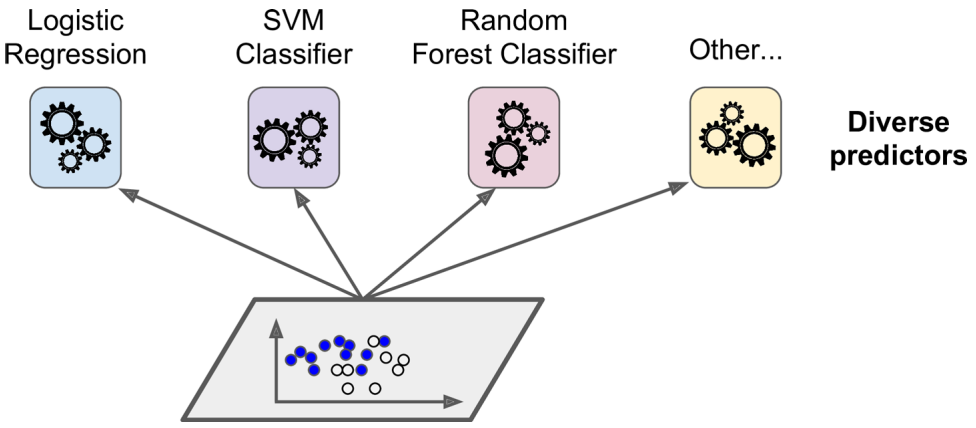


Figure: Training diverse classifiers. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

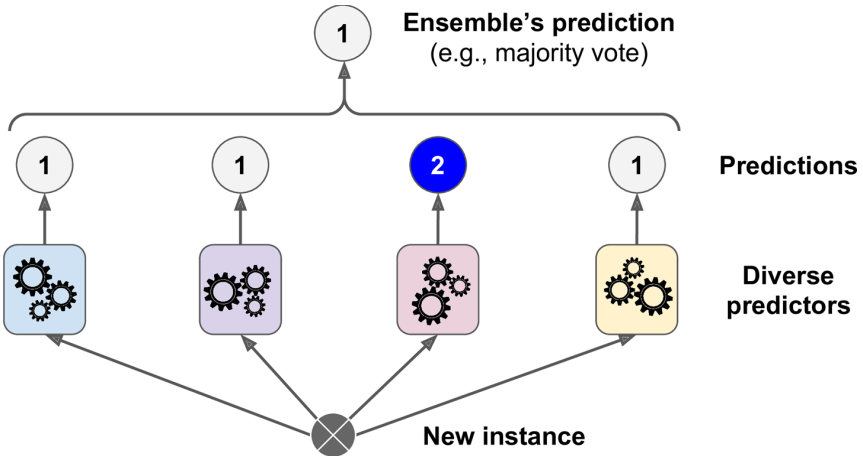


Figure: Hard voting classifier predictions. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Points to consider for an ensemble method:

- Combining multiple **weak learners** (low accuracy) often results in a **strong learner** (with high accuracy) provided there are enough weak learners and they are diverse.
- Voting classifiers often achieve higher accuracy than the best classifier in the ensemble.

Example

Let's build three diverse classifiers using the moons data set and combine their predictions using a hard voting classifier (from Chapter 7 of Géron, 2019).

The following provides the accuracy of each classifier on the test data set.

Run PYTHON

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()
```

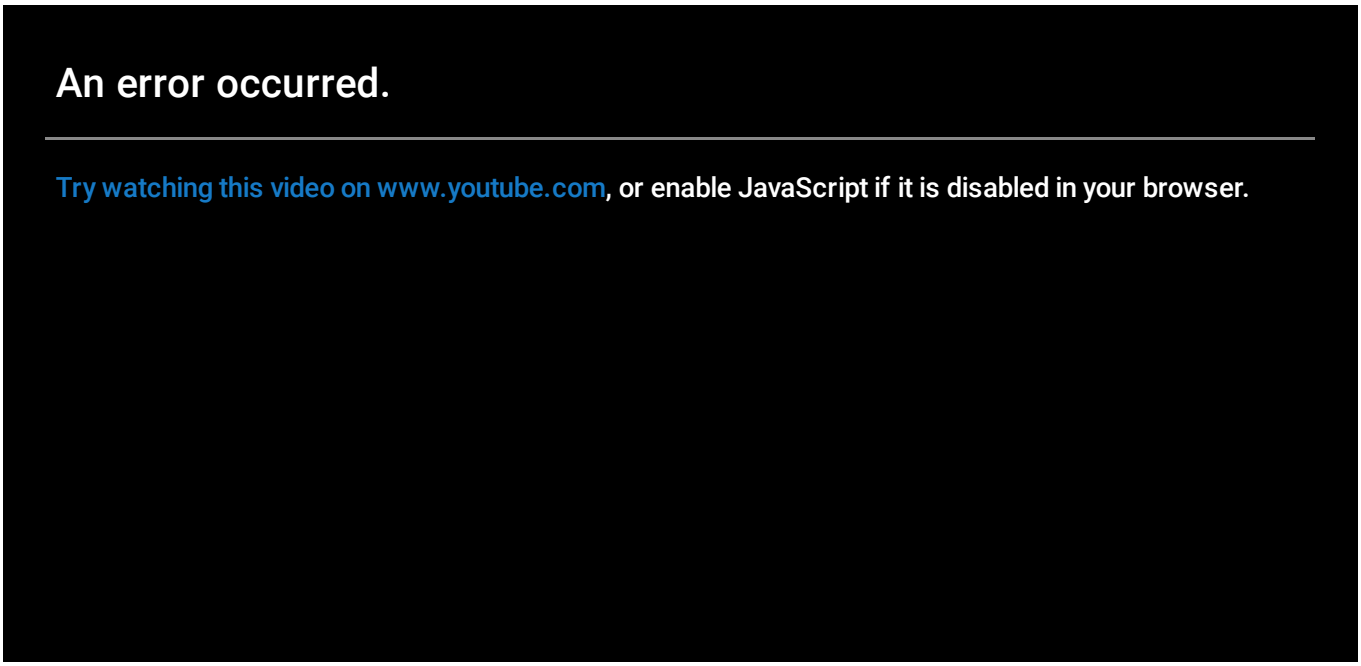
As you can see, the accuracy of the ensemble method (VotingClassifier) is the highest, even though the improvement here is small.

In the following slides we will discuss popular schemes for ensemble learning:

- Bagging
- Random forests

Next week, we will cover

- Boosting with Adaboost, Gradient Boosting and XGboost
- Stacking



[redacted]

References

1. Liu, Y., & Yao, X. (1999). Ensemble learning via negative correlation. *Neural networks*, *12*(10), 1399-1404: https://doi.org/10.1016/S0893-6080(99)00073-8
2. Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1249: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1249

Bagging

Use the same learning algorithm and build multiple models by randomly selecting subsets of the training data set. There are two approaches:

Bagging: If a subset is selected by sampling the training data set uniformly and with replacement, the method is called bagging (from **b**ootstrap **agg**regati**ng**).

Pasting: If a subset is selected by sampling the training data set uniformly and with no replacement, the method is called pasting.

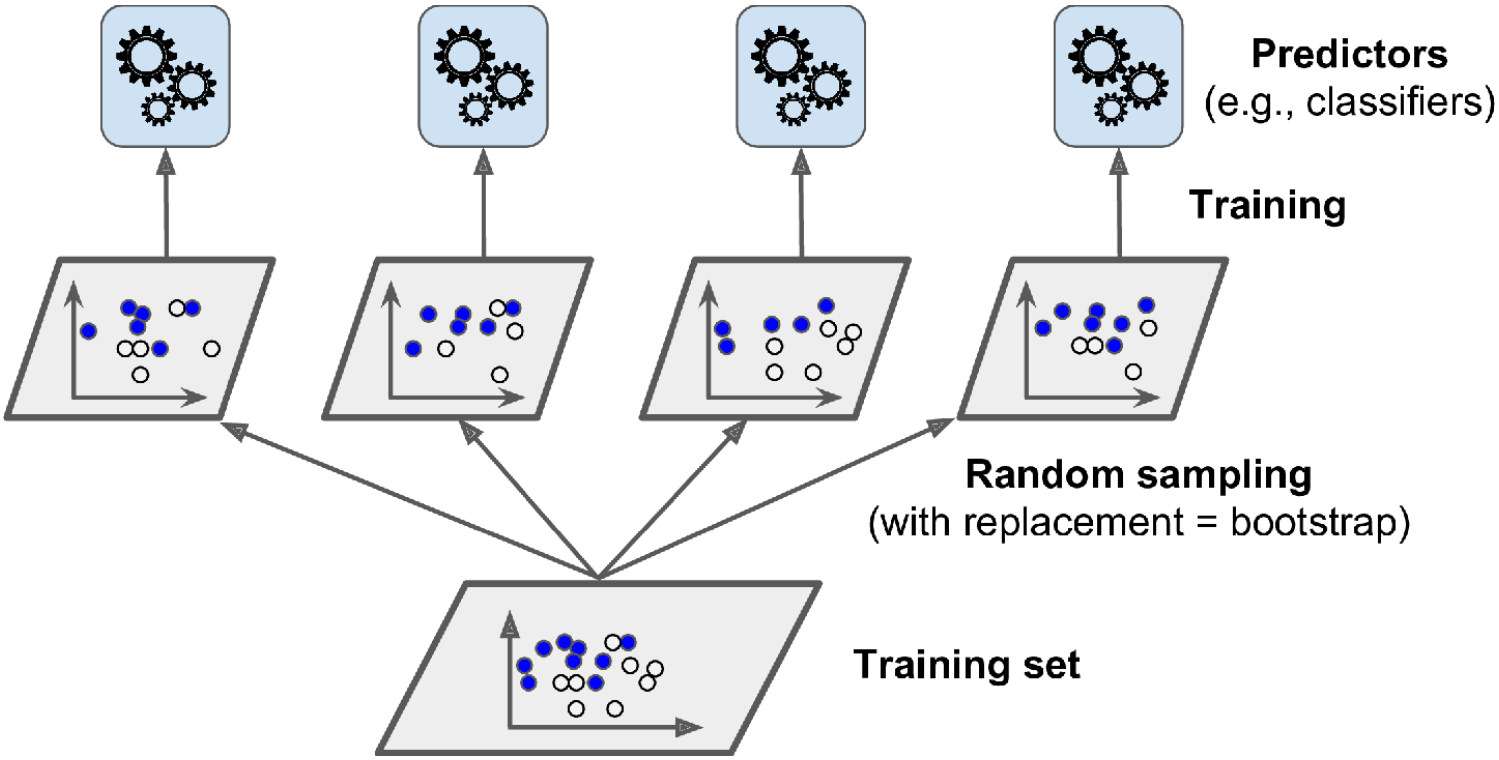


Figure: Bagging and pasting process. Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

As you can see in the above figure, bagging and pasting involve training several predictors on different random samples of the training set.

Typical aggregation functions for this ensemble method (bagging/pasting):

- for classification - most frequent prediction (similar to a hard voting classifier)
- for regression - the average value

Points to consider:

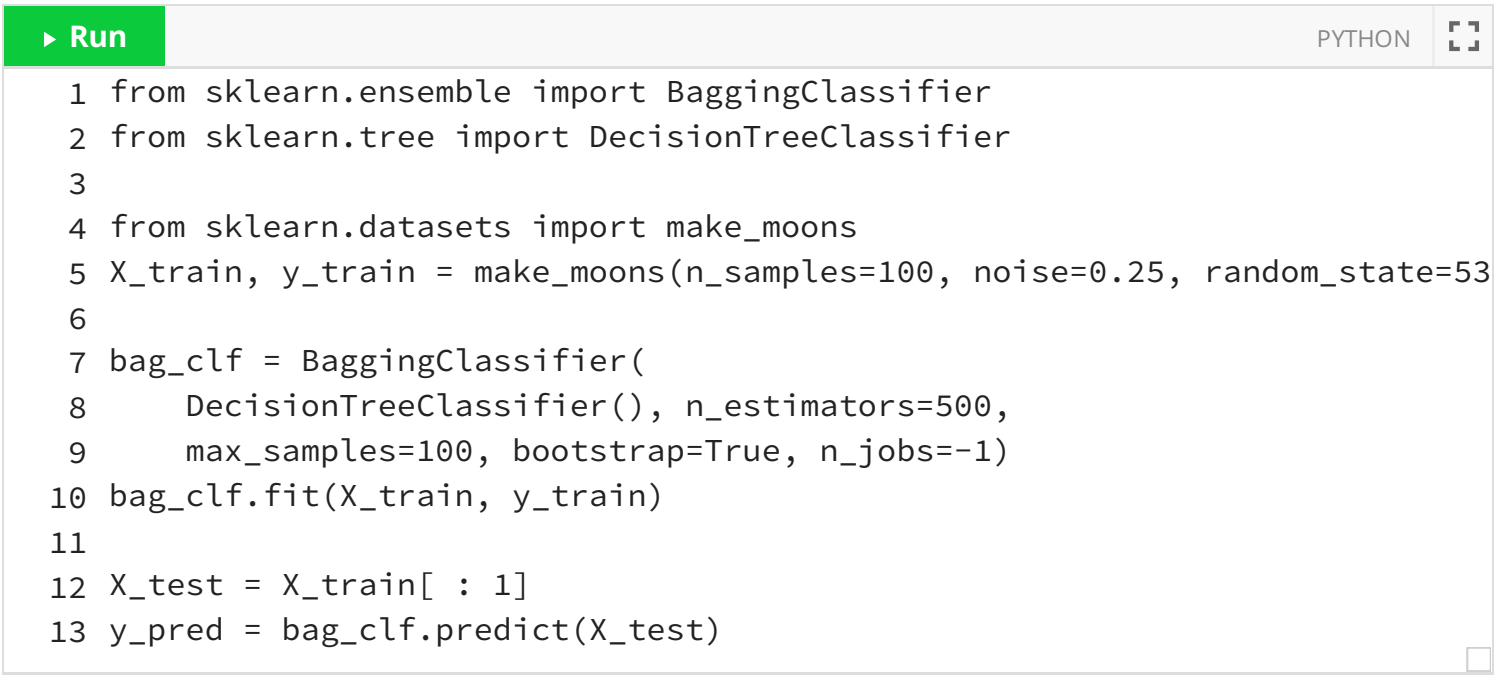
- Bagging/pasting offers lower variance compared to an individual model on the original training

set, and bias remains similar.

- Multiple models can be trained in parallel (multiple CPU cores, multiple servers, etc.), therefore easy to scale bagging/pasting.
- Use cross-validation to select a suitable strategy: bagging or pasting. Generally bagging is preferred over pasting because it results in better models.

Example

The following example is from Chapter 7 of Géron, 2019. Let's build 500 decision tree classifier models on the moons data set, where each model is trained using 100 cases randomly sampled from the training data set with replacement (for bagging).




```
1 from sklearn.ensemble import BaggingClassifier
2 from sklearn.tree import DecisionTreeClassifier
3
4 from sklearn.datasets import make_moons
5 X_train, y_train = make_moons(n_samples=100, noise=0.25, random_state=53
6
7 bag_clf = BaggingClassifier(
8     DecisionTreeClassifier(), n_estimators=500,
9     max_samples=100, bootstrap=True, n_jobs=-1)
10 bag_clf.fit(X_train, y_train)
11
12 X_test = X_train[ : 1]
13 y_pred = bag_clf.predict(X_test)
```

In the above example, set `bootstrap=False` for pasting and `n_jobs= -1` for using all available CPU cores.

The following figure shows behaviour of a single decision tree and the ensemble method (bagging ensemble of 500 decision trees, discussed above). Points to observe:

- small variance: the bagging ensemble offers smoother (less irregular) decision boundary, likely resulting in a better generalisation (better accuracy on test cases)
- similar bias: error rates on the training data set is almost the same in both cases.

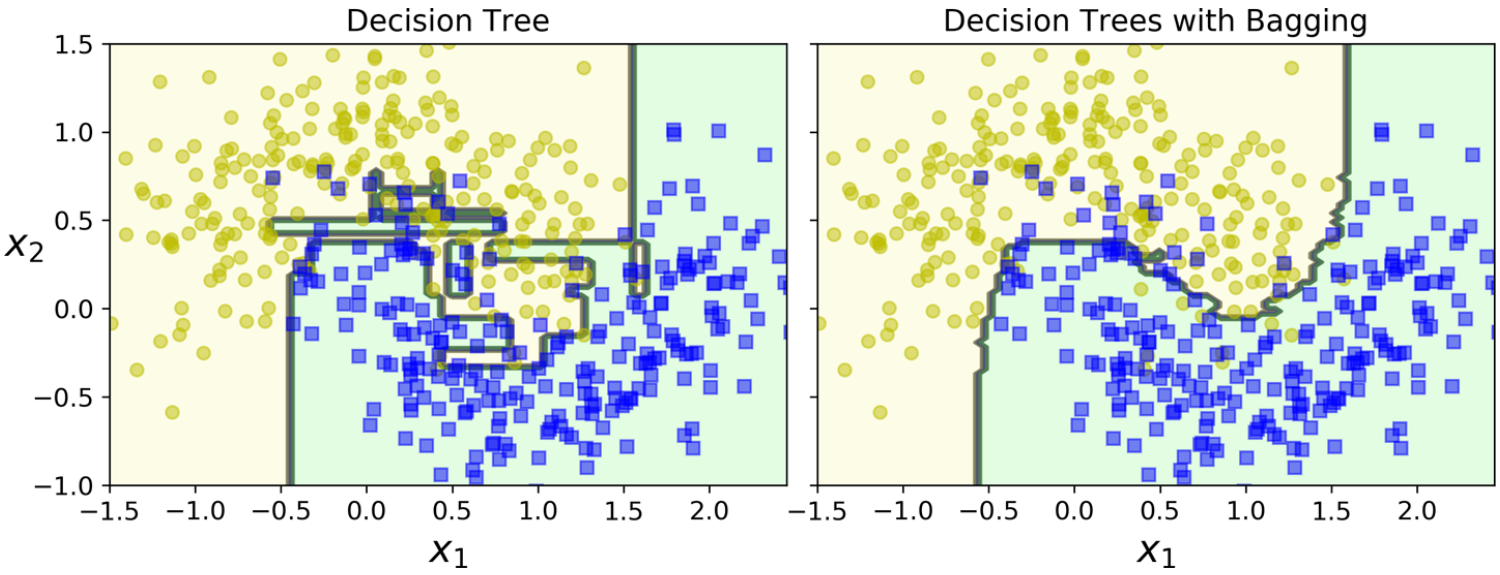


Figure: A single decision tree (left) versus a bagging ensemble of 500 trees (right). Adapted from *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by A. Géron, 2019, Sebastopol; CA: O'Reilly Media.

Out-of-bag evaluation

Considering only a subset of training cases are used for each predictor, we can test the predictor on the remaining training cases (approximately 37% cases for bagging). The remaining training cases that are not sampled are called **out-of-bag (oob)** cases. We can evaluate the ensemble performance by taking the average of oob evaluations of each predictor.

Random patches and random subspaces

We can also sample the input features (attributes), similar to sampling the training cases for bootstrapping described above. The aim is to train each predictor on a random subset of the input features (attributes).

Random patches: method samples both training cases and input features (attributes).

Random subspace: method samples only input features (attributes) and uses all the training cases.

The above approaches offer a higher degree of diversity, essential for a good ensemble method.

Another code example below with Scikit-learn.

Run PYTHON

```python
#Source: https://machinelearningmastery.com/bagging-ensemble-with-python
# evaluate bagging algorithm for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=
# define the model
model = BaggingClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

Code Source: https://machinelearningmastery.com/bagging-ensemble-with-python/

An important hyperparameter for the bagging algorithm is the number of decision trees in the ensemble which is increased until the model performance stabilizes. Bagging of trees known as Random Forest are immune to overfitting the training dataset given the stochastic nature of the learning algorithm.

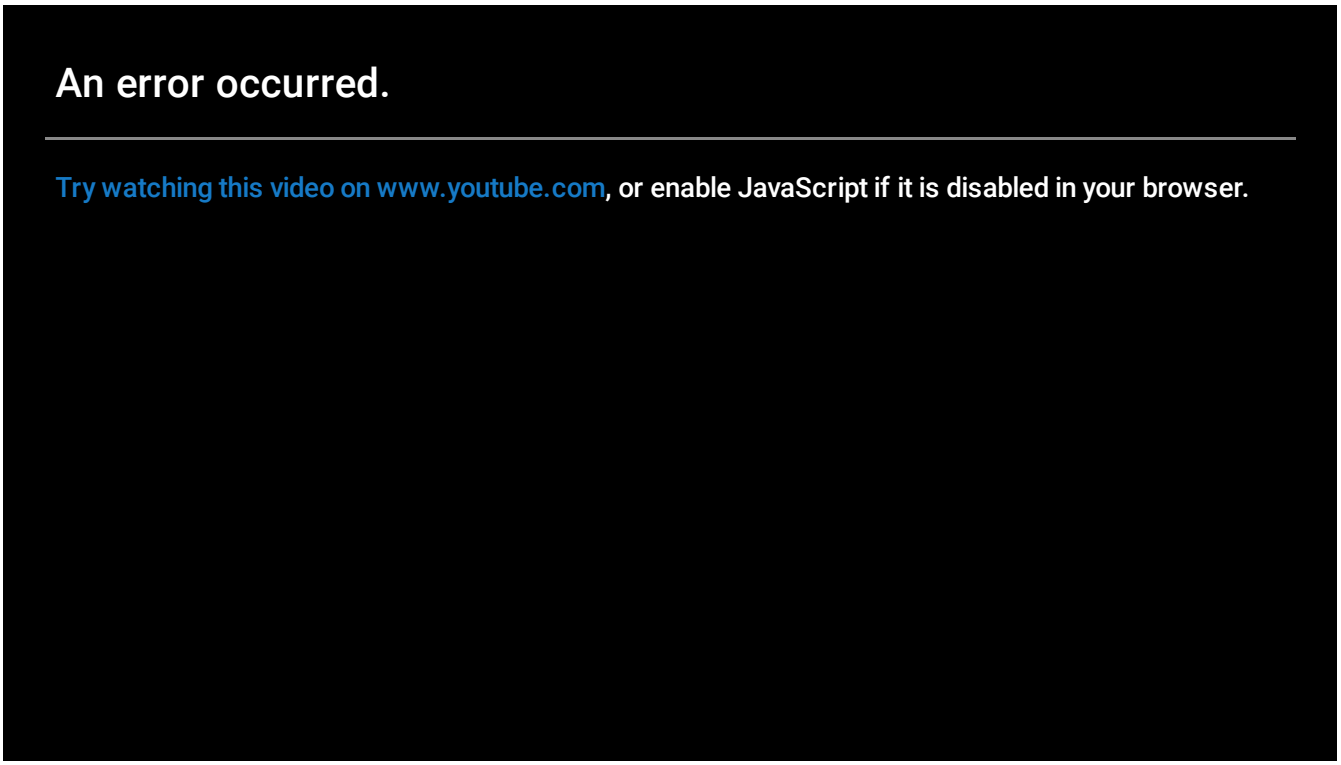
The number of trees can be set via the "*n_estimators*" argument and defaults to 100 and the example below explores the effect of the number of trees with values between 10 to 500.

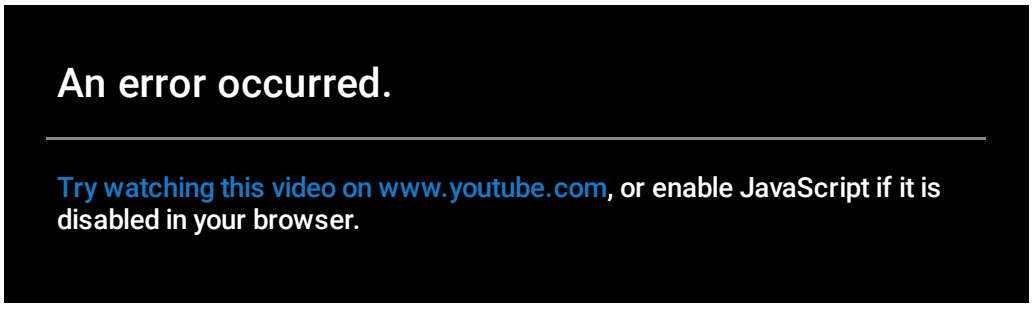
Run PYTHON

```python
#Code Source:  https://machinelearningmastery.com/bagging-ensemble-with-
# explore bagging ensemble number of trees effect on performance
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
from matplotlib import pyplot

# get the dataset
def get_dataset():
    X, y = make_classification(n_samples=1000, n_features=20, n_informat
    return X, y
```

Code Source: https://machinelearningmastery.com/bagging-ensemble-with-python/

Finally, the video below summarizes bagging.





References

1. Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123-140: https://link.springer.com/content/pdf/10.1007/BF00058655.pdf
2. Bühlmann, P., & Yu, B. (2002). Analyzing bagging. *The Annals of Statistics*, *30*(4), 927-961: https://projecteuclid.org/download/pdf_1/euclid.aos/1031689014
3. Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, *36*(1-2), 105-139: https://link.springer.com/content/pdf/10.1023/A:1007515423169.pdf

Random Forest

Random Forest is an ensemble method where each predictor is a decision tree and the predictions are aggregated using the bagging ensemble learning approach. The Random Forest algorithm also introduce further randomness by altering node splitting criteria. In place of using the best feature amongst all available features, they look for the best feature amongst a randomly selected subset of features. The key Random Forest algorithm was proposed by Ho in 1995.

Ho, T. K. (1995, August). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278-282). IEEE.

However, the following is the most popular reference.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32: https://www.cise.ufl.edu/~anand/fa11/Breiman_Random_Forests.pdf

This approach results in greater diversity in the decision tree and generally results in higher bias and lower variance.

Earlier we discussed how to build separate decision trees and combine them using a bagging ensemble method. Scikit-learn offers the specialised class that is optimised for Random Forest ensemble learning, called `RandomForestClassifier`.

Below we provide an overview comparison of Decision Tree with Random Forests.

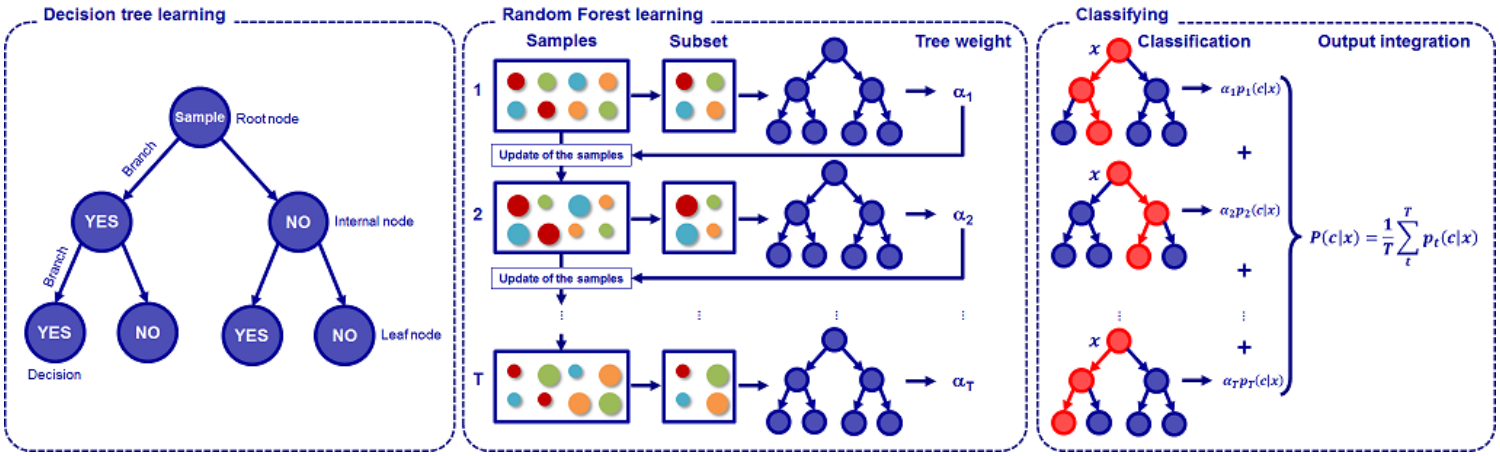


Figure Source. Overview of Decision Tree and Random Forest: http://nanowiki.s2nano.org/index.php/Random_Forest_(RF)

We need highly uncorrelated decision trees that split as randomly as possible in order to get better predictions. Below the figures showcases where the trees are correlated and uncollerated and you

can see the difference in splits due to different feature subsets.

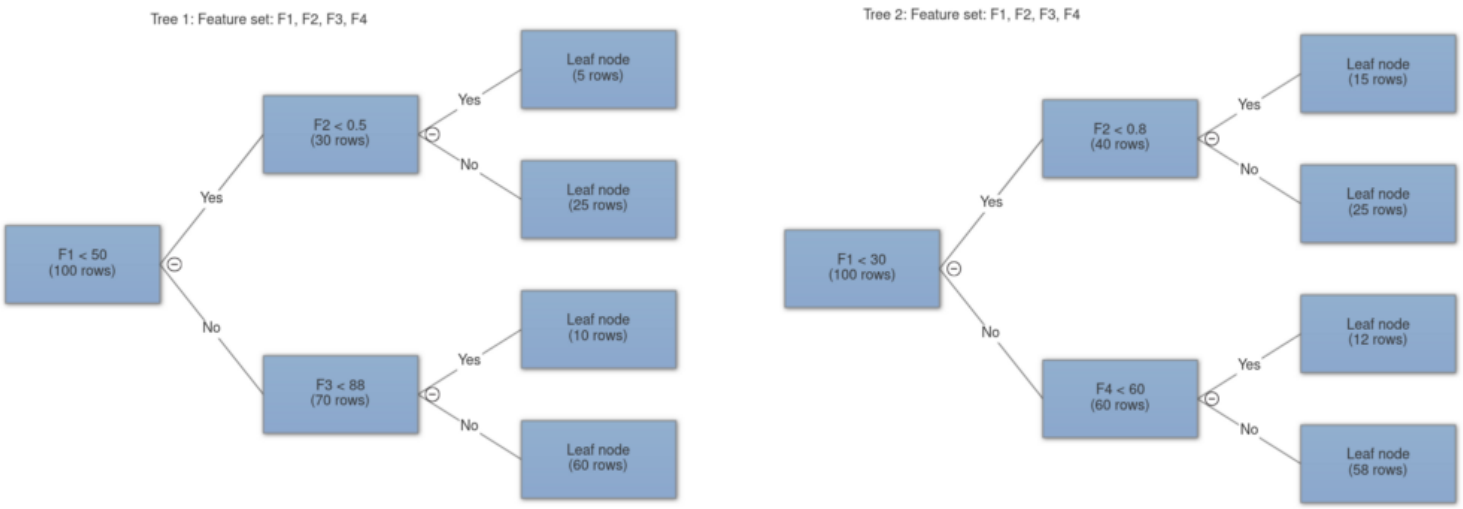


Figure Source. Correlated trees | Both have the same feature set and make similar splits: https://towardsdatascience.com/random-forests-and-decision-trees-from-scratch-in-python-3e4fa5ae4249

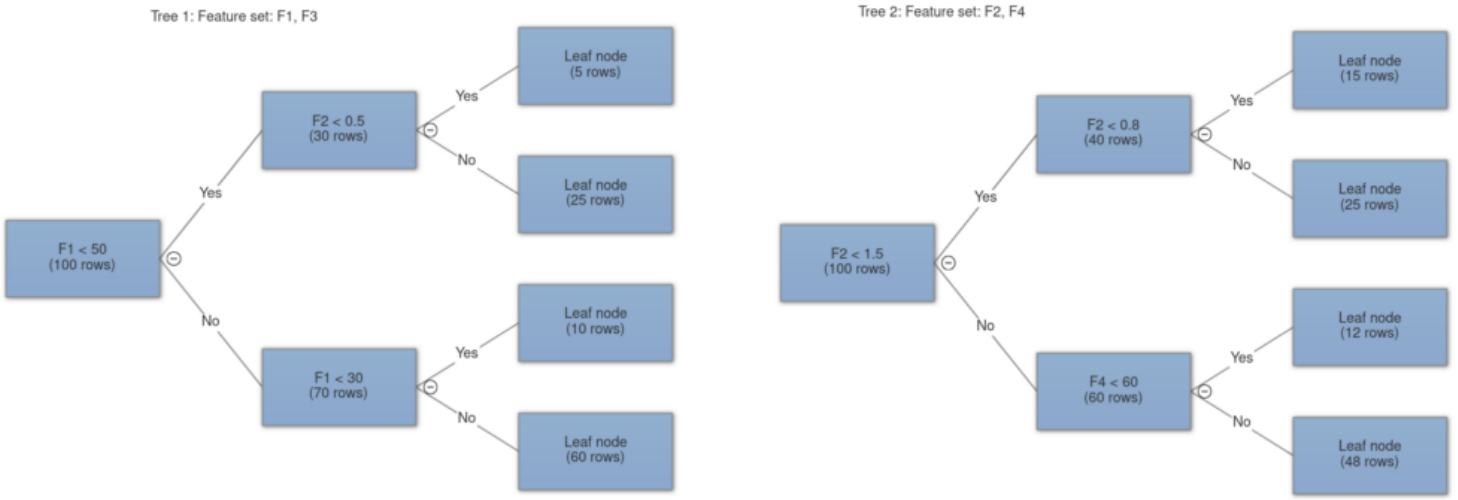


Figure Source: Uncorrelated trees | Both have randomly selected different feature set and make different splits: https://towardsdatascience.com/random-forests-and-decision-trees-from-scratch-in-python-3e4fa5ae4249

Aggregation is the core concept that makes random forests better than decision trees by aggregating uncorrelated trees. The idea is to create several poor model trees (with low depth) and average them out to create a better random forest.

In case of regression, we can average out the prediction of each tree (mean) while in case of classification problems we can simply take the majority of the class voted by each tree (mode). The figure below explains it further.

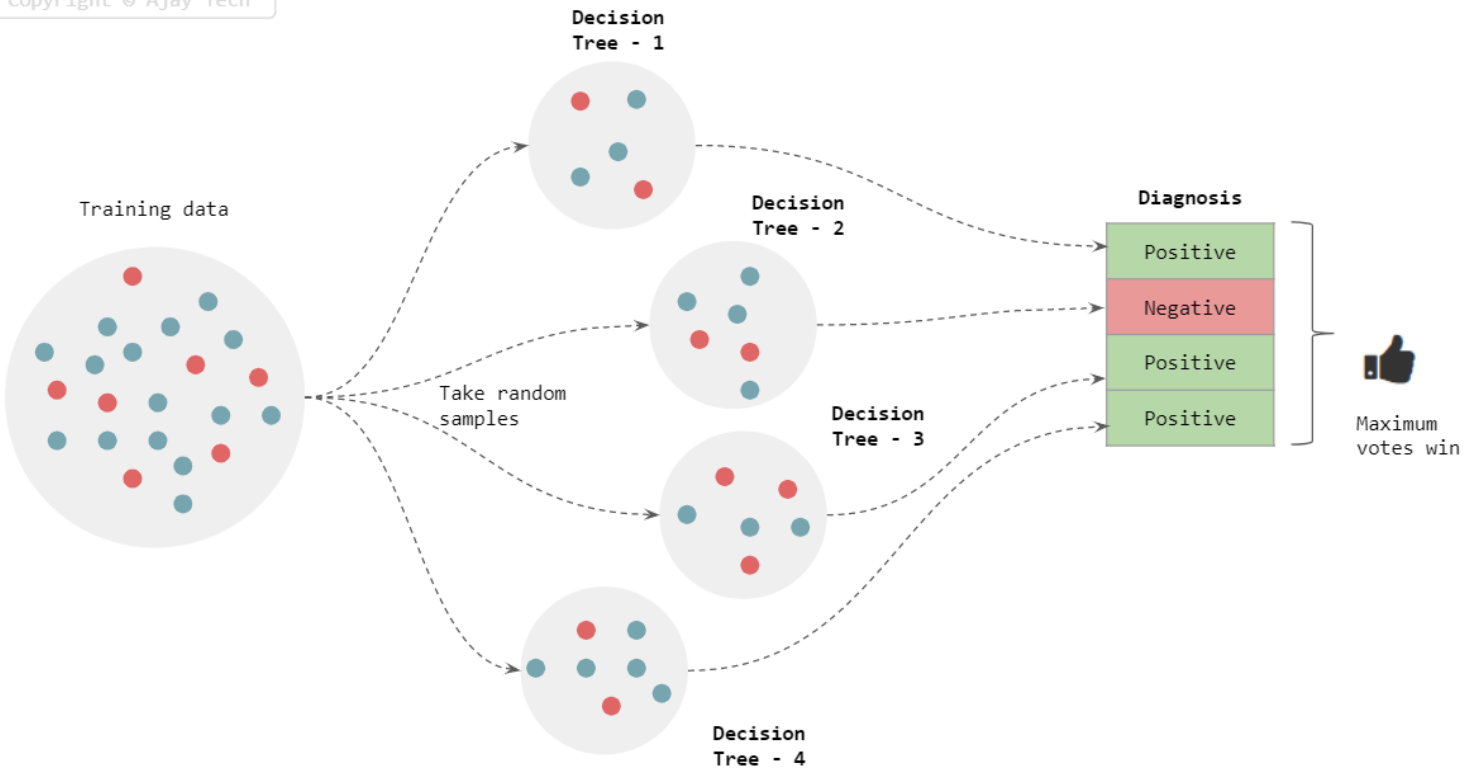


Figure Source: Ensemble Methods: https://ajaytech.co/python-ensemble-methods/

We need to note that the random sub-samples are created by randomly selected certain instances and attributes(features) and the process is highlighted below in three different stages.

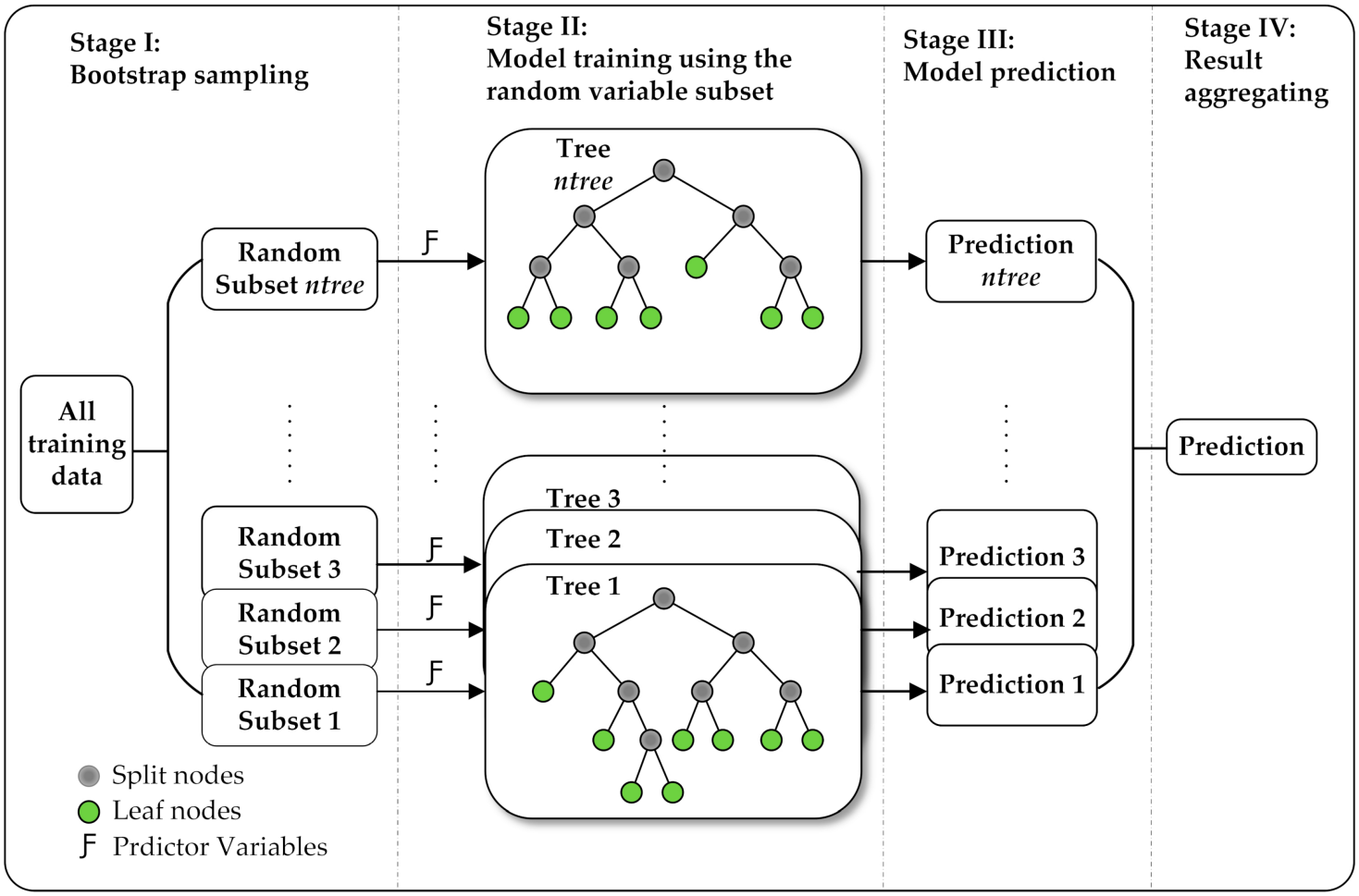


Figure Source: Water Price Prediction for Increasing Market Efficiency Using Random Forest Regression: A Case Study in the Western United States, *Water* **2019**, *11*(2), 228; https://doi.org/10.3390/w11020228

Next we show the pseudo-code that implements Random Forest.

```
1. Given training data set
2. Select number of trees to build (ntrees)
3. for i = 1 to ntrees do
4. |  Generate a bootstrap sample of the original data
5. |  Grow a regression tree to the bootstrapped data
6. |  for each split do
7. |  | Select m variables at random from all p variables
8. |  | Pick the best variable/split-point among the m
9. |  | Split the node into two child nodes
10. |  end
11. | Use typical tree model stopping criteria to determine when a tree
12. end
```

Code Source. Random Forests: https://uc-r.github.io/random_forests

Below is an example of Scikit-learn Random Forest.

▸ Run PYTHON

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import make_moons
X_train, y_train = make_moons(n_samples=100, noise=0.25, random_state=53

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_
rnd_clf.fit(X_train, y_train)

X_test = X_train[0:2]

y_pred_rf = rnd_clf.predict(X_test)
print(y_pred_rf)

```

The above can be achieved using the `BaggingClassifier` as shown below. However as mentioned earlier, it is preferable to use the specialised `RandomForestClassifier` which is optimised for building Random Forests.

PYTHON

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features="auto", max_leaf_nodes=16),
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1)
```

Now we compare Random Forest with Decision Trees.

Run PYTHON

```python
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier

X, y = make_blobs(n_samples=10000, n_features=10, centers=100,random_sta

clf = DecisionTreeClassifier(max_depth=None, min_samples_split=2,random_
scores = cross_val_score(clf, X, y, cv=5)
print(scores.mean(), ' decision trees')

clf = RandomForestClassifier(n_estimators=10, max_depth=None, min_sample
scores = cross_val_score(clf, X, y, cv=5)
```

Code Source: https://scikit-learn.org/stable/modules/ensemble.html

Extremely Randomised Trees ensemble (Extra-Trees)

This approach introduces further randomness by using random thresholds for each feature, avoiding the expensive search for the best possible thresholds. This results in a significant improvement in the training speed. We need to use cross-validation to decide whether Extra-Trees improves prediction accuracy over Random Forest or not.

Extra Tree classifier provides parallel computing features as shown in code below.
This example employs trees to evaluate the impurity-based importance of the pixels in an image classification task (faces). The hotter the pixel, the more important the features.

The code below also illustrates how the construction and the computation of the predictions can be parallelized within multiple jobs.

Try n_jobs = 4 and 1.

Run PYTHON

```python
from time import time
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_olivetti_faces
from sklearn.ensemble import ExtraTreesClassifier

# Number of cores to use to perform parallel fitting of the forest model
n_jobs = 1

# Load the faces dataset
data = fetch_olivetti_faces()
X, y = data.data, data.target

mask = y < 5  # Limit to 5 classes
```

Feature importance

Random Forest ensemble method offers very useful insights into the importance of each feature (attribute) in building multiple decision tree models. All nodes with a given feature are examined to calculate the weighted average of reduction in impurity. Here, the number of training cases at a node represents the weight for that node. Higher impurity reduction indicates that the feature is more important.

Random Forests are often used to quickly rank features based on their importance and are frequently used for feature selection.

Example below for feature importance on an artificial classification task where the red bars are the impurity-based feature importances of the forest, along with their inter-trees variability. The plot suggests that 3 features are informative, while the remaining are not.

Run PYTHON

```python
#Source: https://scikit-learn.org/stable/auto_examples/ensemble/plot_for

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000,
                           n_features=10,
                           n_informative=3,
                           n_redundant=0,
                           n_repeated=0,
```

Code Source: https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

MNIST Dataset results

Data details: https://en.wikipedia.org/wiki/MNIST_database

- [x] Random Forest classifier on the MNIST data set: http://www.cs.virginia.edu/~tjt7a/demos/demos.html

- [x] MNIST Results when compared to deep neural networks: https://rstudio-pubs-static.s3.amazonaws.com/284590_22eecf60de804fd6aa2c9627615cba49.html

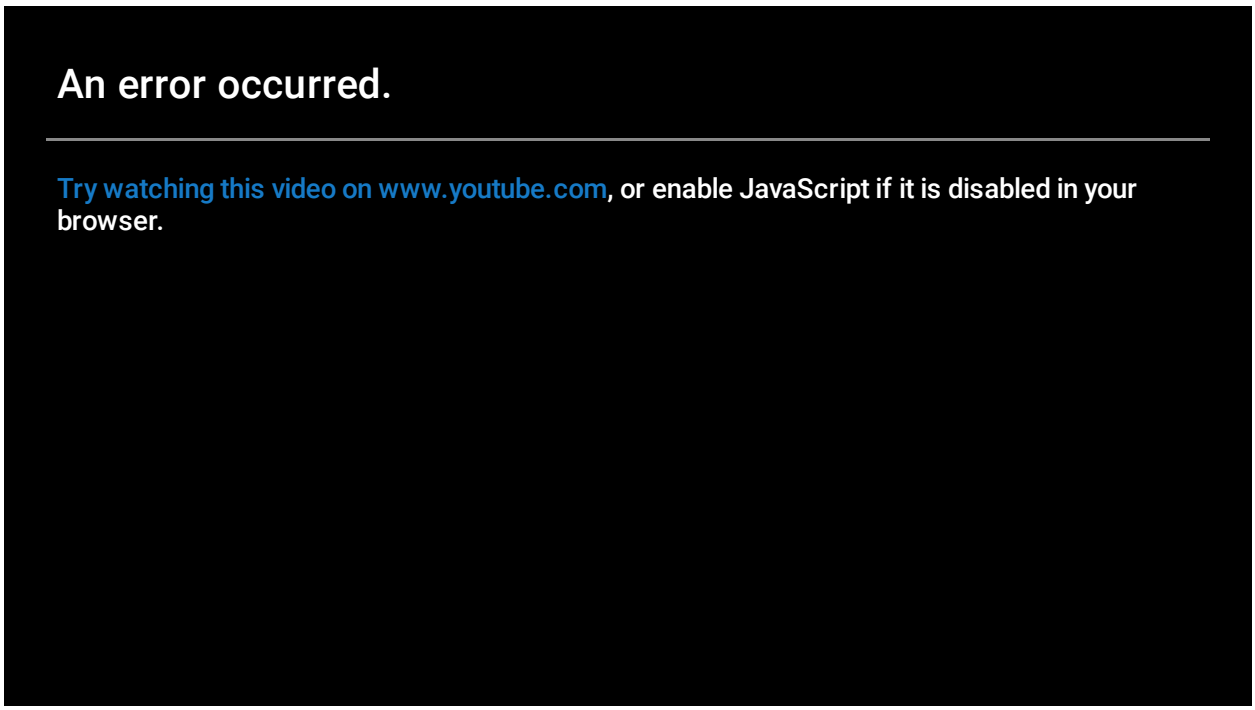
R Randomforest code

https://datascienceplus.com/random-forests-in-r/

Run

```r
#Source: https://machinelearningmastery.com/machine-learning-ensembles-w

# Load libraries
library(mlbench)
library(caret)
library(caretEnsemble)

# Load the dataset
data(Ionosphere)
dataset <- Ionosphere
dataset <- dataset[,-2]
dataset$V1 <- as.numeric(as.character(dataset$V1))

head(dataset)
```

Here are some videos that summarize Random Forests.



References

1. Ho, T. K. (1995, August). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278-282). IEEE https://ieeexplore.ieee.org/abstract/document/598994
2. Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32: https://www.cise.ufl.edu/~anand/fa11/Breiman_Random_Forests.pdf
3. Rodriguez, J. J., Kuncheva, L. I., & Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, *28*(10), 1619-1630.
4. Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R news*, *2*(3), 18-22
5. Strobl, C., Boulesteix, A. L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics*, *8*(1), 25: https://link.springer.com/article/10.1186/1471-2105-8-25
6. Oshiro, T. M., Perez, P. S., & Baranauskas, J. A. (2012, July). How many trees in a random forest?. In *International workshop on machine learning and data mining in pattern recognition* (pp. 154-168). Springer: https://www.researchgate.net/profile/Jose_Baranauskas/publication/230766603_How_Many_Trees_in_a_Random_Forest/links/0912f5040fb35357a1000000/How-Many-Trees-in-a-Random-Forest.pdf
7. Belgiu, M., & Drăguţ, L. (2016). Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, *114*, 24-31: https://doi.org/10.1016/j.isprsjprs.2016.01.011
8. Fawagreh, K., Gaber, M. M., & Elyan, E. (2014). Random forests: from early developments to recent advancements. *Systems Science & Control Engineering: An Open Access Journal*, *2*(1), 602-609: https://doi.org/10.1080/21642583.2014.956265

Code

Implementing Random Forest from Scratch in Python:
https://machinelearningmastery.com/implement-random-forest-scratch-python/

Random Forest

Apply Random Forest to the given scenario using Python.

Grow a forest by following these steps:

- Continuing the previous exercise, generate 1,000 subsets of the training set, each containing 100 instances selected randomly. Hint: you can use scikit-learn's `ShuffleSplit` class for this.
- Train one Decision Tree on each subset, using the best hyperparameter values found in the previous exercise. Evaluate these 1,000 Decision Trees on the test set. Since they were trained on smaller sets, these Decision Trees will likely perform worse than the first Decision Tree, achieving only about 80% accuracy.
- Now comes the magic. For each test set instance, generate the predictions of the 1,000 Decision Trees, and keep only the most frequent prediction (you can use SciPy's `mode()` function for this). This approach gives you **majority-vote predictions** over the test set.
- Evaluate these predictions on the test set: you should obtain a slightly higher accuracy than your first model (about 0.5 to 1.5% higher). Congratulations, you have trained a Random Forest classifier!

(Exercise 8, in Chapter 6 of Géron, 2019)

Exercise 2

Taking the same datasets selected from Exercise 1, apply and see how Random Forests improve the results.