Decision Trees

APT 3025: APPLIED MACHINE LEARNING

Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

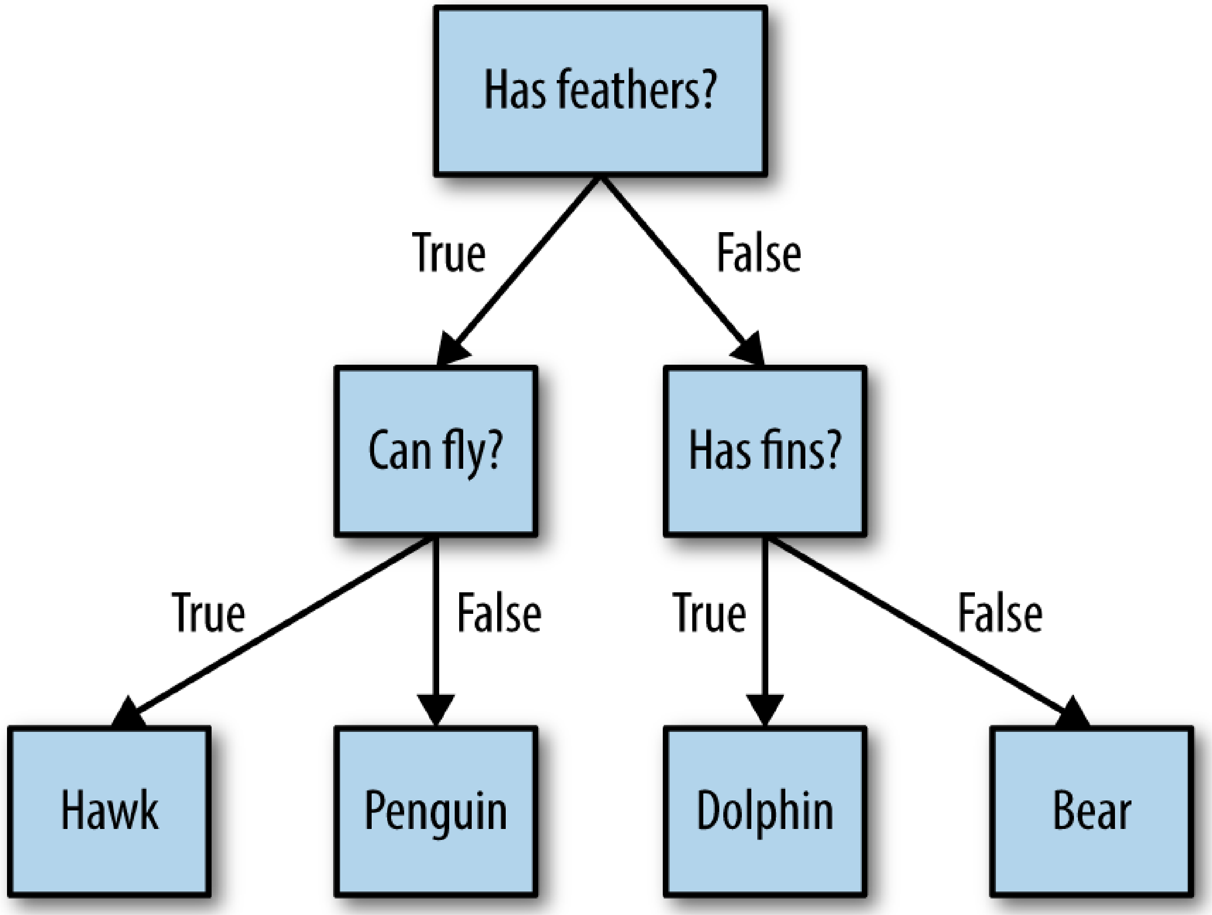
Lecture Overview

- What are decision trees?
- Decision tree learning (informally)
- Controlling complexity
- Classification example

Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

What is a decision tree?

- Decision trees are widely used models for classification and regression tasks.
- Essentially, they learn a hierarchy of if/else questions, leading to a decision.
- Here is a decision tree to distinguish animals.



Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

What is a decision tree?

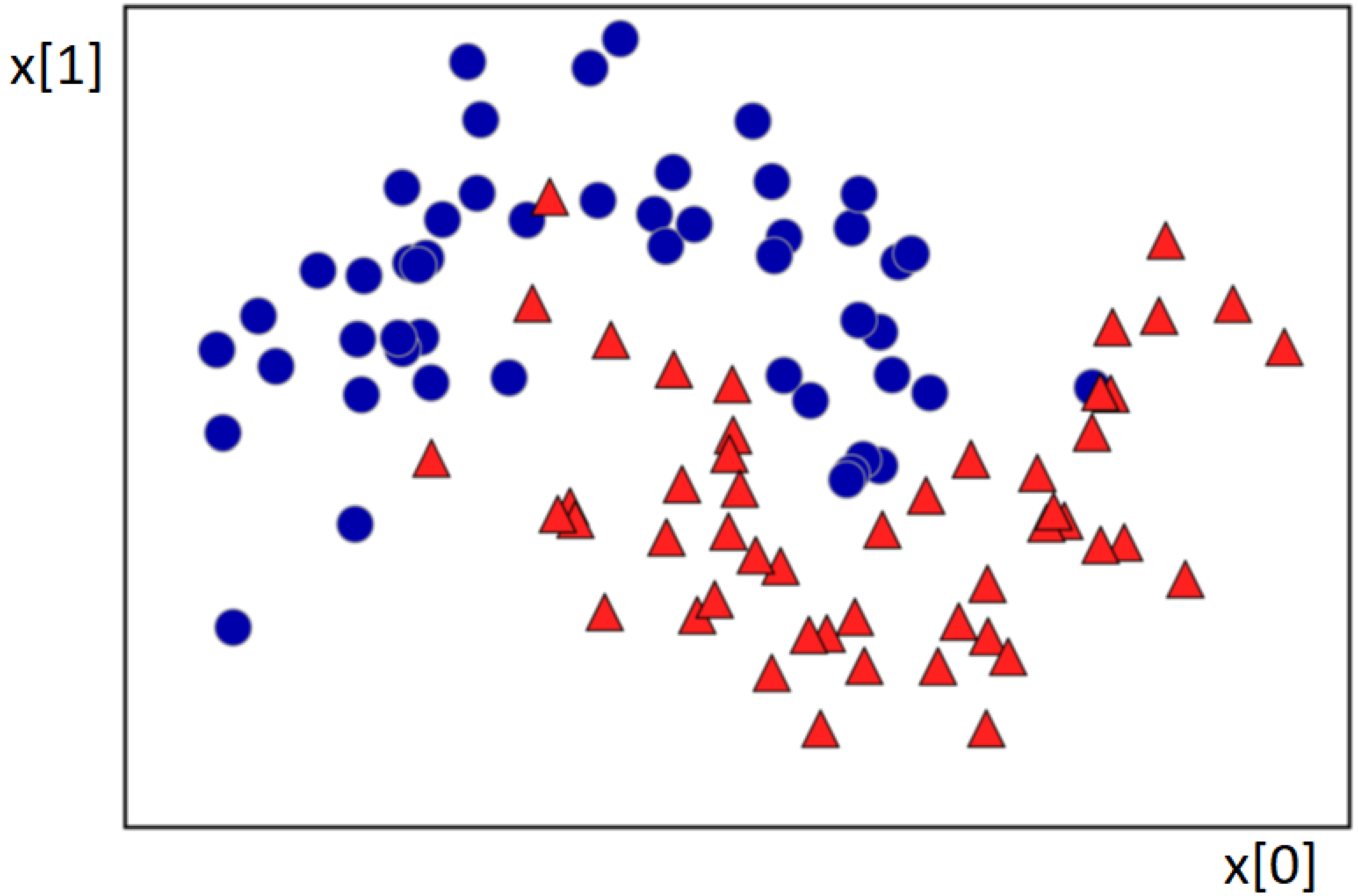
- In this illustration, each node in the tree either represents a question or a terminal node (also called a leaf) that contains the answer.
- The edges connect the answers to a question with the next question you would ask.
- We have just built a model to distinguish between four classes of animals (hawks, penguins, dolphins, and bears) using the three features “has feathers,” “can fly,” and “has fins.”
- Instead of building these models by hand, we can learn them from data using supervised learning.

Learning a Decision Tree

- Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.
- In the machine learning setting, these questions are called *tests*.
- Usually data does not come in the form of binary yes/no features as in the animal example, but is instead represented as continuous features such as in the 2D dataset shown below.
- The tests that are used on continuous data are of the form "Is feature i larger than value a?"

Two moons dataset

- The dataset consists of two half-moon shapes, with each class consisting of 75 data points.
- We will refer to this dataset as two_moons.

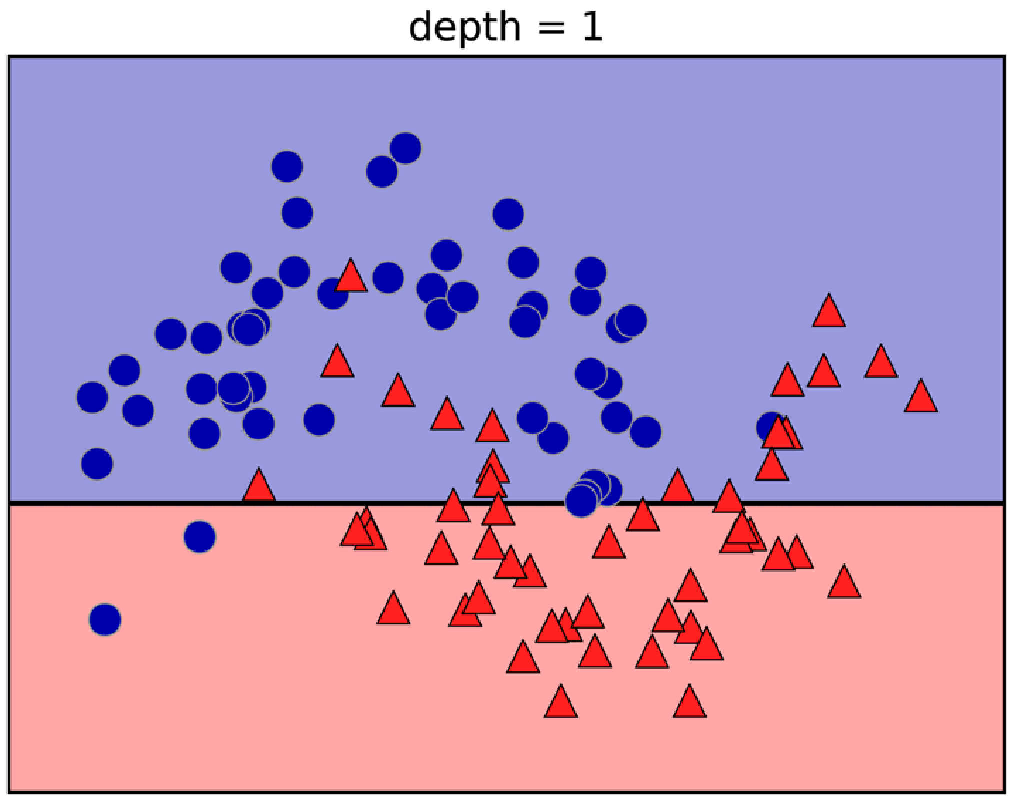


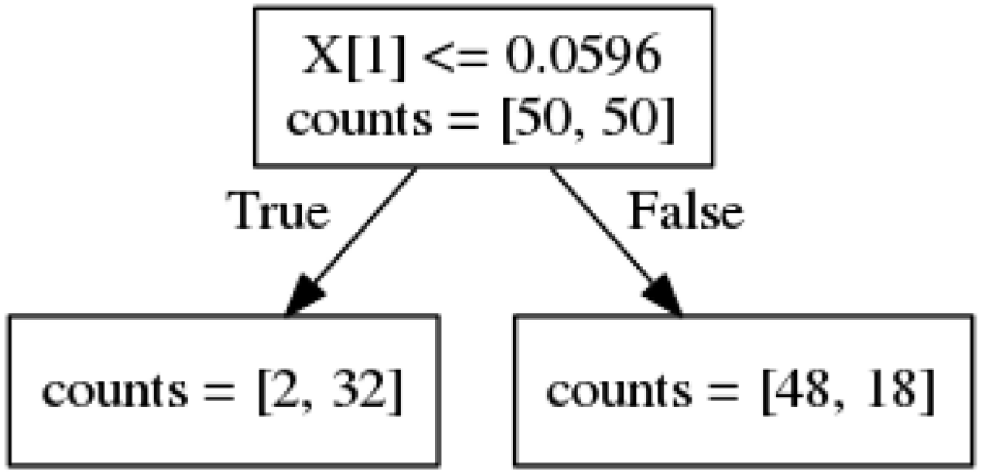
Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

Building a tree from data

- To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.
- The figure below shows the first test that is picked.
- Splitting the dataset horizontally at x[1]=0.0596 yields the most information; it best separates the points in class 0 from the points in class 1.

Decision boundary of tree with depth 1 (left) and corresponding tree (right)





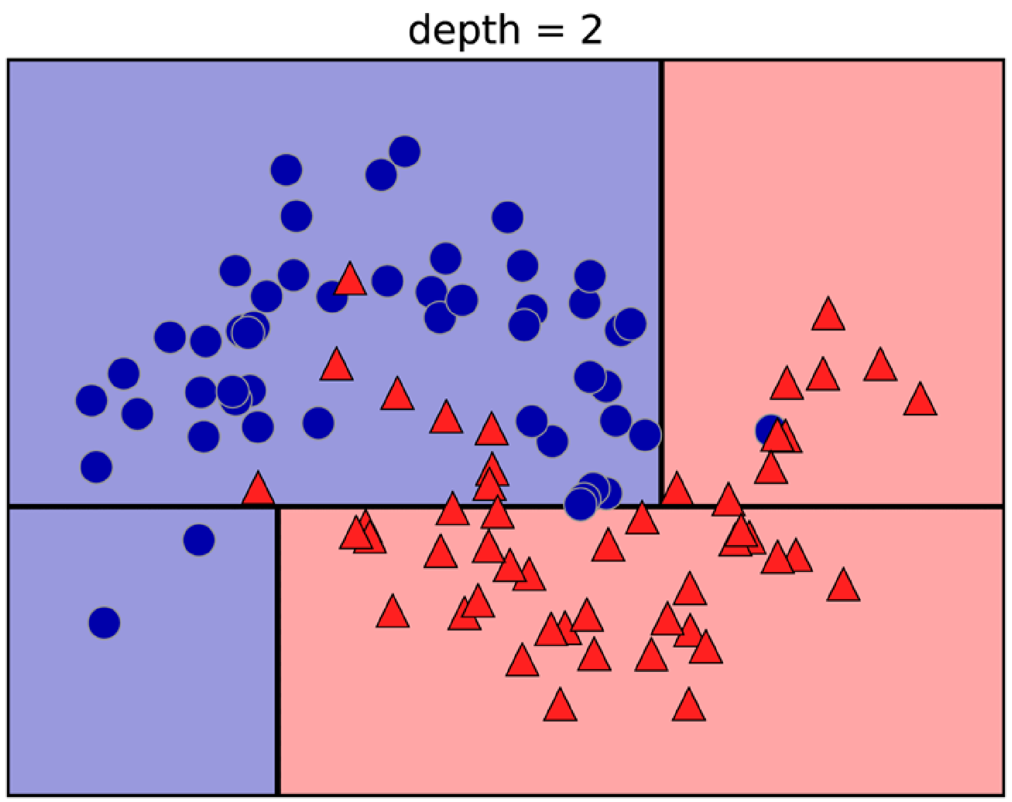
Building a tree from data

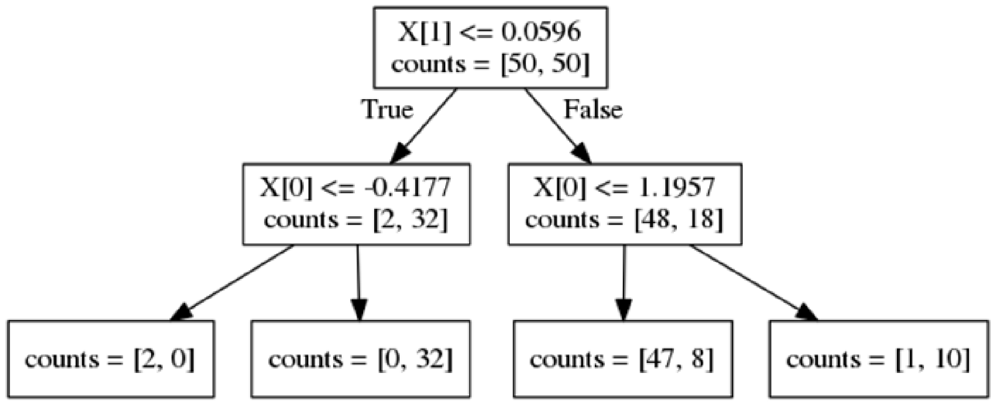
- The top node, also called the root, represents the whole dataset, consisting of 50 points belonging to class 0 and 50 points belonging to class 1.
- The split is done by testing whether x[1] <= 0.0596, indicated by a black line.
- If the test is true, a point is assigned to the left node, which contains 2 points belonging to class 0 and 32 points belonging to class 1.
- Otherwise the point is assigned to the right node, which contains 48 points belonging to class 0 and 18 points belonging to class 1.

Building a tree from data

- Even though the first split did a good job of separating the two classes, the bottom region still contains points belonging to class 0, and the top region still contains points belonging to class 1.
- We can build a more accurate model by repeating the process of looking for the best test in both regions.
- The figure below shows that the most informative next split for the left and the right region is based on x[0]

Decision boundary of tree with depth 2 (left) and corresponding decision tree (right)





Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

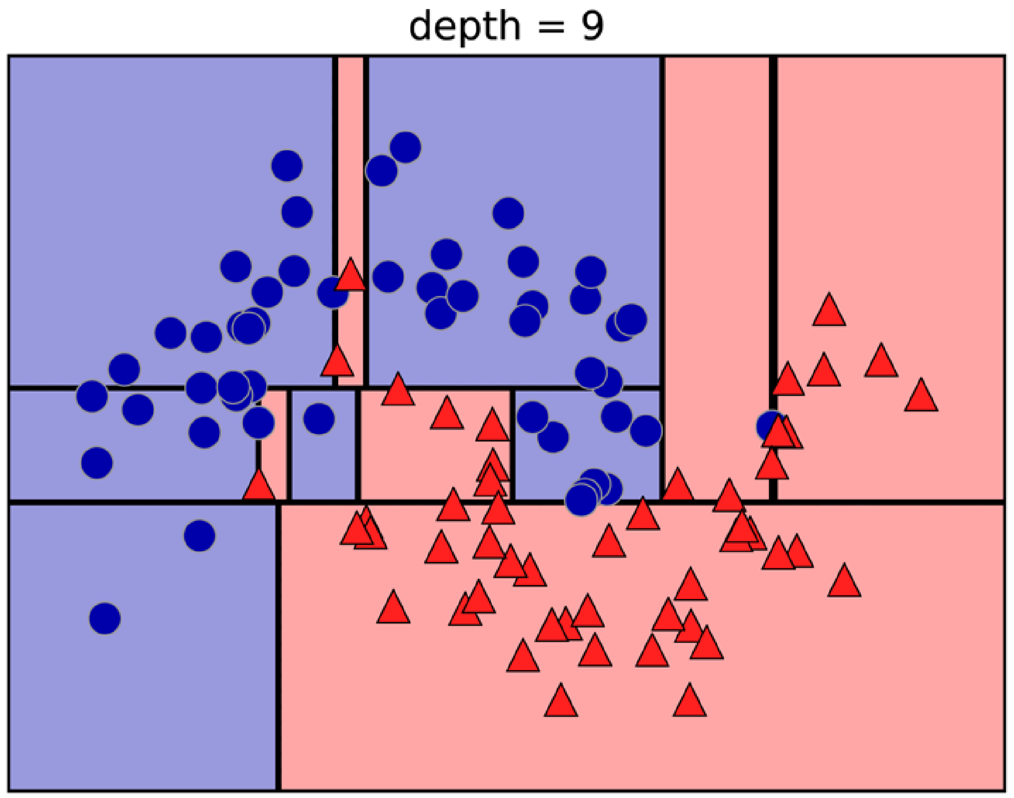
Decision tree learning

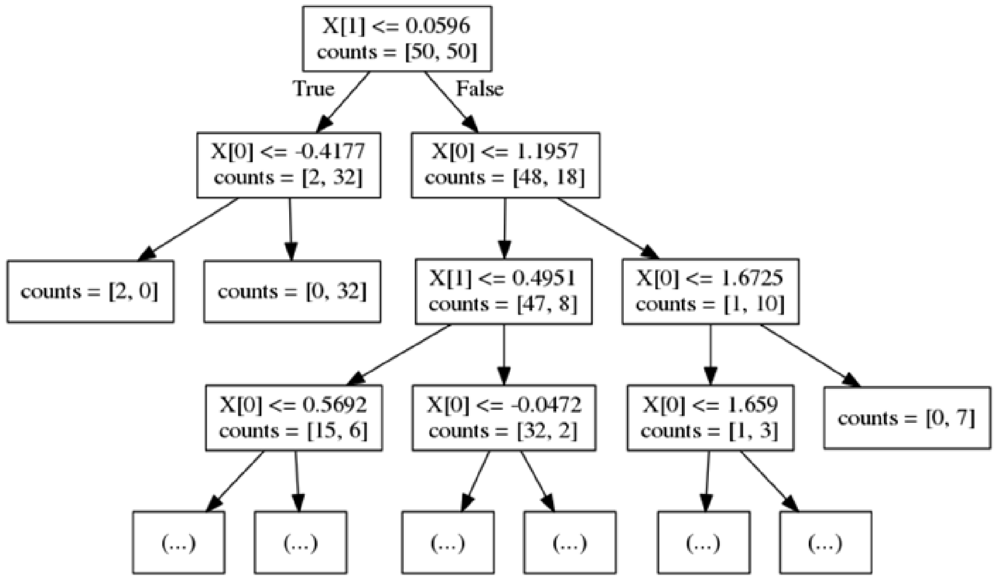
- This recursive process yields a binary tree of decisions, with each node containing a test.
- Alternatively, you can think of each test as splitting the part of the data that is currently being considered along one axis.
- This yields a view of the algorithm as building a hierarchical partition.
- As each test concerns only a single feature, the regions in the resulting partition always have axis-parallel boundaries.

Decision tree learning

- The recursive partitioning of the data is repeated until each region in the partition (each leaf in the decision tree) only contains a single target value (a single class or a single regression value).
- A leaf of the tree that contains data points that all share the same target value is called pure.
- The final partitioning for this dataset is shown in the figure below.

Final tree





Decision boundary of tree with depth 9 (left) and part of the corresponding tree (right); the full tree is quite large and hard to visualize

Prediction on new data

- A prediction on a new data point is made by checking which region of the partition of the feature space the point lies in, and then predicting the majority target (or the single target in the case of pure leaves) in that region.
- The region can be found by traversing the tree from the root and going left or right, depending on whether the test is fulfilled or not.

Using decision trees for regression

- It is also possible to use trees for regression tasks, using exactly the same technique.
- To make a prediction, we traverse the tree based on the tests in each node and find the leaf the new data point falls into.
- The output for this data point is the mean target of the training points in this leaf.

Controlling complexity

- Typically, building a tree as described here and continuing until all leaves are pure leads to models that are very complex and highly overfit to the training data.
- The presence of pure leaves mean that a tree is 100% accurate on the training set; each data point in the training set is in a leaf that has the correct majority class.
- The overfitting can be seen on the final tree shown previously.

Controlling complexity

- You can see the regions determined to belong to class 1 in the middle of all the points belonging to class 0.
- On the other hand, there is a small strip predicted as class 0 around the point belonging to class 1 to the very right.
- This is not how one would imagine the decision boundary to look, and the decision boundary focuses a lot on single outlier points that are far away from the other points in that class.

Pruning

- There are two common strategies to prevent overfitting:
    - stopping the creation of the tree early (also called pre-pruning), or
    - building the tree but then removing or collapsing nodes that contain little information (also called post-pruning or just pruning)
- Possible criteria for pre-pruning include
    - limiting the maximum depth of the tree,
    - limiting the maximum number of leaves, or
    - requiring a minimum number of points in a node to keep splitting it.

Classification Example

- Let’s look at the effect of pre-pruning in more detail on the Breast Cancer dataset.
- As always, we import the dataset and split it into a training and a test part.
- Then we build a model using the default setting of fully developing the tree (growing the tree until all leaves are pure).

Classification Example

```python
from sklearn.tree import DecisionTreeClassifier

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.937
```

Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

Overfitting

- As expected, the accuracy on the training set is 100%—because the leaves are pure, the tree was grown deep enough that it could perfectly memorize all the labels on the training data.
- If we don’t restrict the depth of a decision tree, the tree can become arbitrarily deep and complex.
- Unpruned trees are therefore prone to overfitting and not generalizing well to new data.

Applying Pruning

- Now let’s apply pre-pruning to the tree, which will stop developing the tree before we perfectly fit to the training data.
- One option is to stop building the tree after a certain depth has been reached.
- Here we set max_depth=4, meaning only four consecutive questions can be asked.
- Limiting the depth of the tree decreases overfitting. This leads to a lower accuracy on the training set, but an improvement on the test set.

Pruning by limiting depth

```python
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

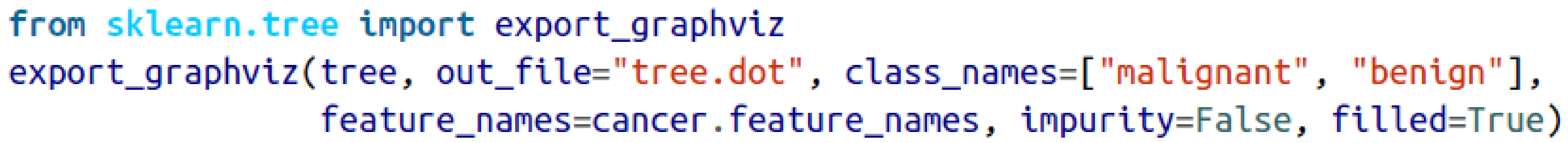
```
Accuracy on training set: 0.988
Accuracy on test set: 0.951
```

Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

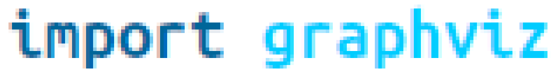
Visualising the decision tree

- We can visualize the tree using the export_graphviz function from the tree module.
- This writes a file in the .dot file format, which is a text file format for storing graphs.
- We set an option to color the nodes to reflect the majority class in each node and pass the class and feature names so the tree can be properly labeled

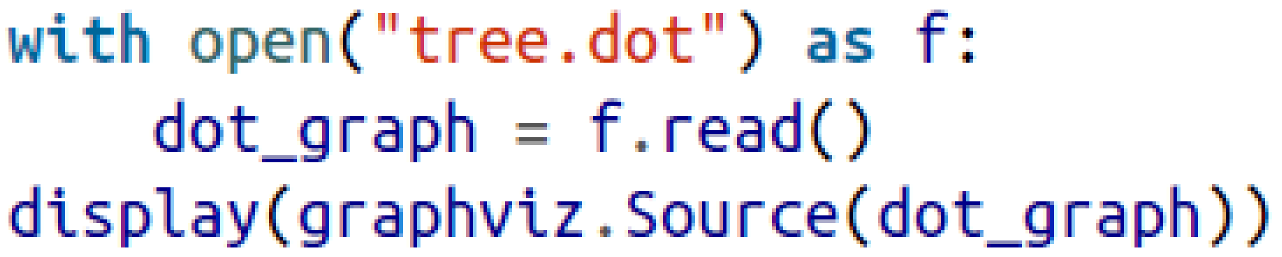
Visualising the decision tree



```python
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["malignant", "benign"],
                feature_names=cancer.feature_names, impurity=False, filled=True)
```



```python
import graphviz
```



```python
with open("tree.dot") as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```

Notes taken from Introduction to Machine Learning with Python by Andreas C. Müller & Sarah Guido, O'Reilley Media, 2016

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- We would like to select the attribute that is most useful for classifying examples.
- What is a good quantitative measure of the worth of an attribute?
- We will define a statistical property, called **information gain**, that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses the information gain measure to select among the candidate attributes at each step while growing the tree.

Entropy

- In order to define information gain precisely, we need to define a measure commonly used in information theory, called **entropy**
- Entropy characterises the impurity of an arbitrary collection of examples.
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this boolean classificaiton is:

Entropy

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

$p_{\oplus}$ is the proportion of positive examples in S

$p_{\ominus}$ is the proportion of negative examples in S

Entopy Example

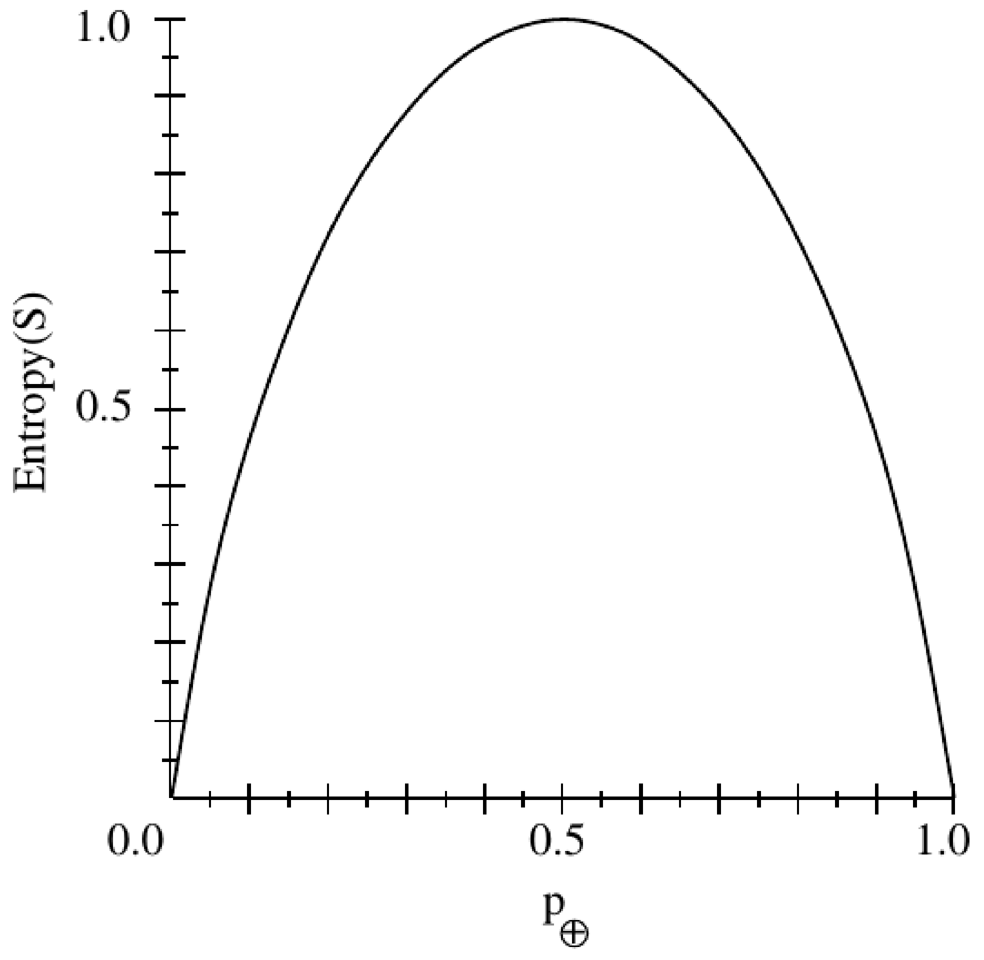
- To illustrate, suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples ([9+, 5-).
- Then the entropy of S relative to this Boolean classification is

$$\text{Entropy}([9+,5-]) = -(9/14)\log(9/14) - (5/14)\log(5/14)$$

$$= 0.940$$

- Entropy is 0 if all members of S belong to the same class
- And it is 1 when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1.

Entropy Curve

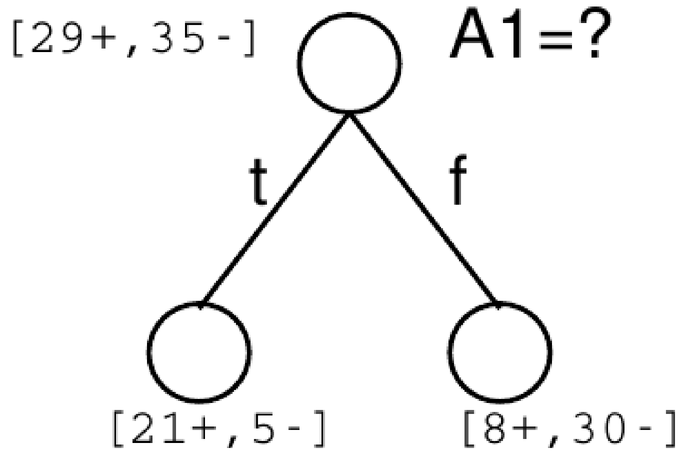


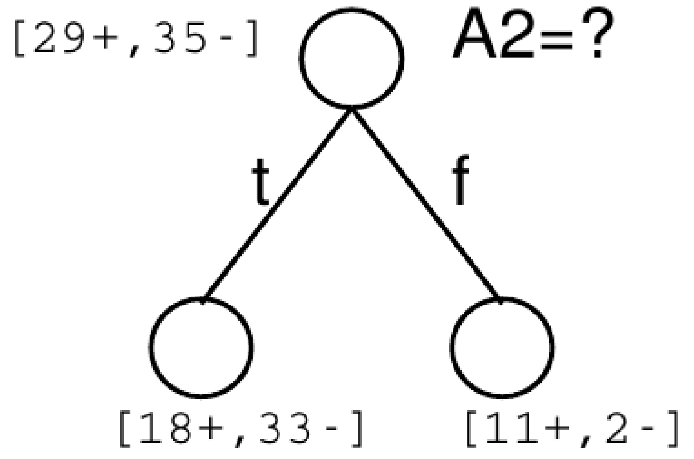
Inforation Gain

- Information gain measures the expected reduction in entropy.
- Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data.
- The measure we will use, called information gain, is simply the expected reduction in entropy caused by partitioning the examples according to this attributed.
- More precisely, the information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as:

Information Gain

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

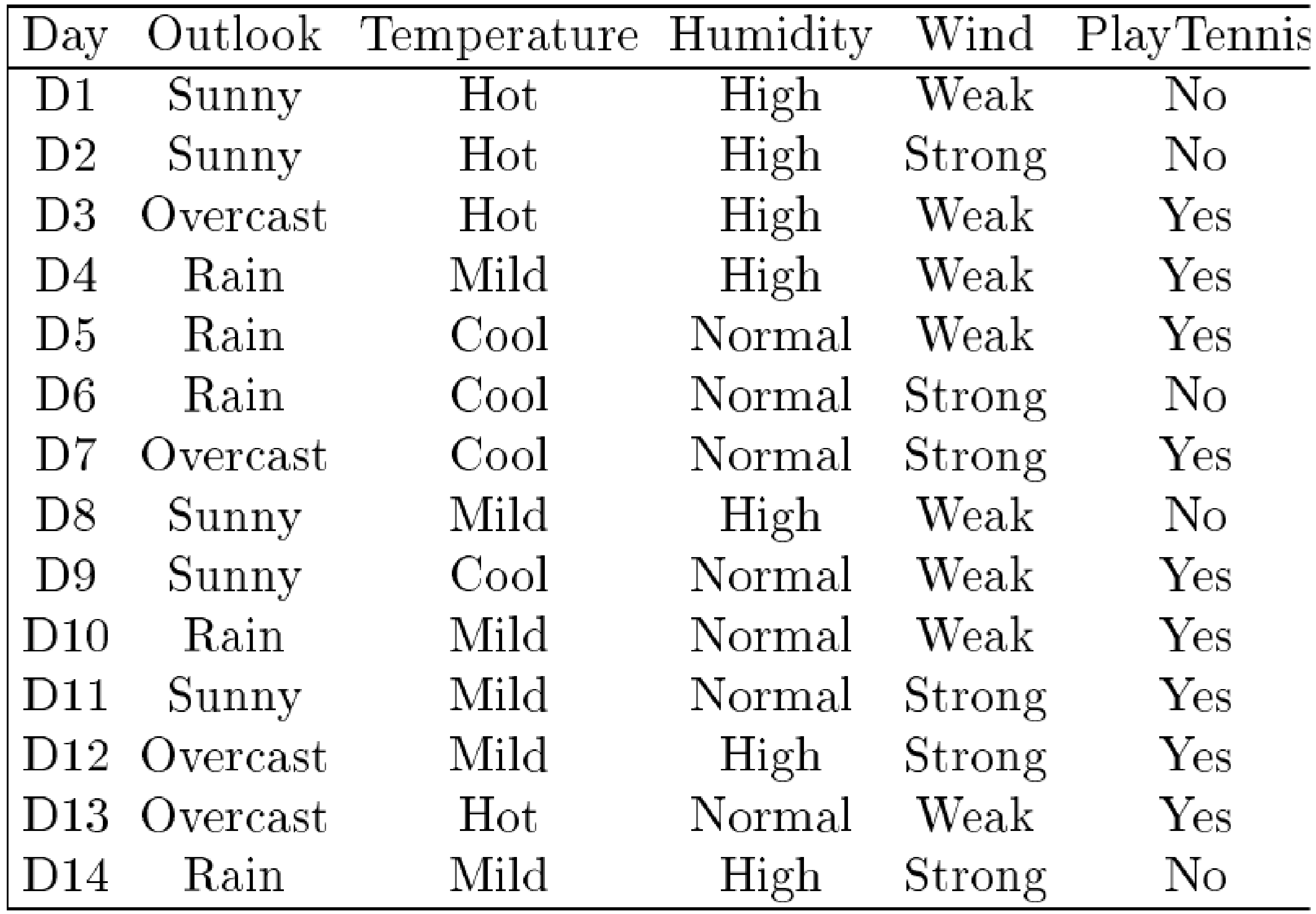




Information Gain

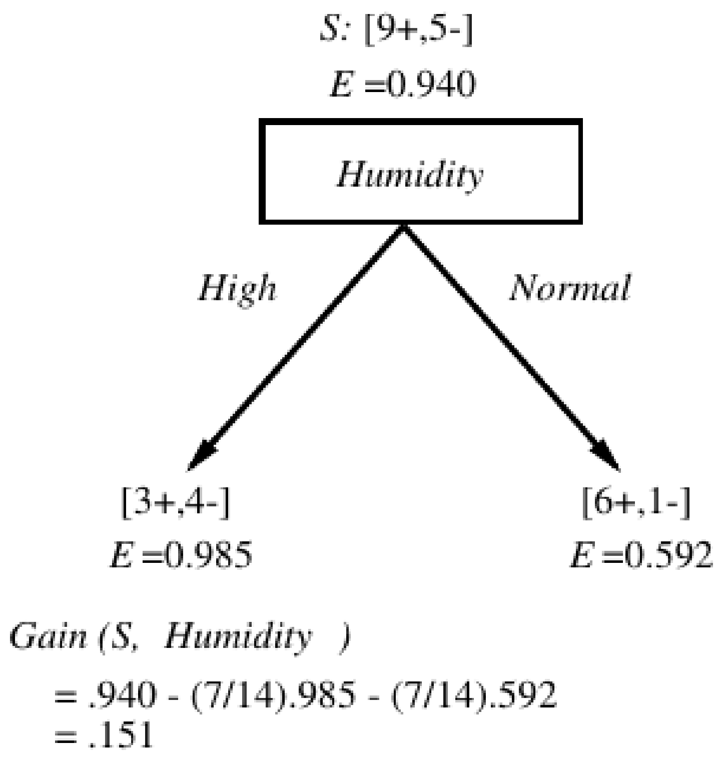
- Values(A) is the set of all possible values for attribute A
- S_v is the subset of S for which attribute A has value v
- The first term in the information gain equation is just the entropy of the entire set S, and the second term is the expected value of the entropy after S is partitioned using attribute A.
- The expected entropy described by this second term is simply the sum of the entropies of each subset S_v, weighted by the fraction of examples $|S_v|/|S|$ that belong to S_v.
- Gain(S,A) is therefore the expected reduction in entropy caused by knowing the value of A.

PlayTennis Example



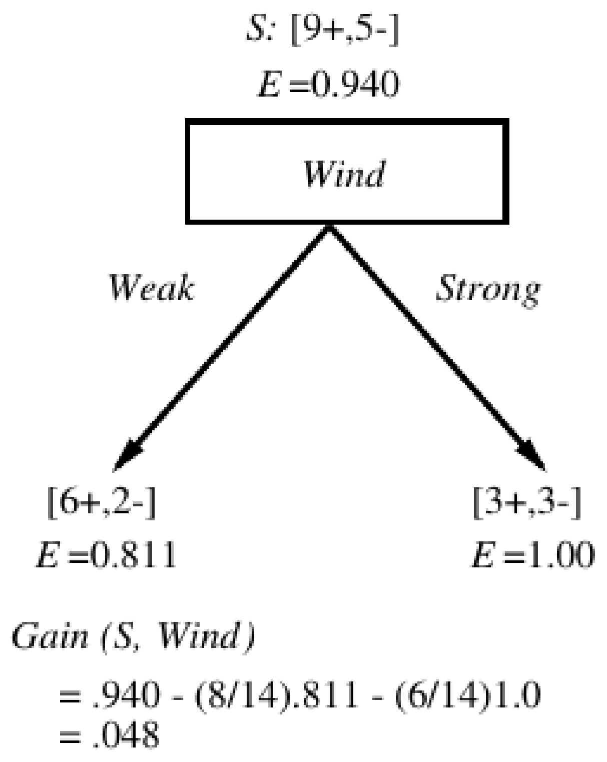
Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Which Attribute is the Best Classifier?



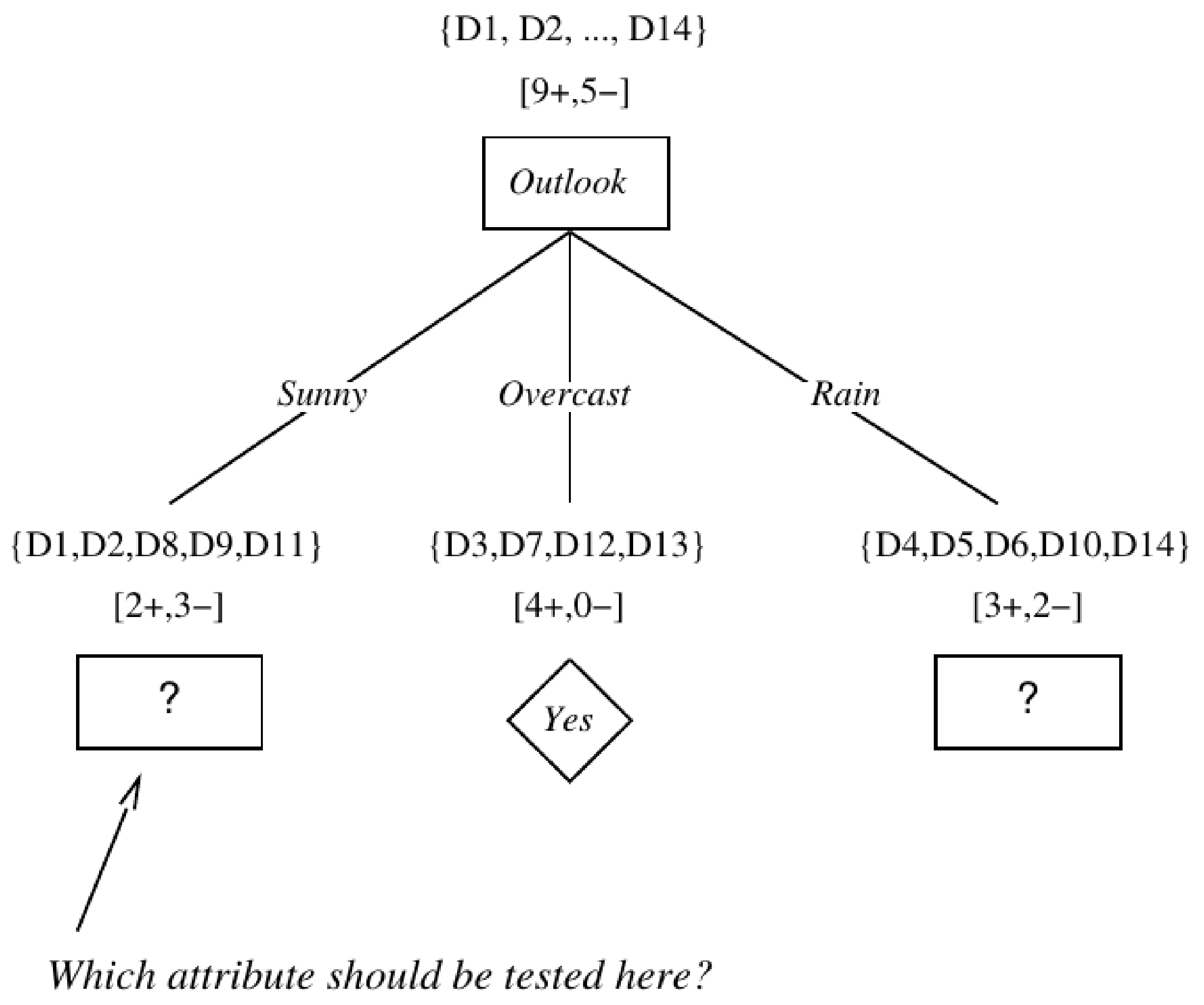
Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151



Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
{D1,D2,D8,D9,D11}
[2+,3−]
{D3,D7,D12,D13}
[4+,0−]
{D4,D5,D6,D10,D14}
[3+,2−]
?
Yes
?
Which attribute should be tested here?

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

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

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

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

Terminating the Algorithm

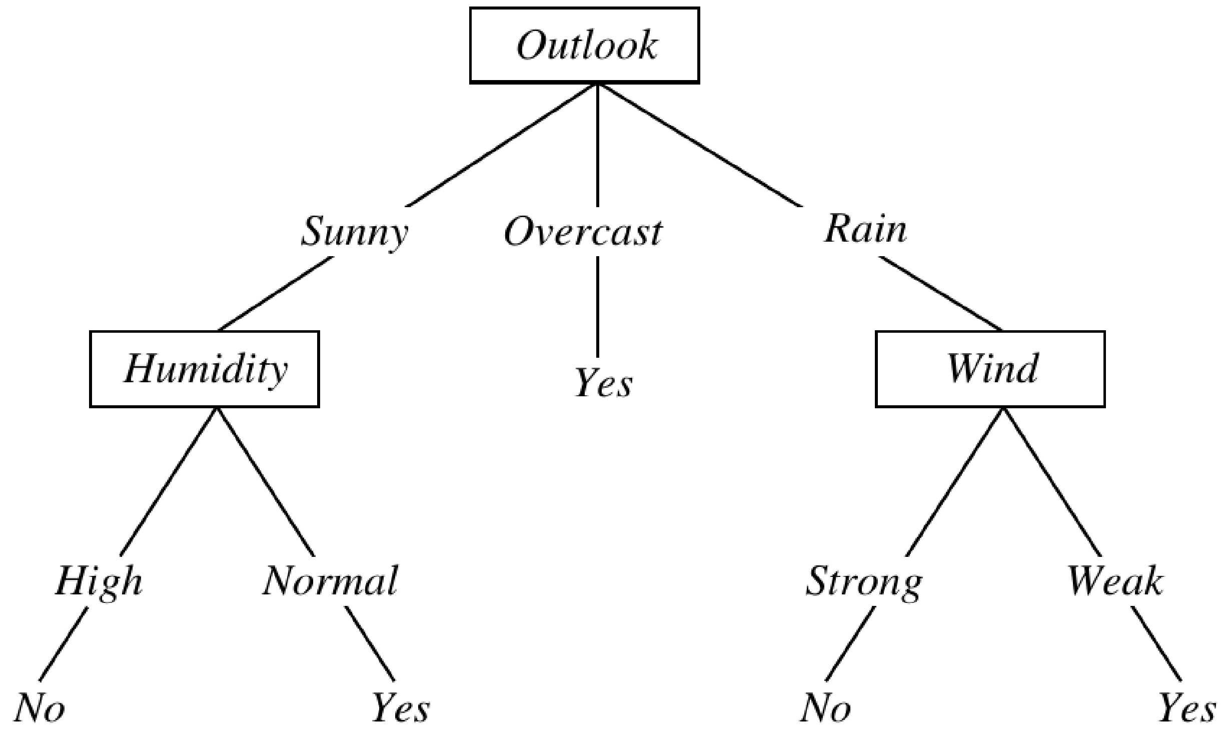
- The process of selecting a new attribute and partitioning the training examples is now repeated for each nonterminal descendant node, this time using only the training examples associated with that node.
- Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along any path through the tree.

Terminating the Algorithm

- This process continues for each new leaf node until either of two conditions is met:
  - every attribute has already been included along this path, or
  - the training examples associated with this leaf node all have the same target attribute value (i.e. their entropy is zero)
- The final decision tree learned by ID3 from the 14 training examples is shown on the next slide.

Final Decision Tree

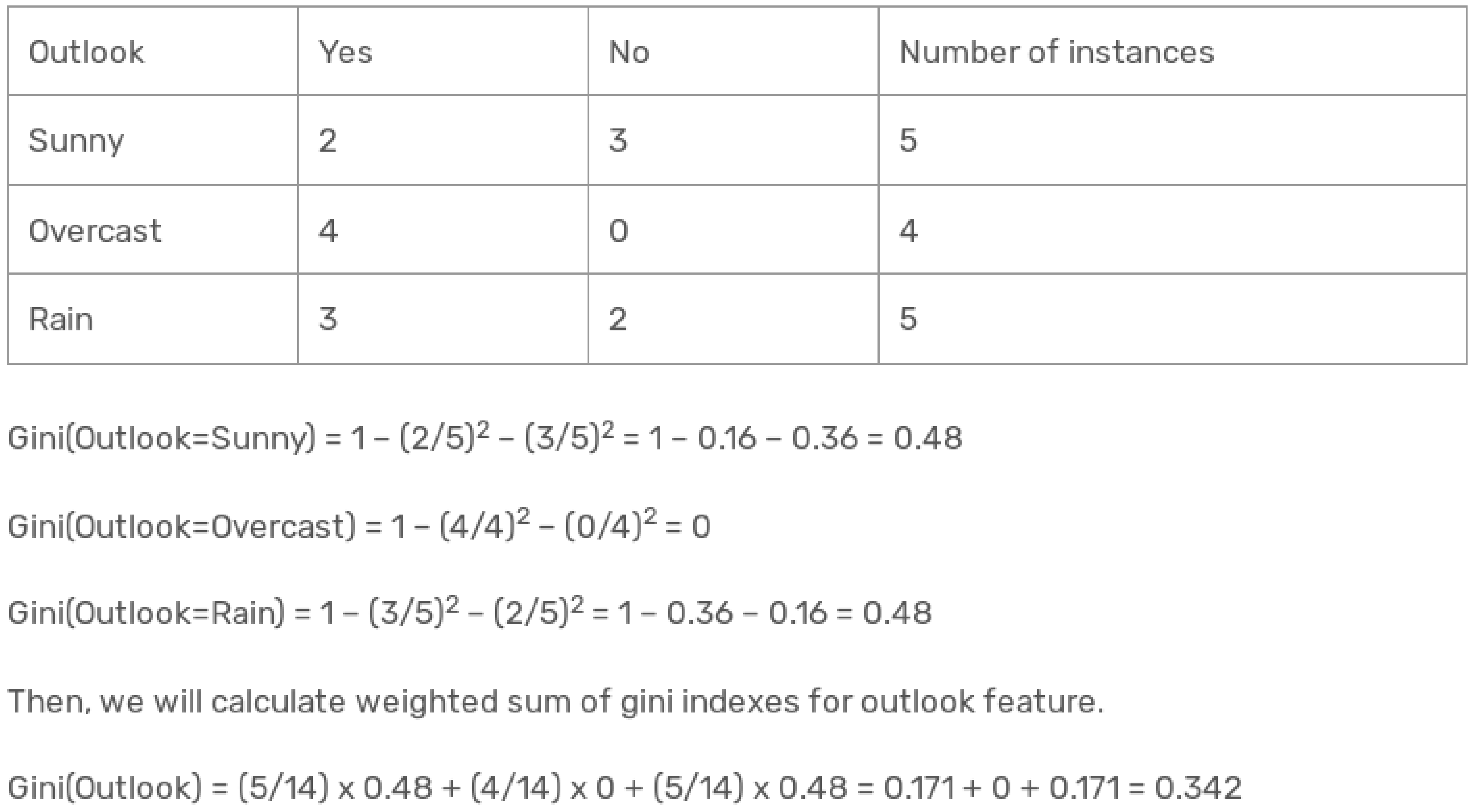
- Final tree constructed (or learnt) using the ID3 algorithm.



The Gini Index

- The Gini index is an alternative to information gain. It is used to decide which attribute is the best to put at the root.
- The Gini index is a measure of impurity (like entropy) and the lower it is the better.
- The Gini index is calculated using the sum of squared probabilities of each class. We can formulate it as illustrated below.
- In the equation below, p_i is the proportion of training data belonging to class i.
- Gini = $1 - \Sigma\ (P_i)^2$ for i=1 to number of classes

Calculating Gini(Outlook)



Outlook	Yes	No	Number of instances
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5

Gini(Outlook=Sunny) = $1 – (2/5)^2 – (3/5)^2 = 1 – 0.16 – 0.36 = 0.48$

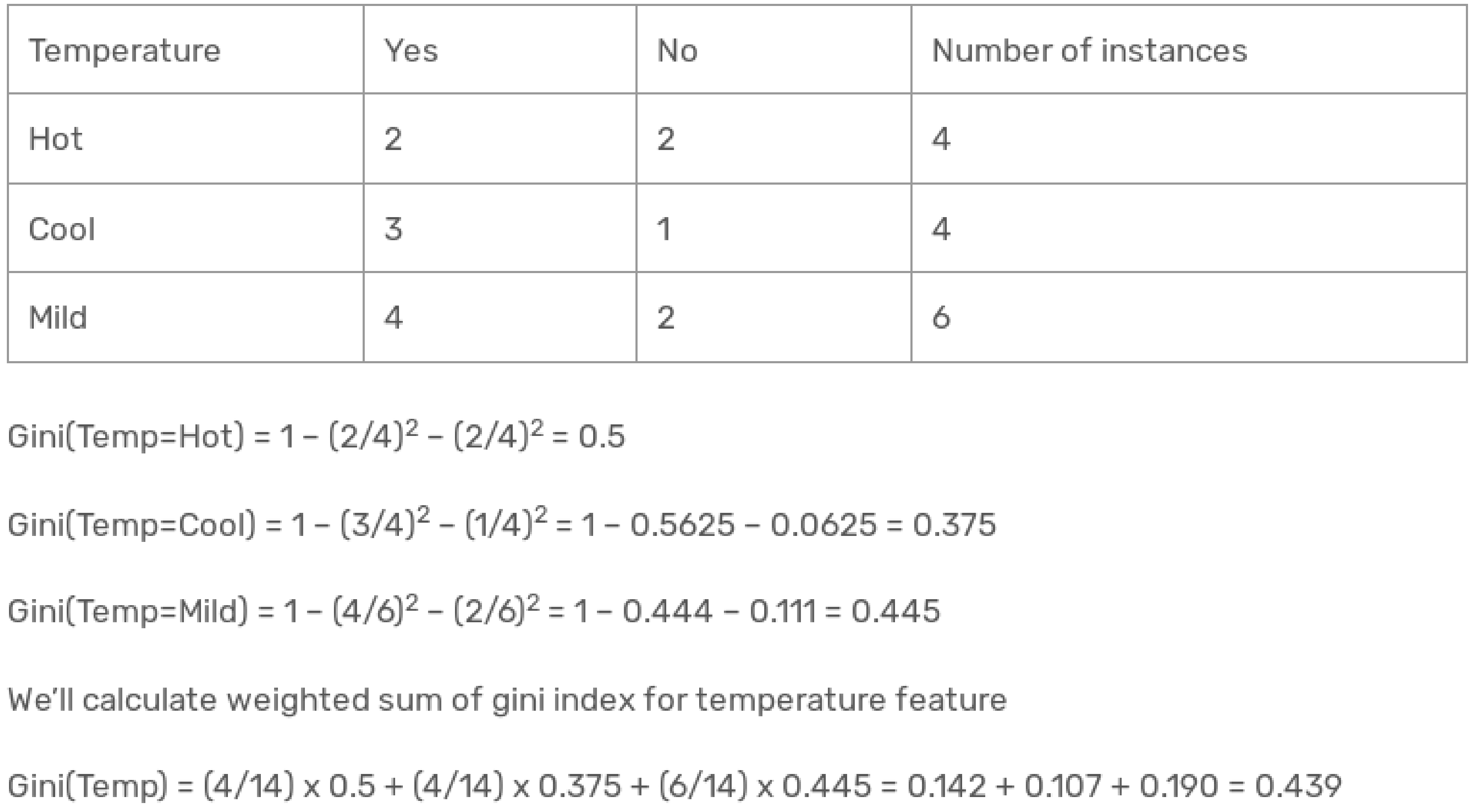
Gini(Outlook=Overcast) = $1 – (4/4)^2 – (0/4)^2 = 0$

Gini(Outlook=Rain) = $1 – (3/5)^2 – (2/5)^2 = 1 – 0.36 – 0.16 = 0.48$

Then, we will calculate weighted sum of gini indexes for outlook feature.

Gini(Outlook) = $(5/14) \times 0.48 + (4/14) \times 0 + (5/14) \times 0.48 = 0.171 + 0 + 0.171 = 0.342$

Gini(Temperature)



Temperature	Yes	No	Number of instances
Hot	2	2	4
Cool	3	1	4
Mild	4	2	6

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) = $1 - (4/6)^2 - (2/6)^2 = 1 - 0.444 - 0.111 = 0.445$

We'll calculate weighted sum of gini index for temperature feature

Gini(Temp) = $(4/14) \times 0.5 + (4/14) \times 0.375 + (6/14) \times 0.445 = 0.142 + 0.107 + 0.190 = 0.439$

Gini(Humidity)

Humidity	Yes	No	Number of instances
High	3	4	7
Normal	6	1	7

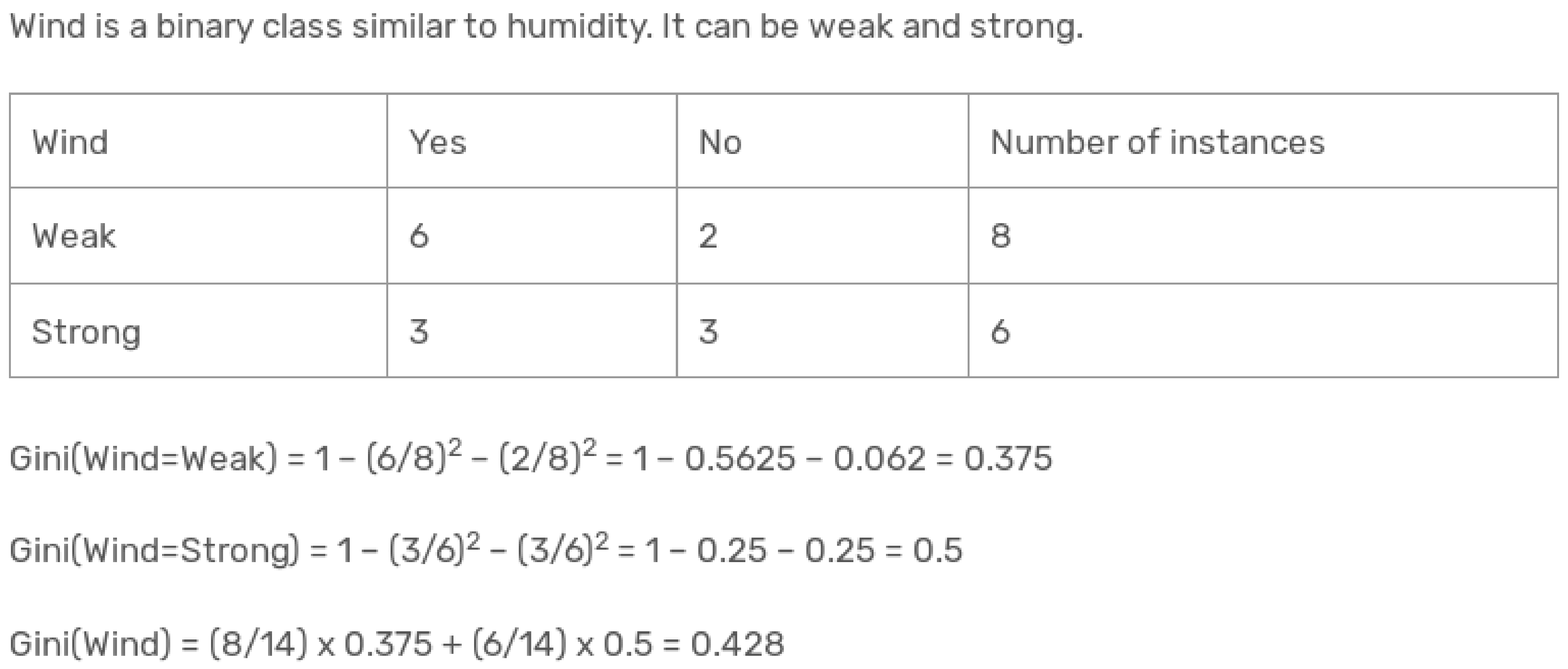
Gini(Humidity=High) = $1 - (3/7)^2 - (4/7)^2 = 1 - 0.183 - 0.326 = 0.489$

Gini(Humidity=Normal) = $1 - (6/7)^2 - (1/7)^2 = 1 - 0.734 - 0.02 = 0.244$

Weighted sum for humidity feature will be calculated next

Gini(Humidity) = $(7/14) \times 0.489 + (7/14) \times 0.244 = 0.367$

Gini(Wind)



Wind is a binary class similar to humidity. It can be weak and strong.

Wind	Yes	No	Number of instances
Weak	6	2	8
Strong	3	3	6

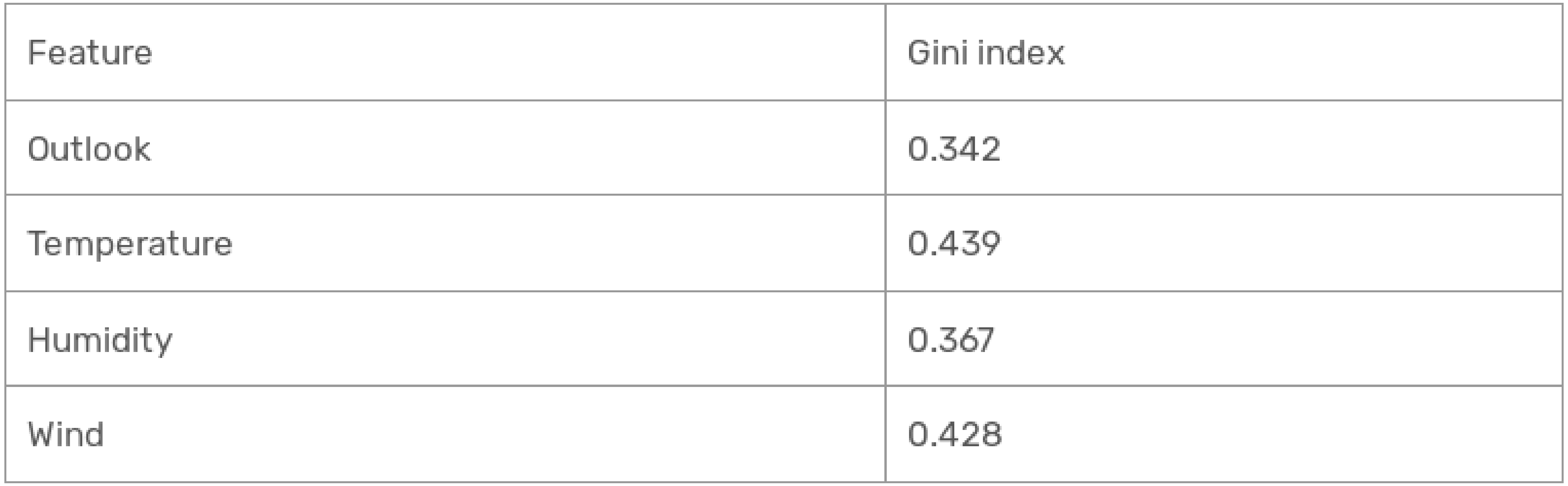
Gini(Wind=Weak) = $1 - (6/8)^2 - (2/8)^2 = 1 - 0.5625 - 0.062 = 0.375$

Gini(Wind=Strong) = $1 - (3/6)^2 - (3/6)^2 = 1 - 0.25 - 0.25 = 0.5$

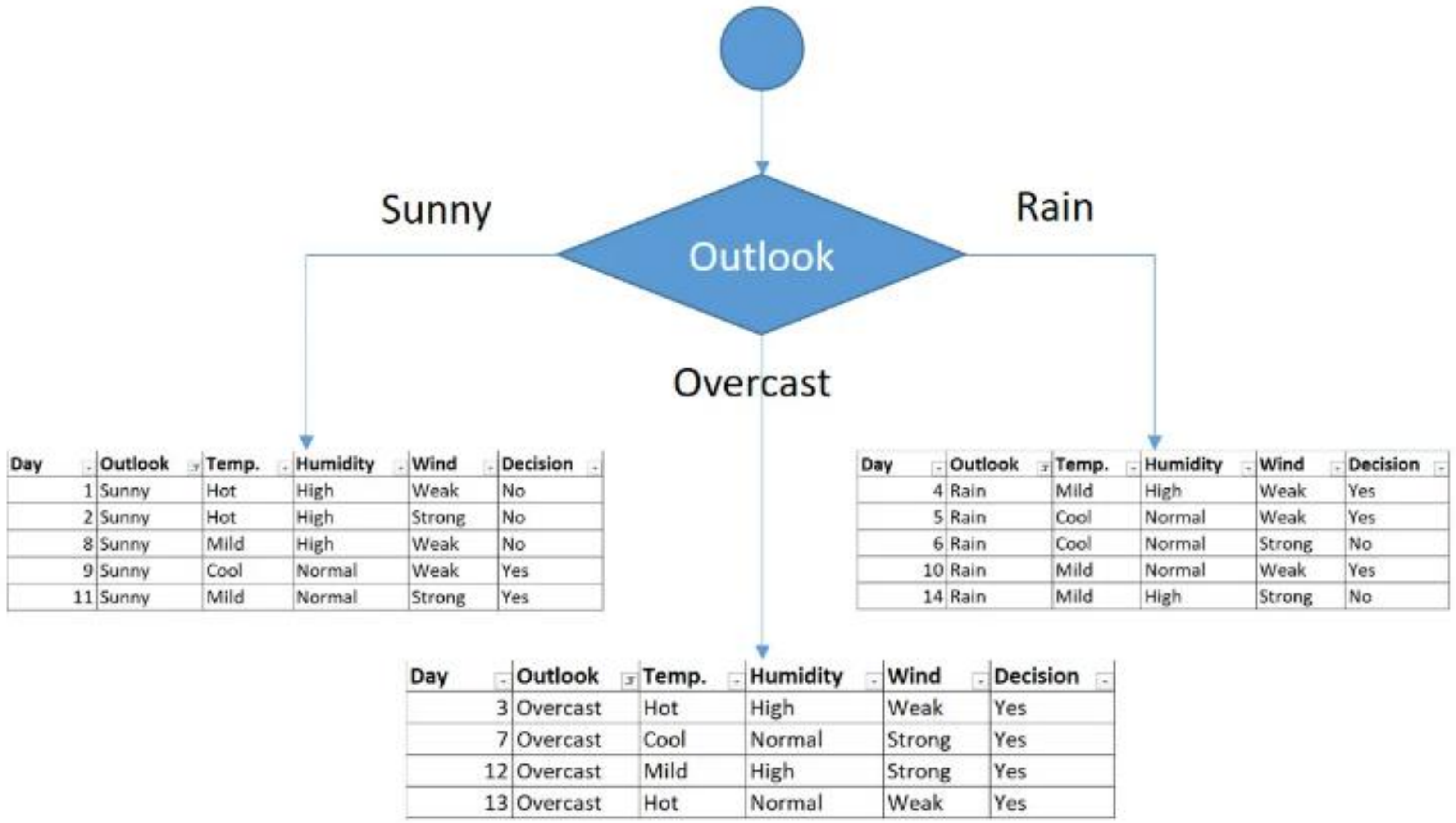
Gini(Wind) = $(8/14) \times 0.375 + (6/14) \times 0.5 = 0.428$

Deciding the Winner

- Outlook wins because it has the lowest gini index value, meaning it is the most pure.



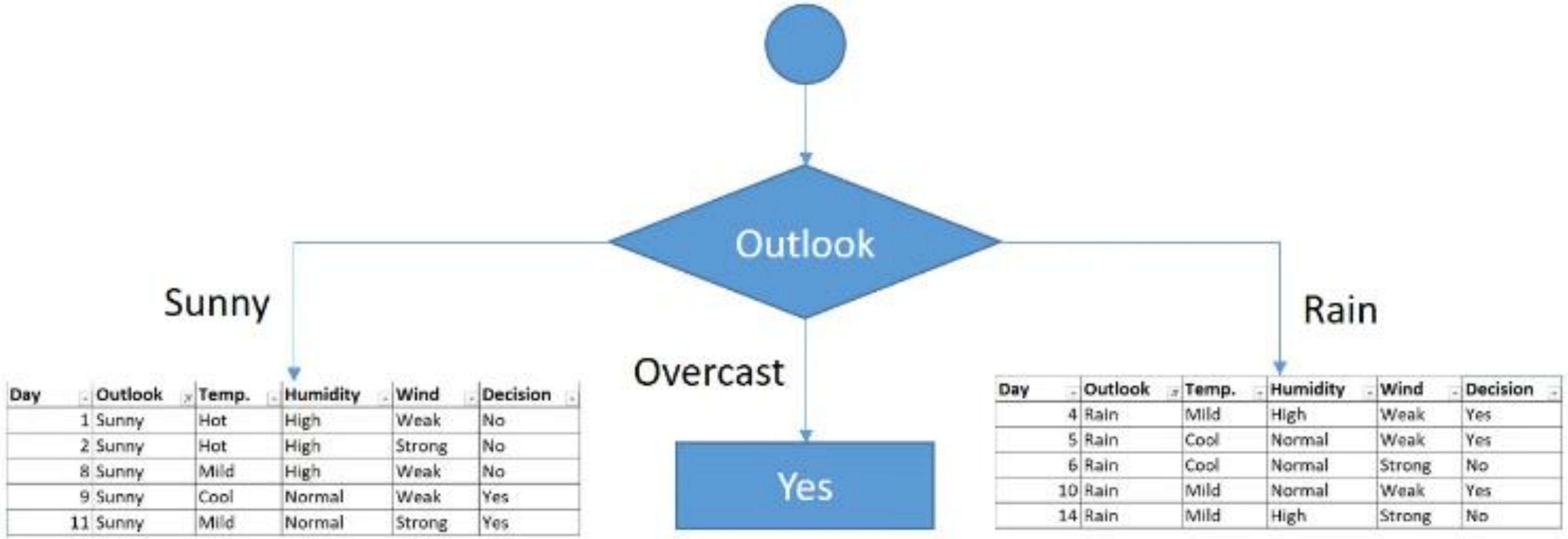
Feature	Gini index
Outlook	0.342
Temperature	0.439
Humidity	0.367
Wind	0.428



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

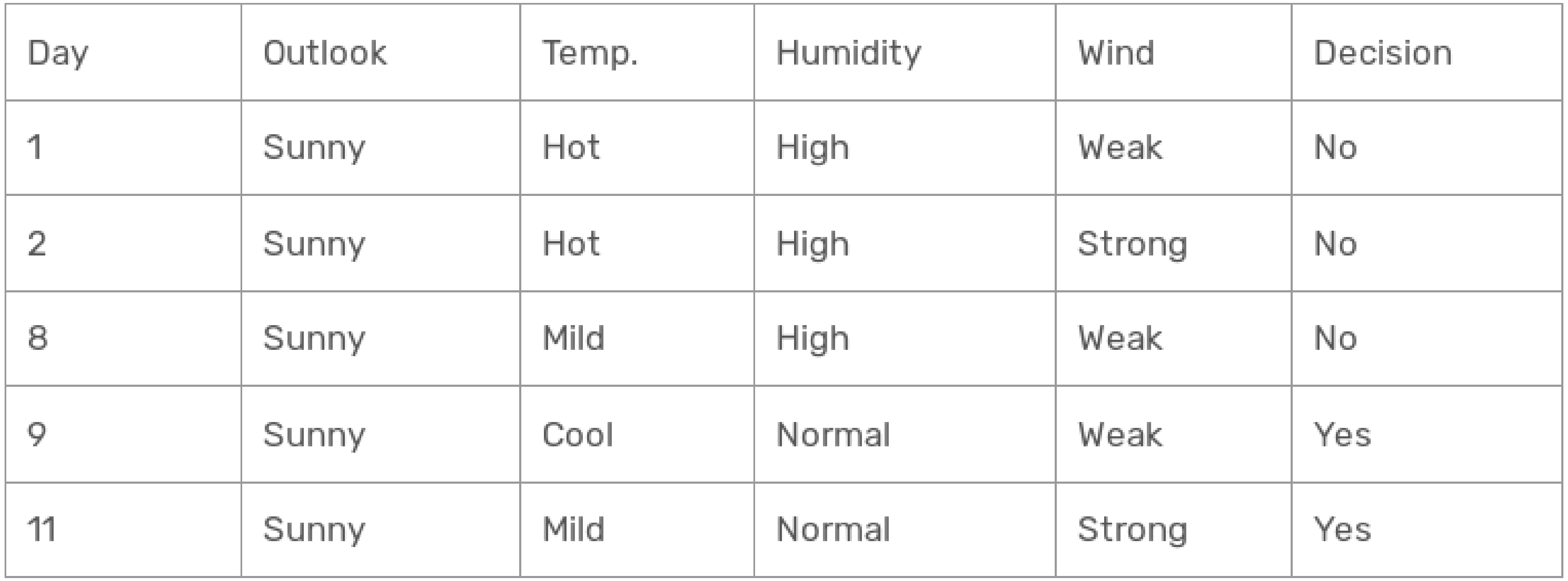


Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

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

Sunny Subset

- Now we want to find the feature that should go on the left of the root node



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

Gini of Temperature for Sunny Outlook

Temperature	Yes	No	Number of instances
Hot	0	2	2
Cool	1	0	1
Mild	1	1	2

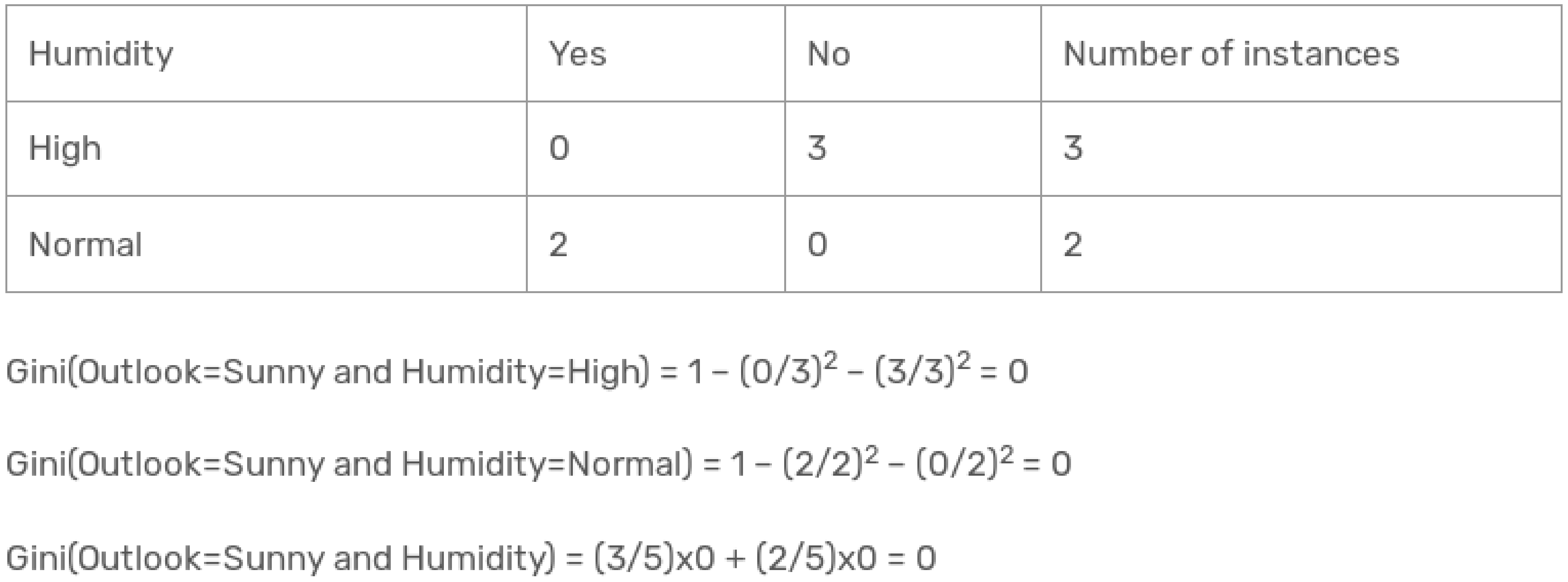
Gini(Outlook=Sunny and Temp.=Hot) = $1 - (0/2)^2 - (2/2)^2 = 0$

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.) = $(2/5)\text{x}0 + (1/5)\text{x}0 + (2/5)\text{x}0.5 = 0.2$

Gini of Humidity for Sunny Outlook



Humidity	Yes	No	Number of instances
High	0	3	3
Normal	2	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) = $(3/5)\text{x}0 + (2/5)\text{x}0 = 0$

Gini of Wind for Sunny Outlook

Wind	Yes	No	Number of instances
Weak	1	2	3
Strong	1	1	2

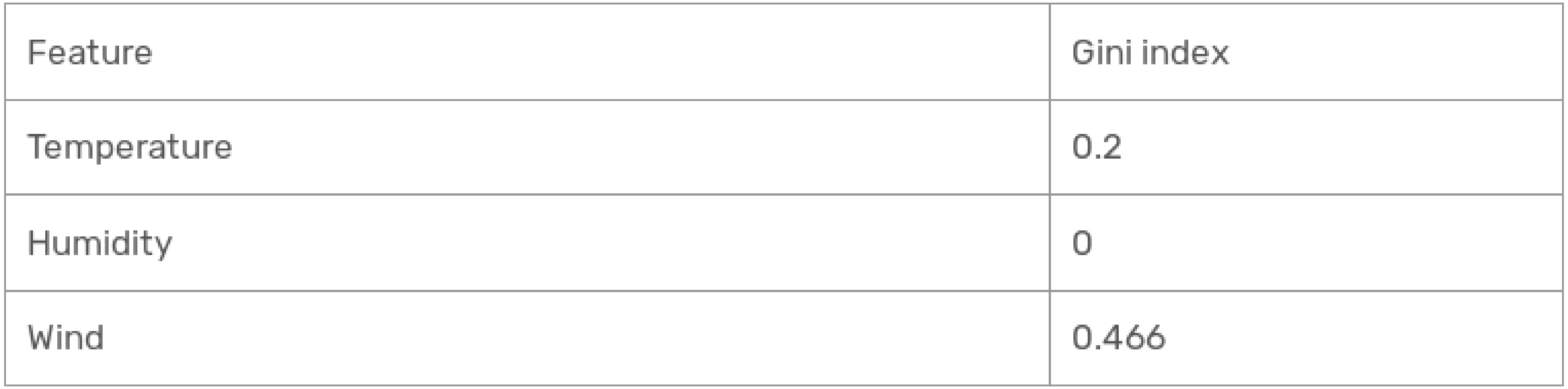
Gini(Outlook=Sunny and Wind=Weak) = $1 - (1/3)^2 - (2/3)^2 = 0.266$

Gini(Outlook=Sunny and Wind=Strong) = $1- (1/2)^2 - (1/2)^2 = 0.2$

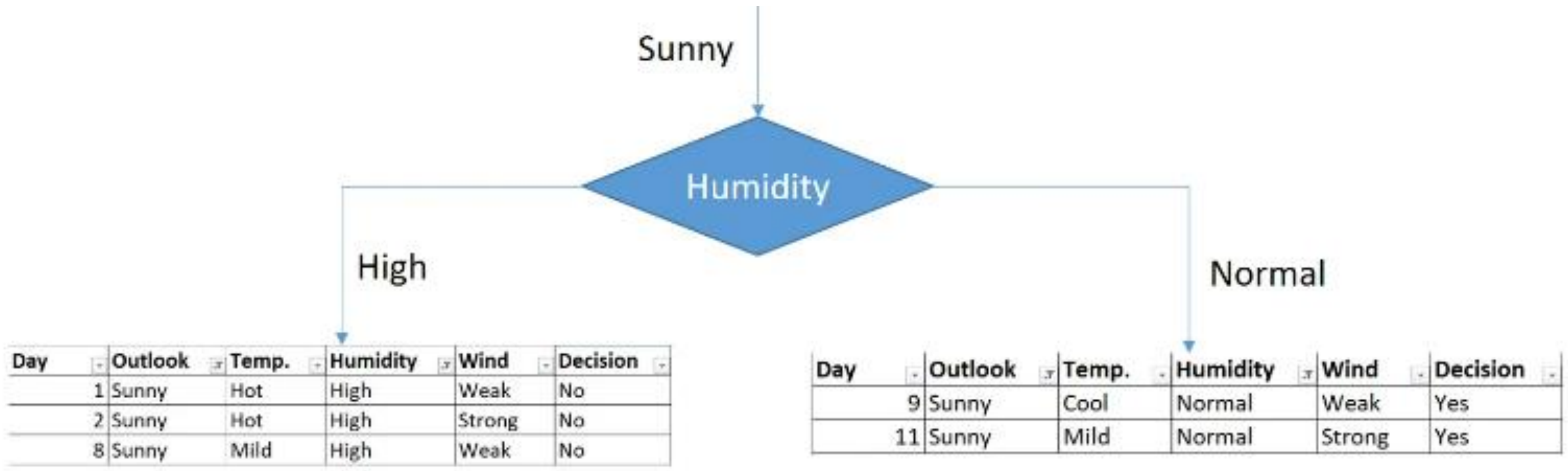
Gini(Outlook=Sunny and Wind) = $(3/5)\text{x}0.266 + (2/5)\text{x}0.2 = 0.466$

Decision for Sunny Outlook

- Humidity is the winner as it has the lowest gini index.

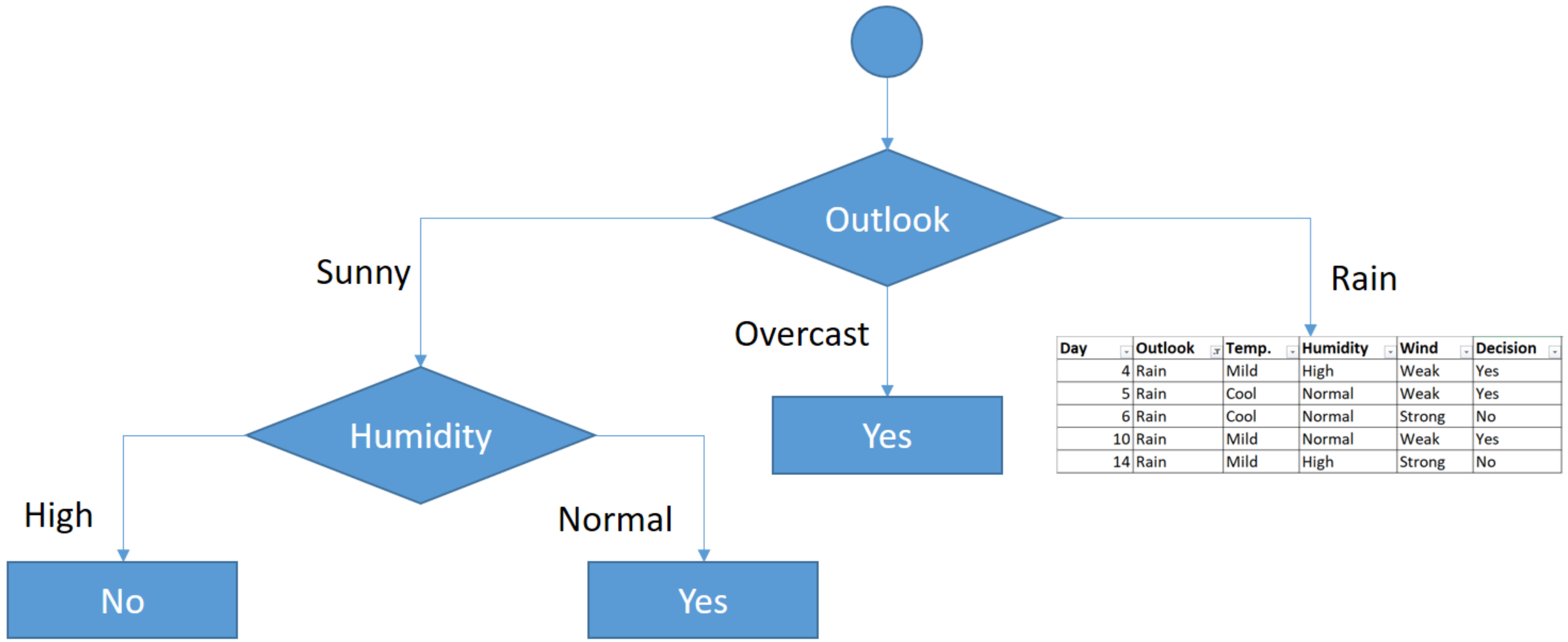


Feature	Gini index
Temperature	0.2
Humidity	0
Wind	0.466



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

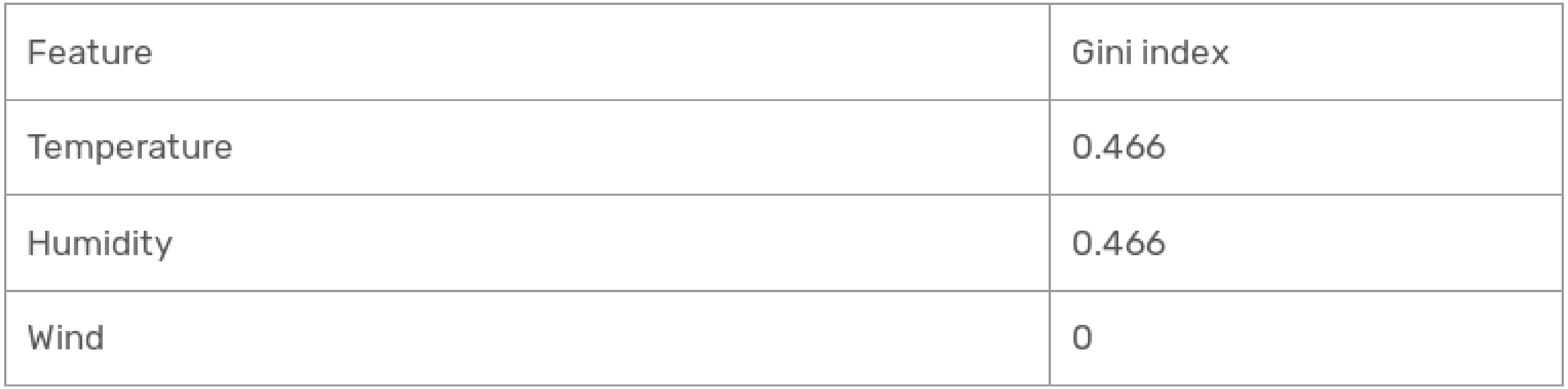
Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes



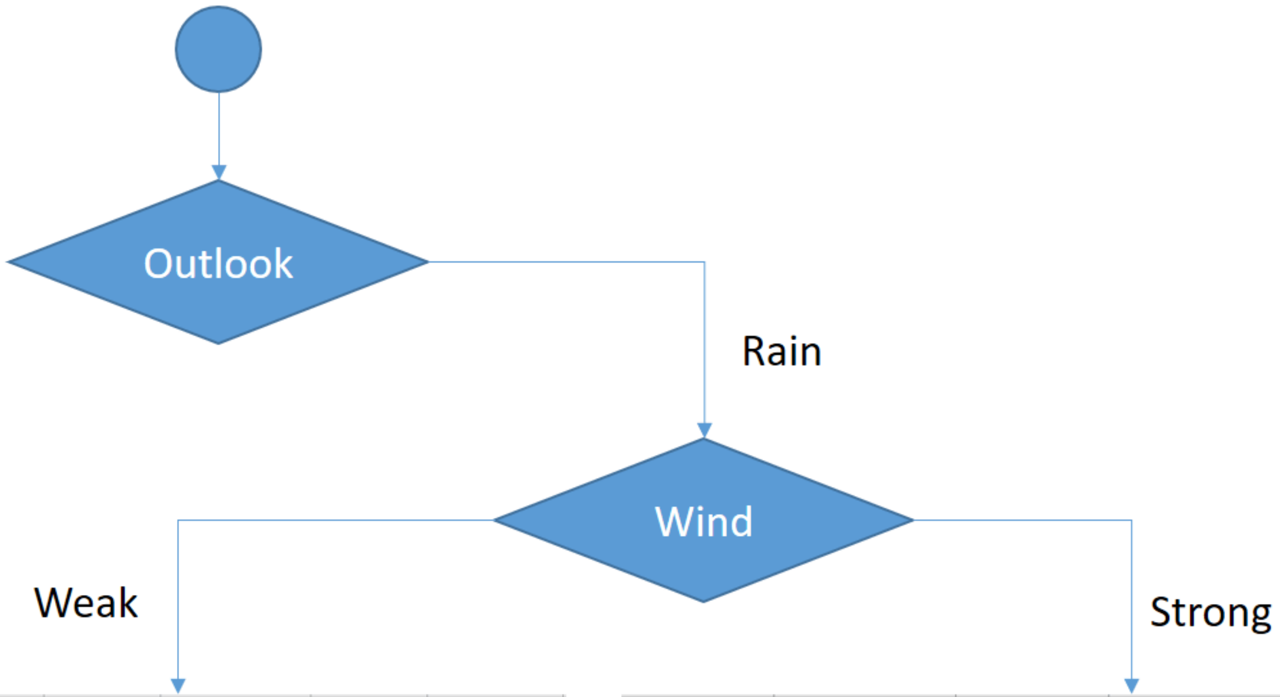
Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Decision for Rain

- Wind is the winning attribute to the right of the root node (calculations are omitted).



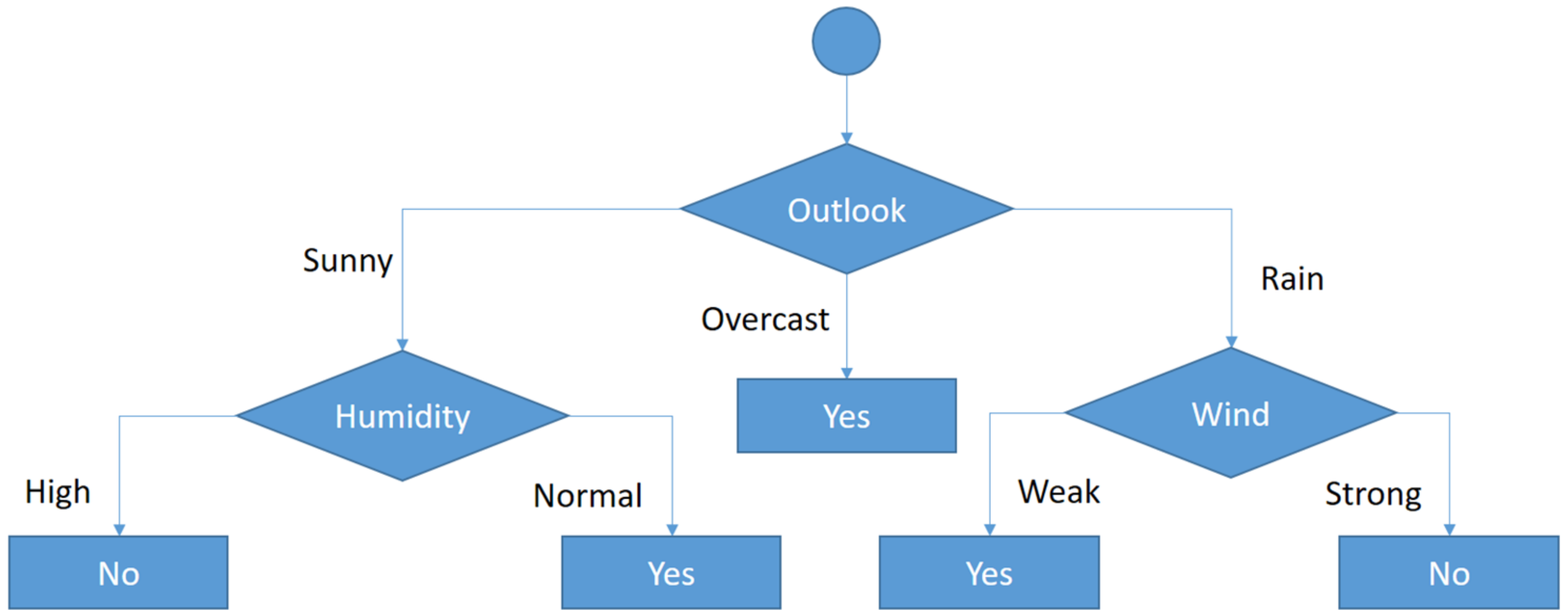
Feature	Gini index
Temperature	0.466
Humidity	0.466
Wind	0



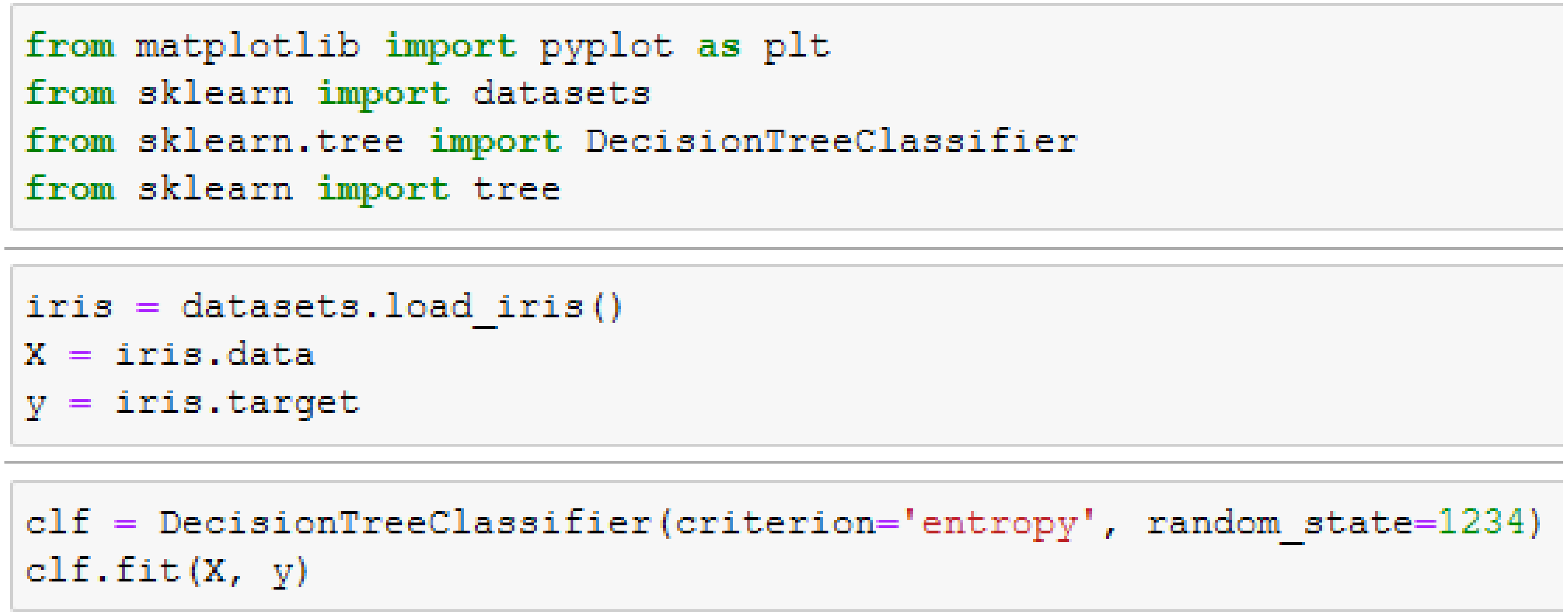
Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
6	Rain	Cool	Normal	Strong	No
14	Rain	Mild	High	Strong	No

The Final Decision Tree



Example

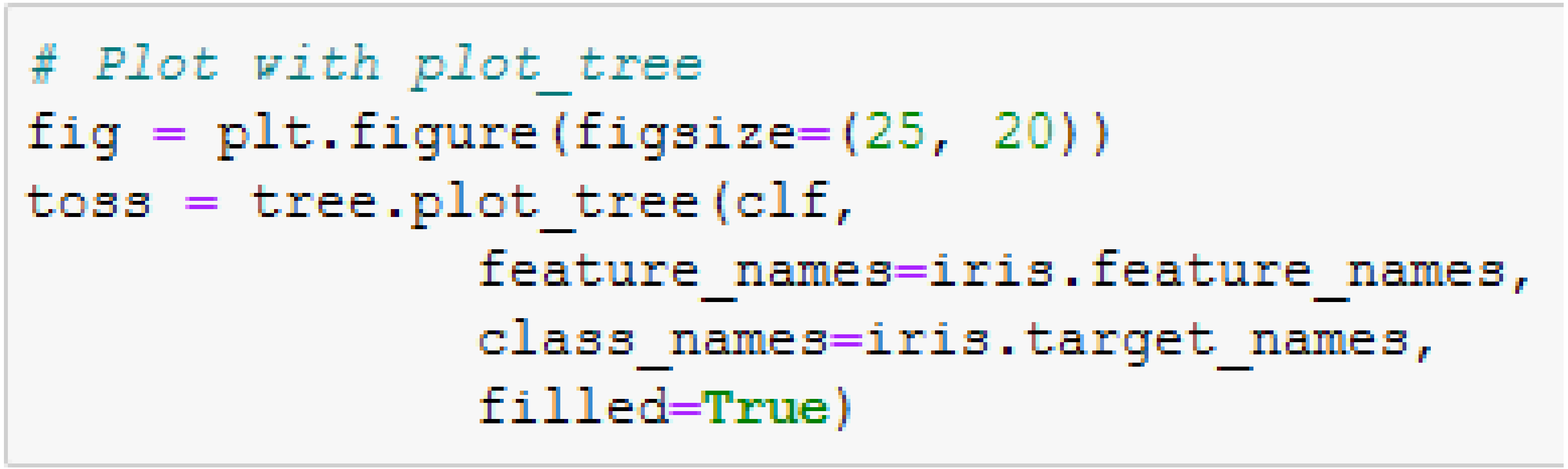


```
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

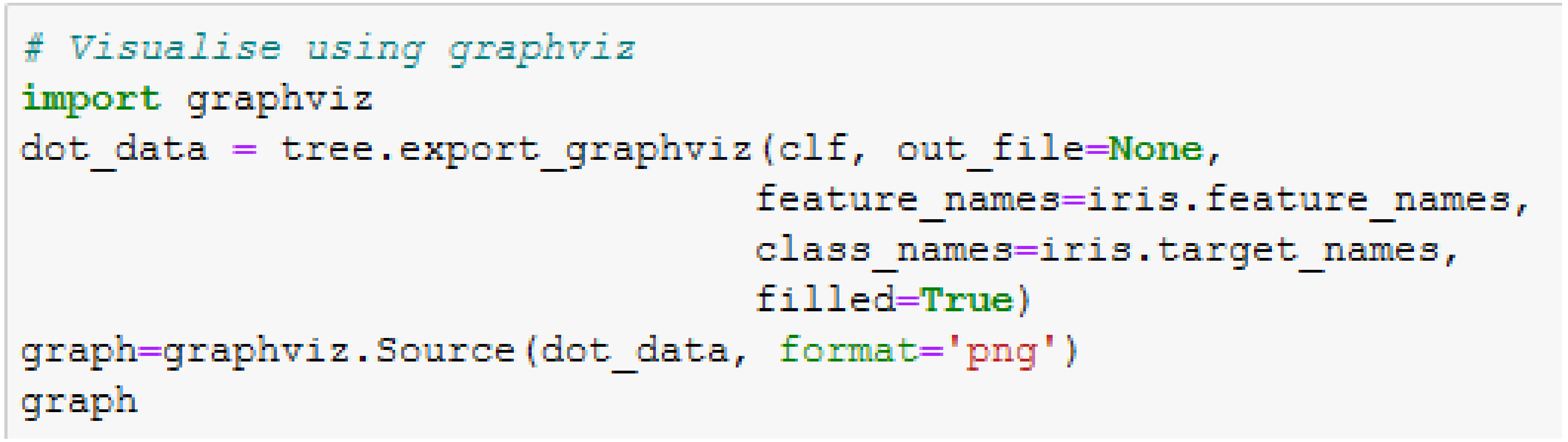
```
clf = DecisionTreeClassifier(criterion='entropy', random_state=1234)
clf.fit(X, y)
```

Visualising with plot_tree

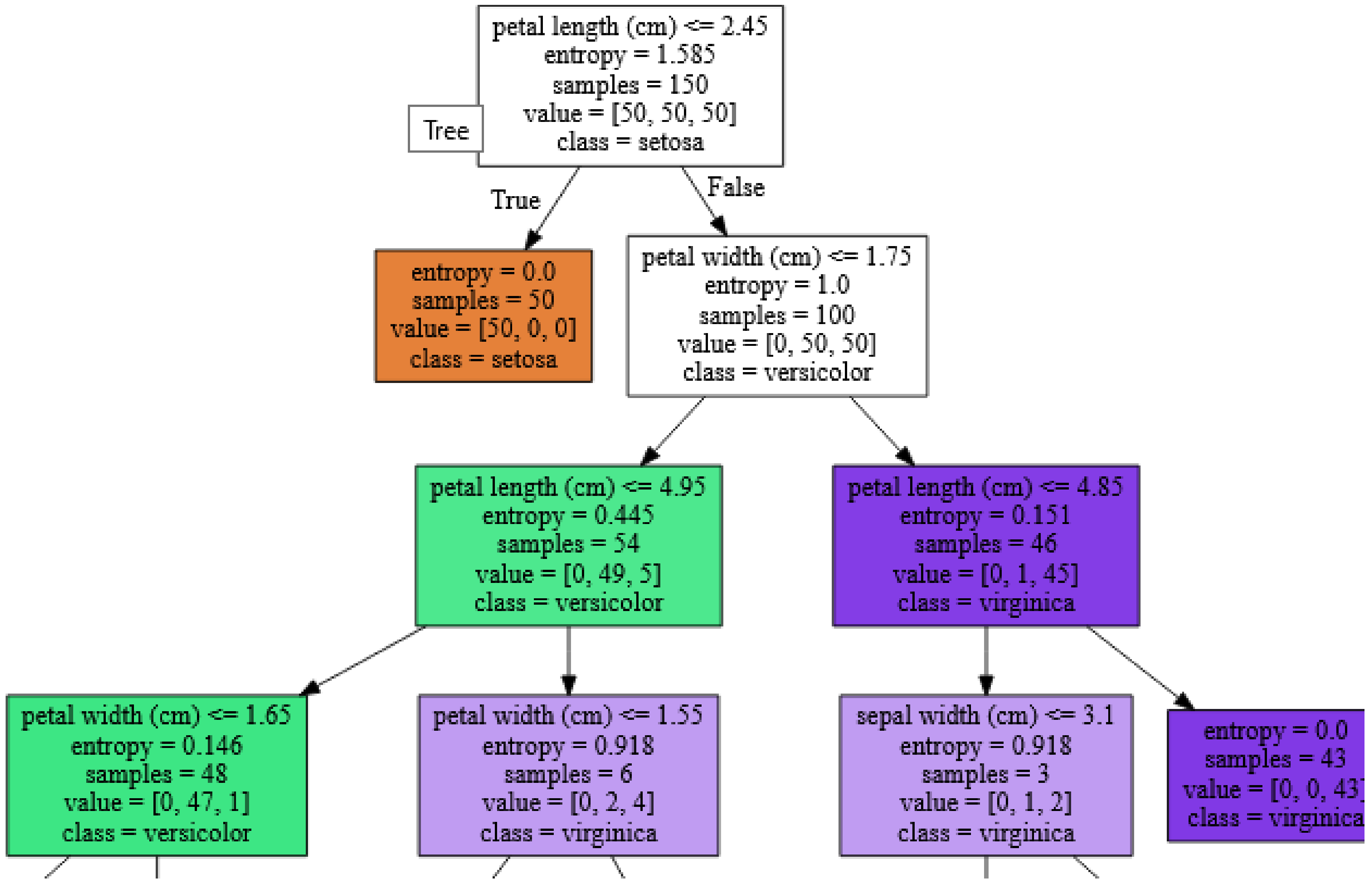


```
# Plot with plot_tree
fig = plt.figure(figsize=(25, 20))
toss = tree.plot_tree(clf,
                      feature_names=iris.feature_names,
                      class_names=iris.target_names,
                      filled=True)
```

Visualising with graphviz



```
# Visualise using graphviz
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None,
                               feature_names=iris.feature_names,
                               class_names=iris.target_names,
                               filled=True)
graph=graphviz.Source(dot_data, format='png')
graph
```


Tree
petal length (cm) <= 2.45
entropy = 1.585
samples = 150
value = [50, 50, 50]
class = setosa
True
False
entropy = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) <= 1.75
entropy = 1.0
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) <= 4.95
entropy = 0.445
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) <= 4.85
entropy = 0.151
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) <= 1.65
entropy = 0.146
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) <= 1.55
entropy = 0.918
samples = 6
value = [0, 2, 4]
class = virginica
sepal width (cm) <= 3.1
entropy = 0.918
samples = 3
value = [0, 1, 2]
class = virginica
entropy = 0.0
samples = 43
value = [0, 0, 43
class = virginica

Concluding Remarks

- Gini is the default in the Scikit-learn library, but it is possible to choose information gain (entropy) by setting criterion = ‘entropy’ in the classifier constructor.
- Decision trees have the advantage of transparency, i.e., their decisions are explainable and understandable by nonexperts.
- The performance of decision trees improves dramatically when a collection of them is used in what is known as a random forest.

Reference

- A Step by Step CART Decision Tree Example, available from https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/
- *Machine Learning* by Tom Mitchell, 1997