YaleNUSCollege

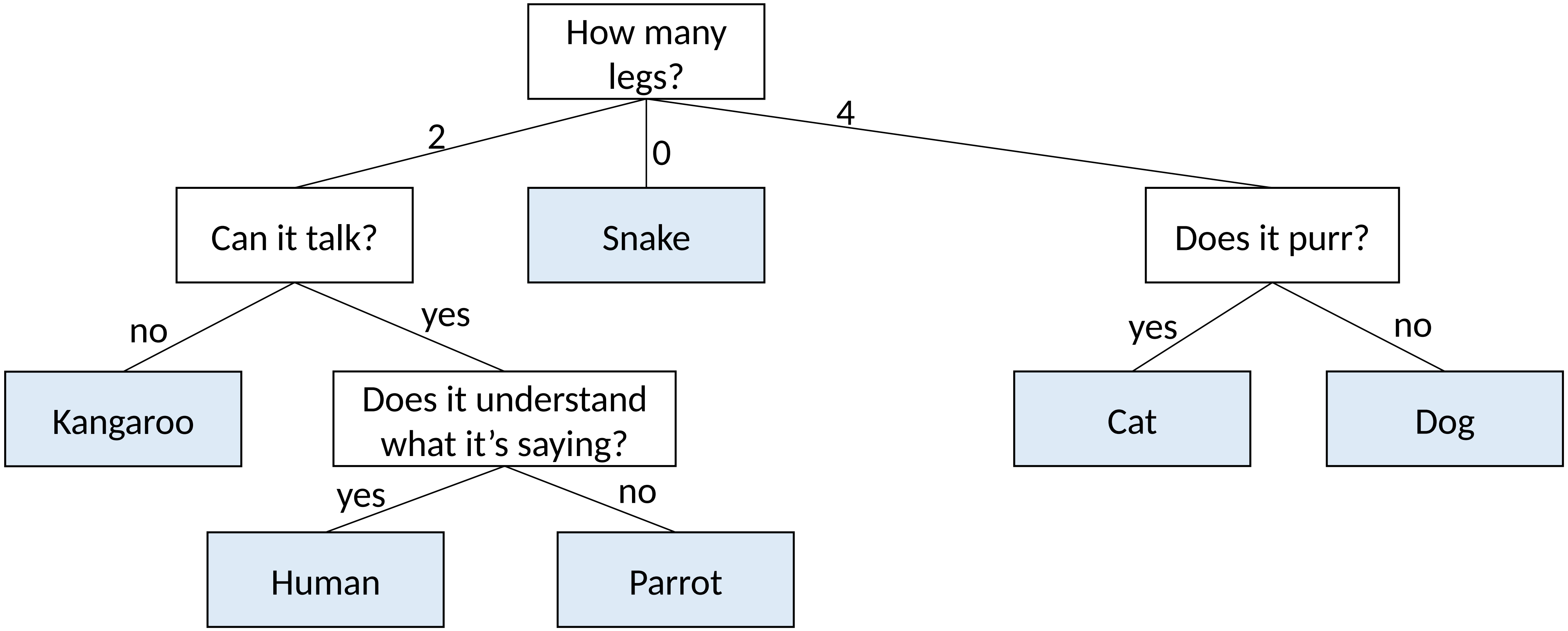
YSC2239 Lecture 21

Today’s class

- Decision Trees

Decision Trees

A Decision Tree is a very simple way to classify data. It is simply a tree of questions that must be answered in sequence to yield a predicted classification.

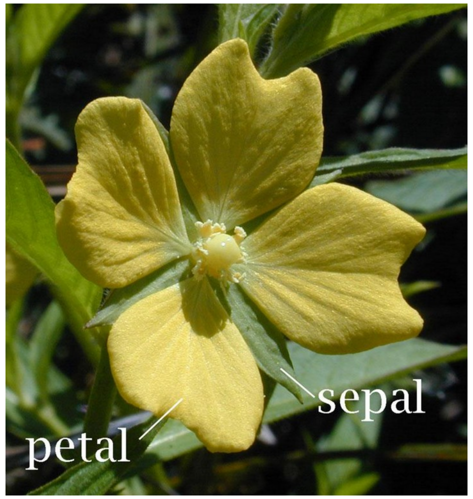


Example: Flower Classification

The Iris flower data set is a commonly used example:

- Created by statistician/biologist Ronald Fisher for his paper “The use of multiple measurements in taxonomic problems”.
- Data set consists of 150 flower measurements from 3 different species.
- For each, we have “petal length”, “petal width”, “sepal length”, “sepal width”.

Goal is to predict species from other data.



https://en.wikipedia.org/wiki/Sepal

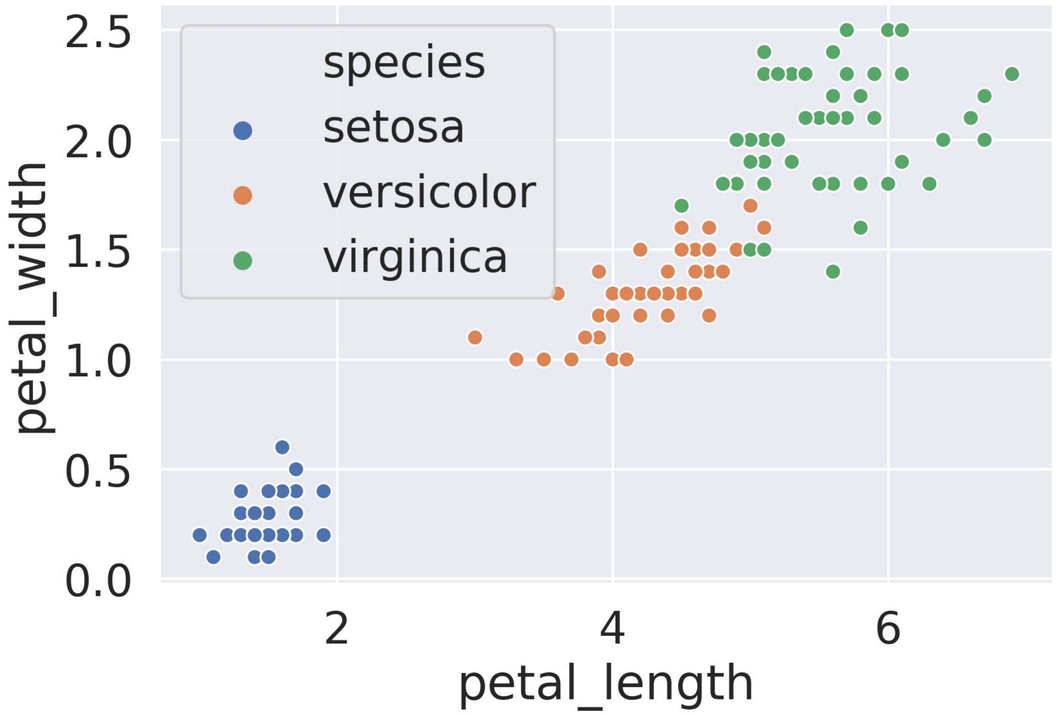
sepal_length	sepal_width	petal_length	petal_width	species
5.5	2.5	4.0	1.3	versicolor
6.4	2.9	4.3	1.3	versicolor
4.8	3.4	1.6	0.2	setosa
5.3	3.7	1.5	0.2	setosa
6.7	2.5	5.8	1.8	virginica

Decision Tree Basics

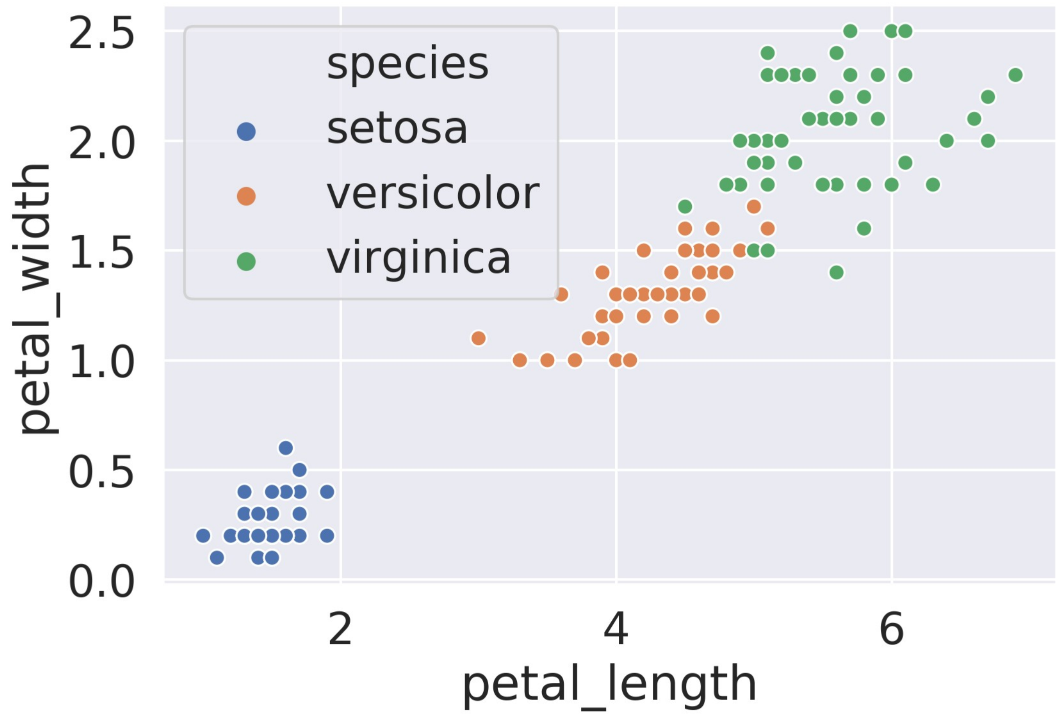
Example: Using Petal Data Only

The plot below shows the width and length of the petals of each flower, with the species annotated in the form of color.

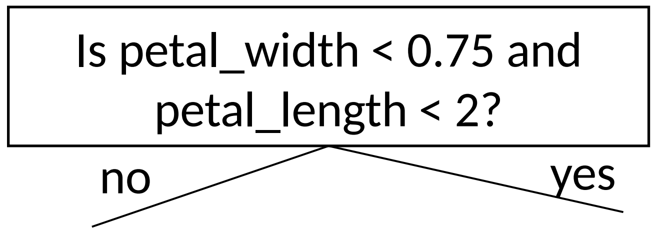
We can build a decision tree manually just by looking at this picture.

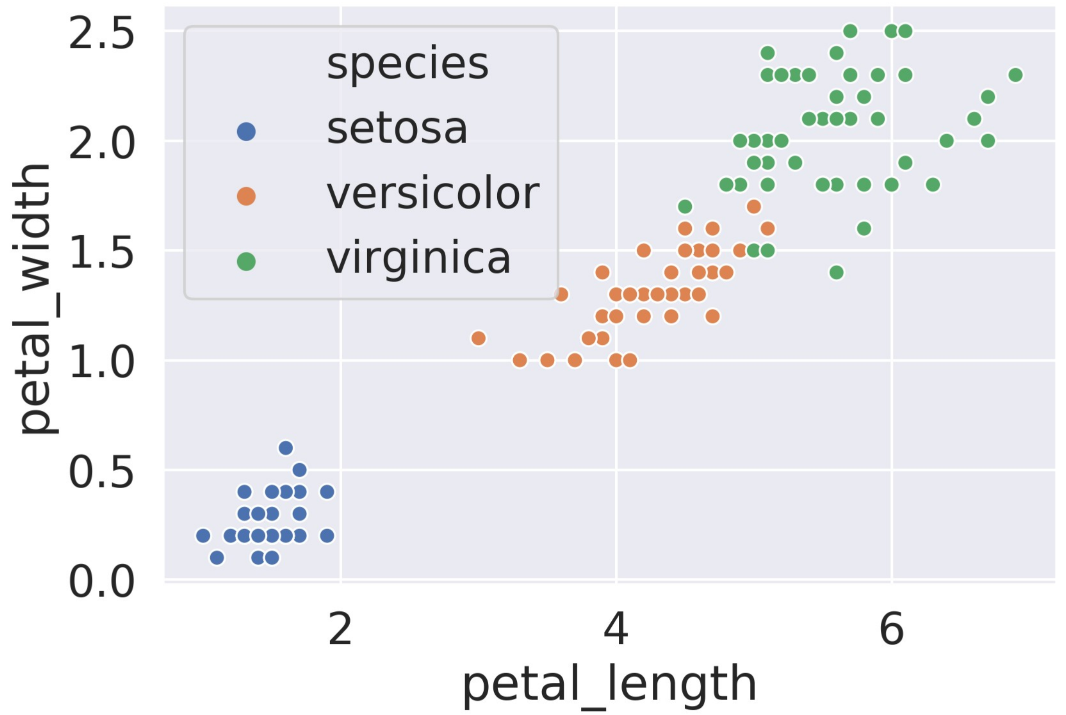


Example: Using Petal Data Only

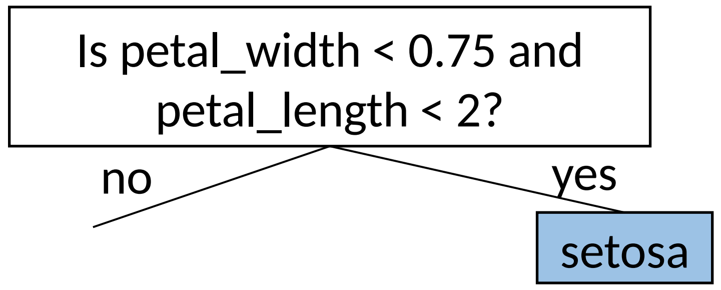


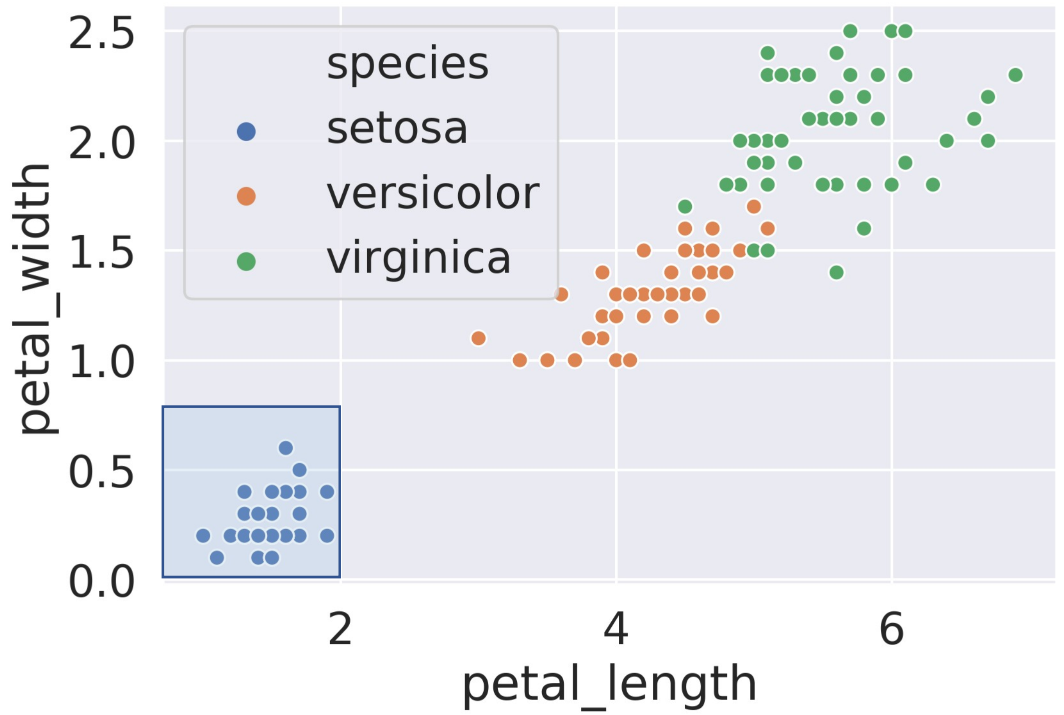
Example: Using Petal Data Only



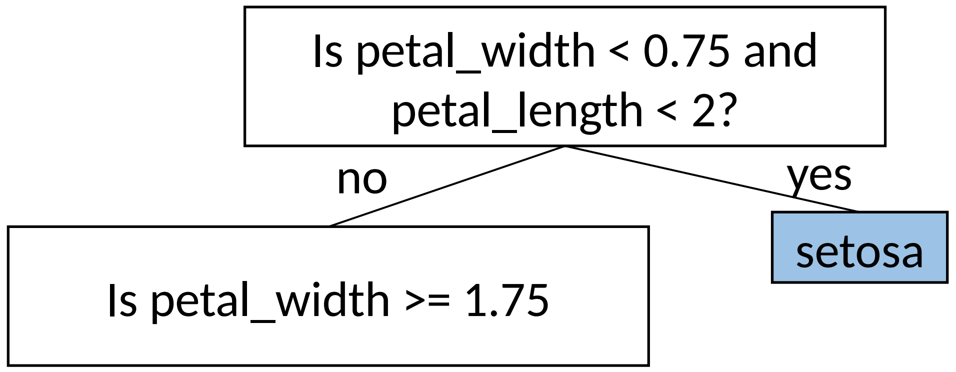


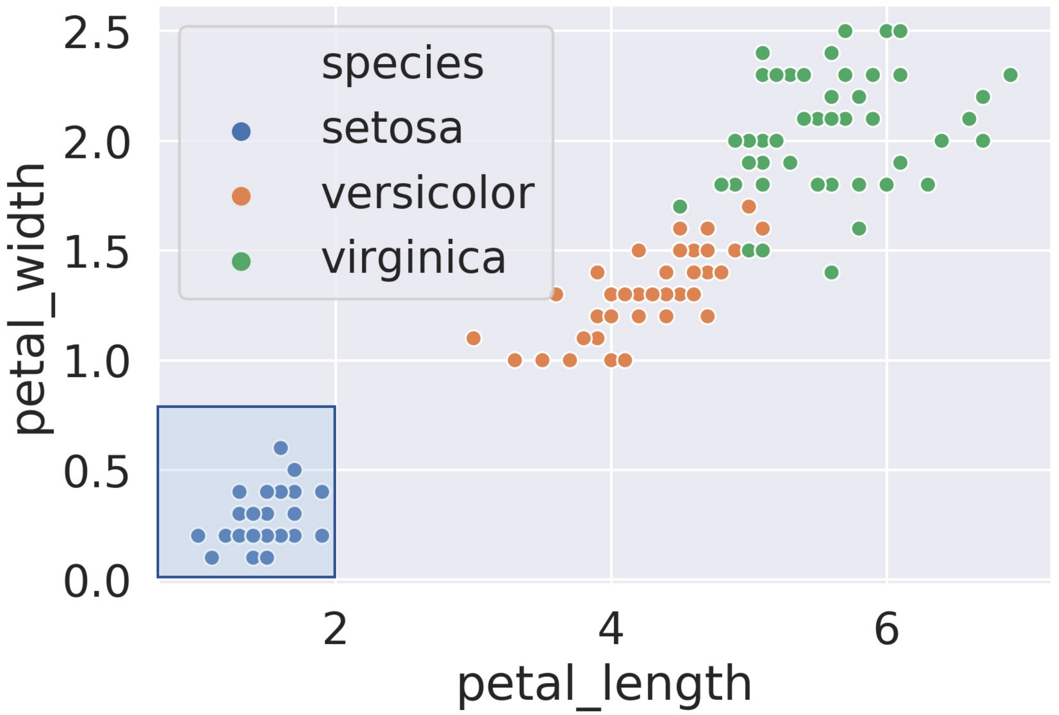
Example: Using Petal Data Only



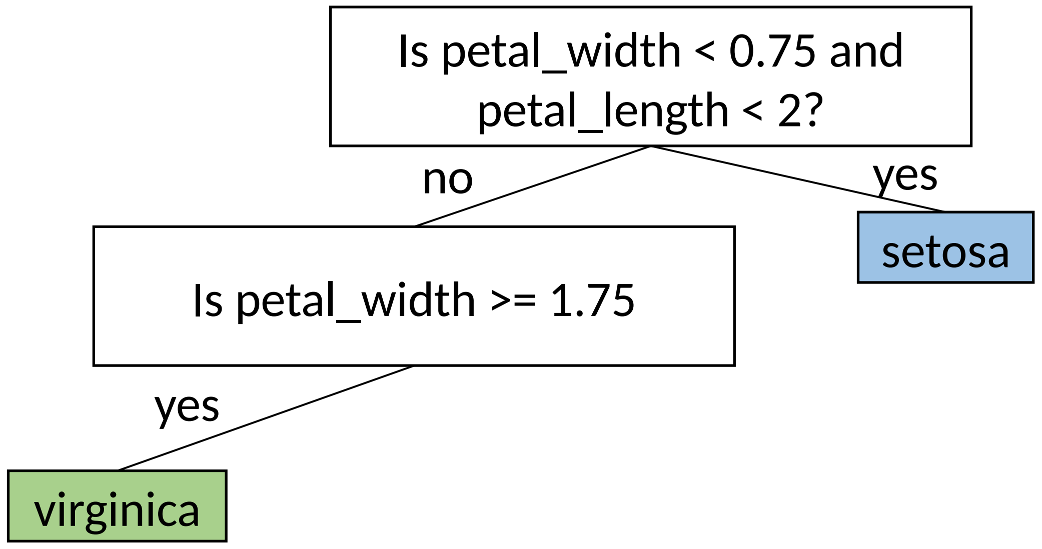


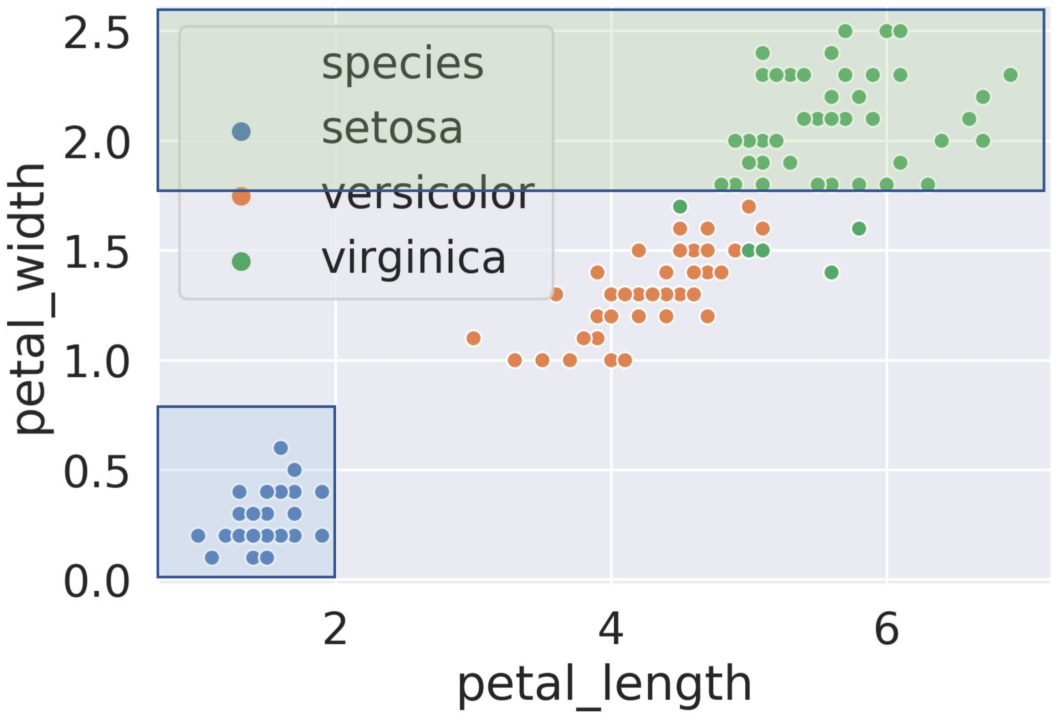
Example: Using Petal Data Only



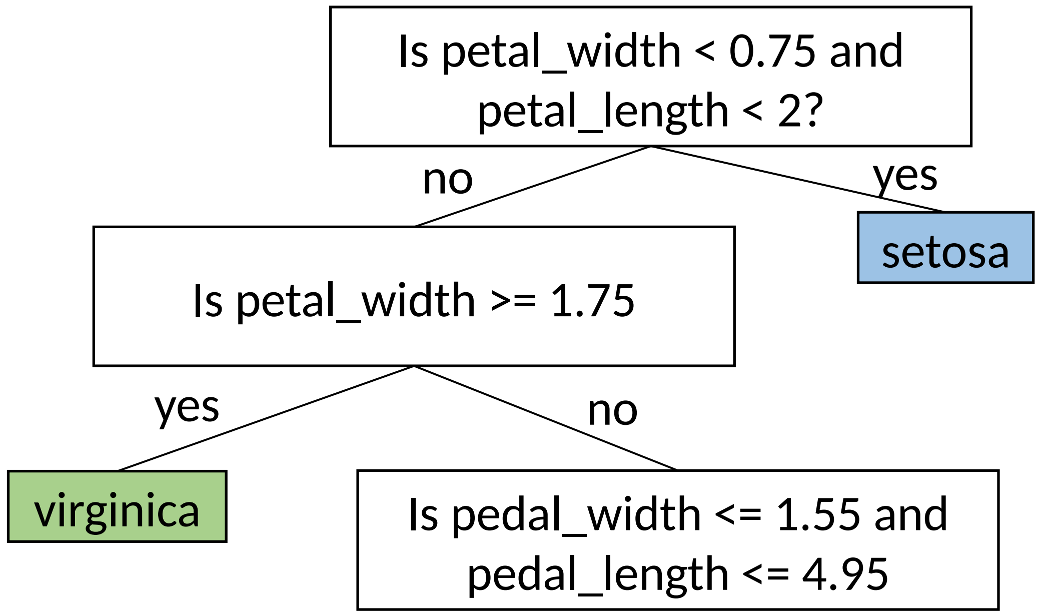


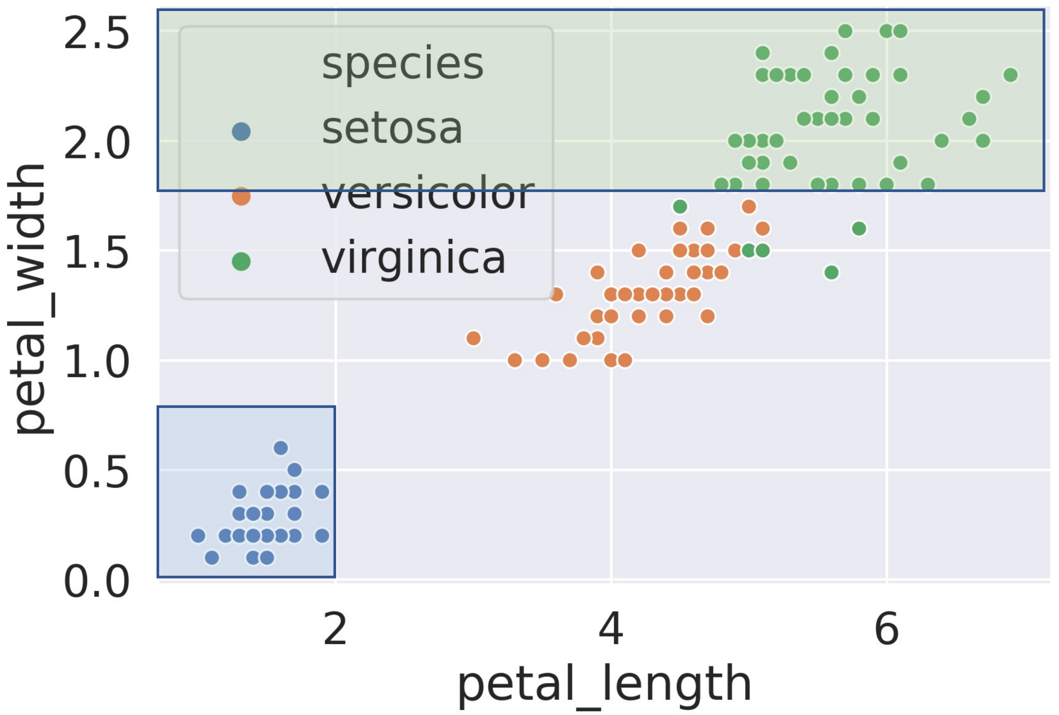
Example: Using Petal Data Only



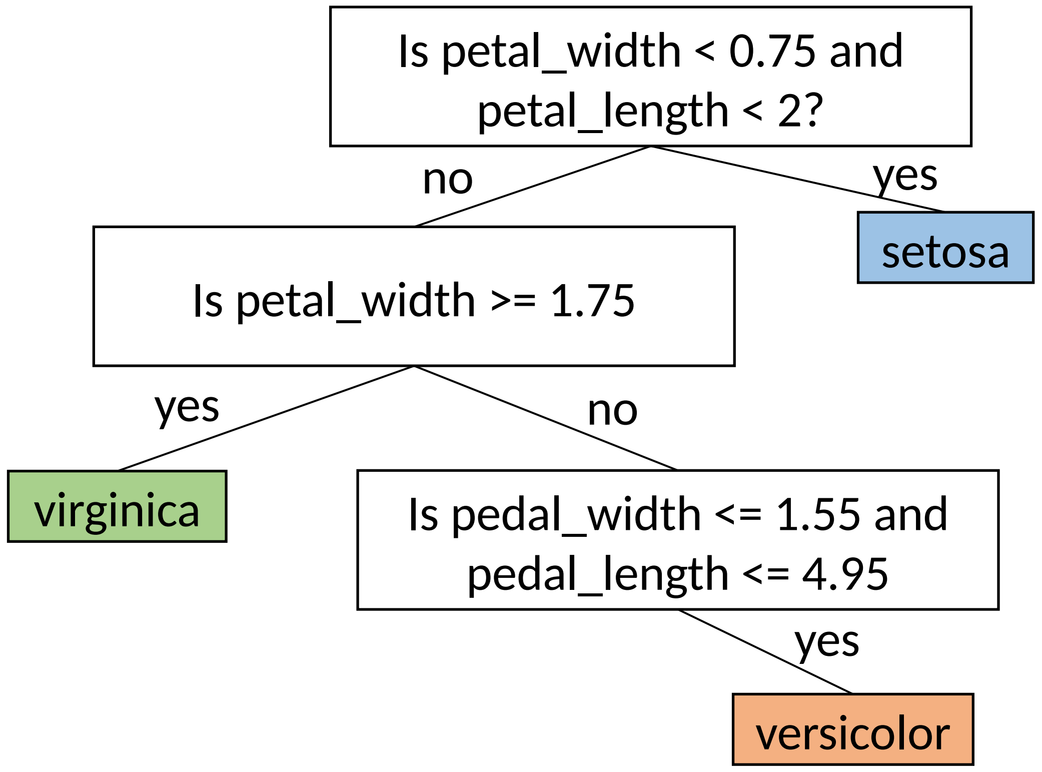


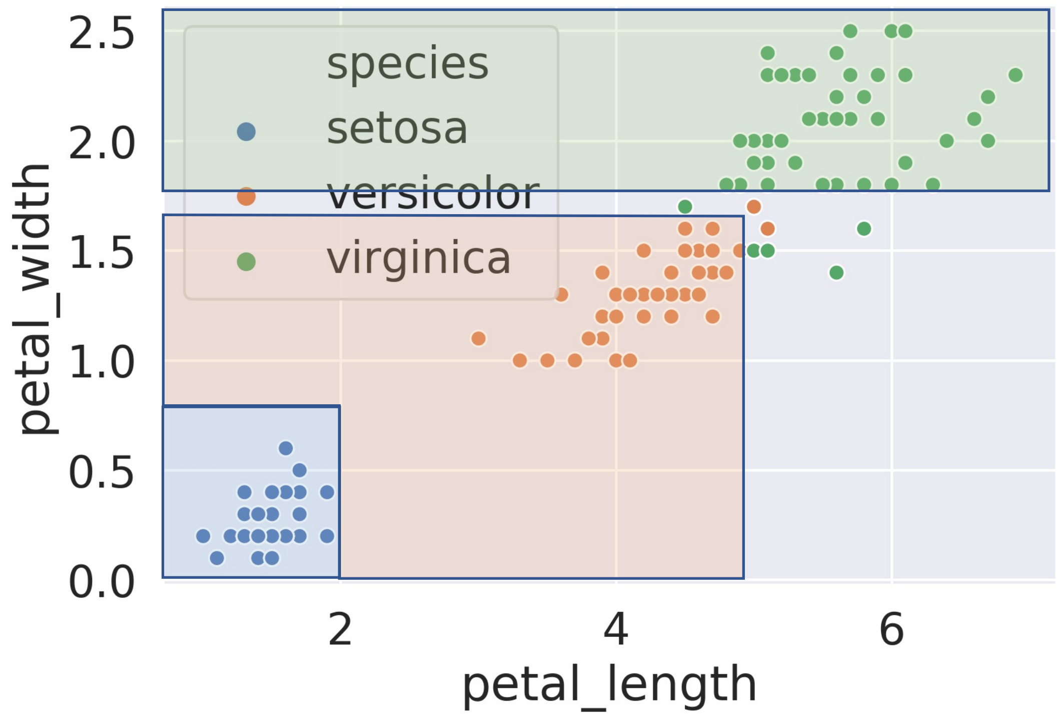
Example: Using Petal Data Only


Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
no
virginica
Is pedal_width <= 1.55 and pedal_length <= 4.95

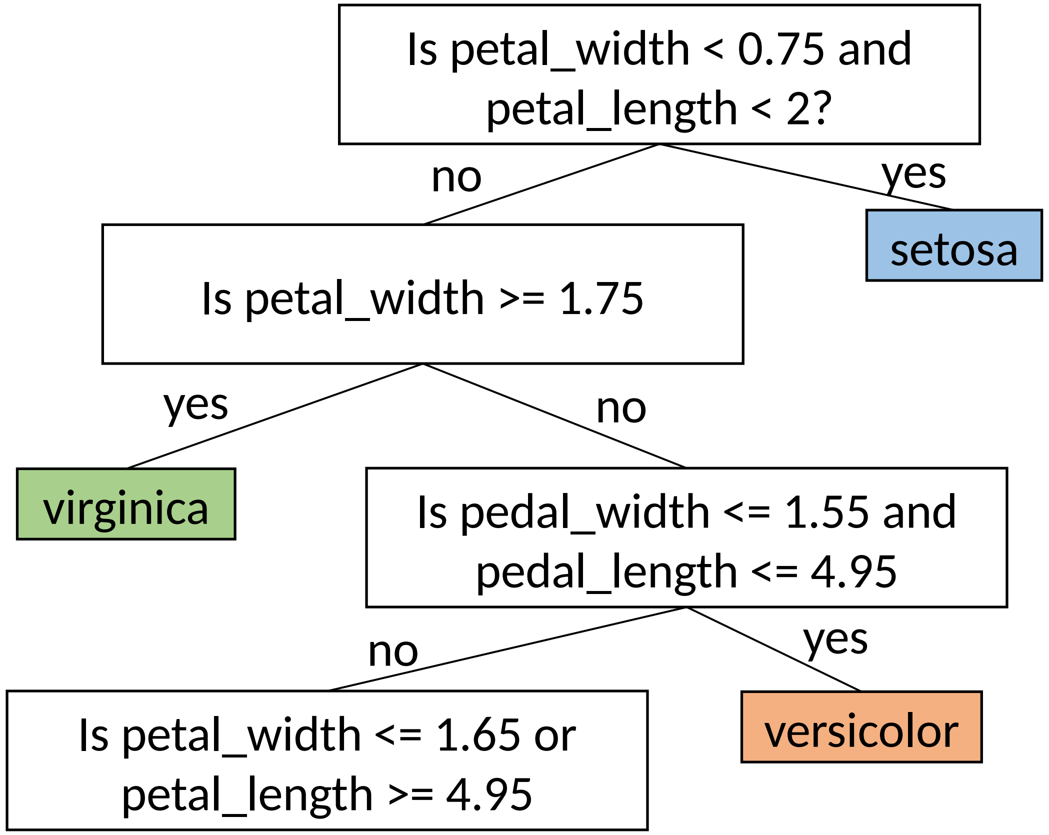


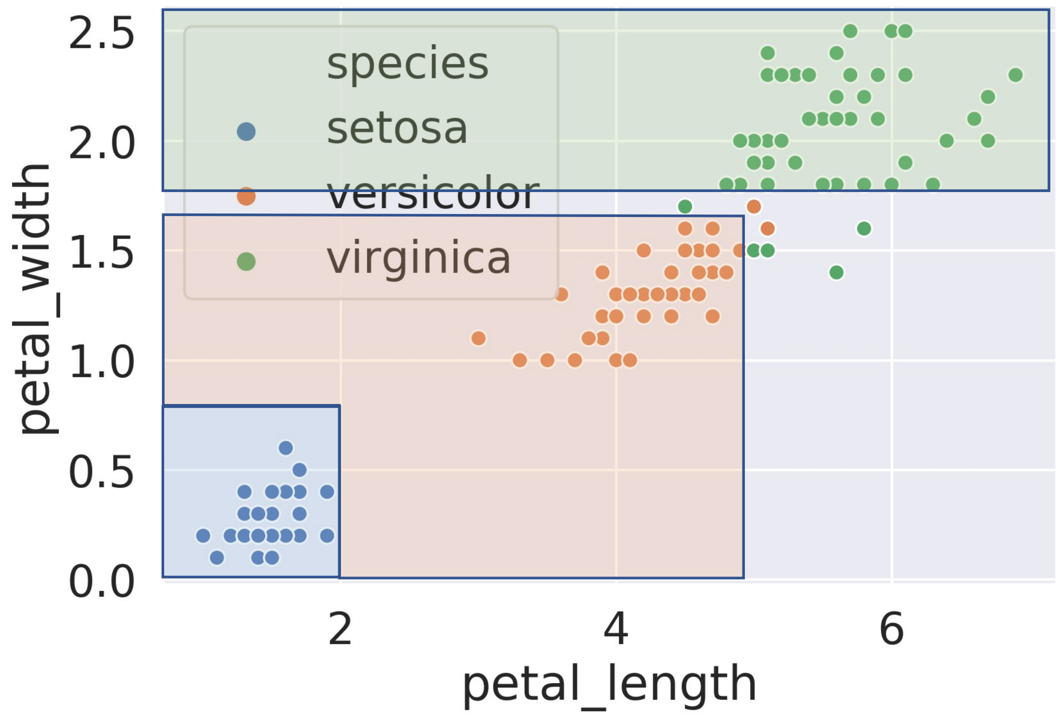
Example: Using Petal Data Only


Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
no
virginica
Is pedal_width <= 1.55 and pedal_length <= 4.95
yes
versicolor

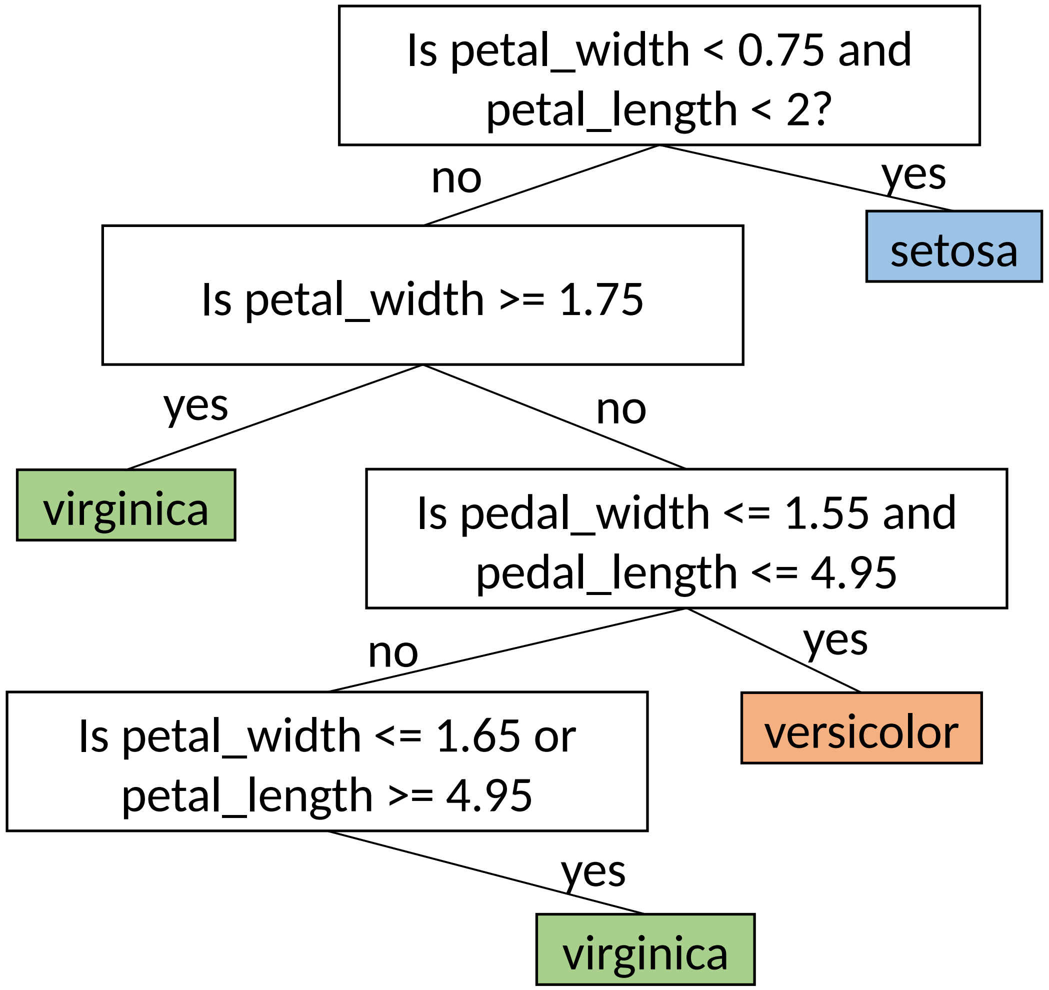


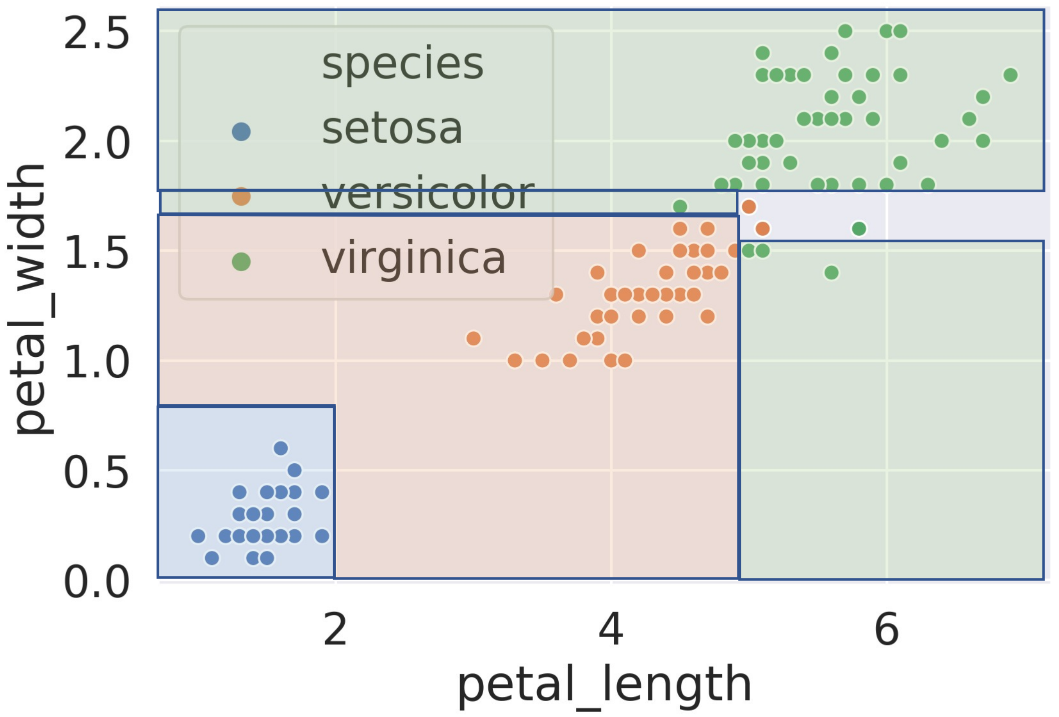
Example: Using Petal Data Only


Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
no
virginica
Is pedal_width <= 1.55 and pedal_length <= 4.95
no
yes
versicolor
Is petal_width <= 1.65 or petal_length >= 4.95

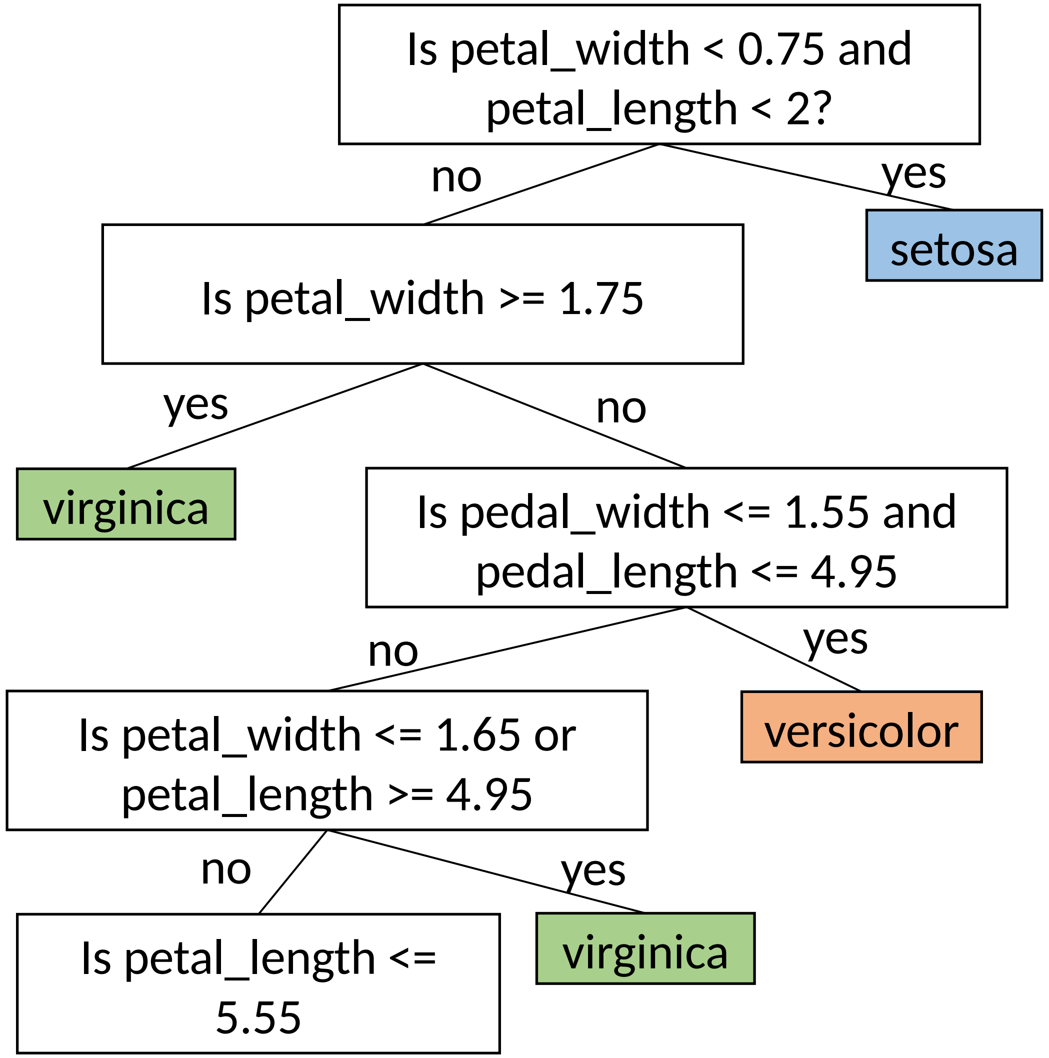


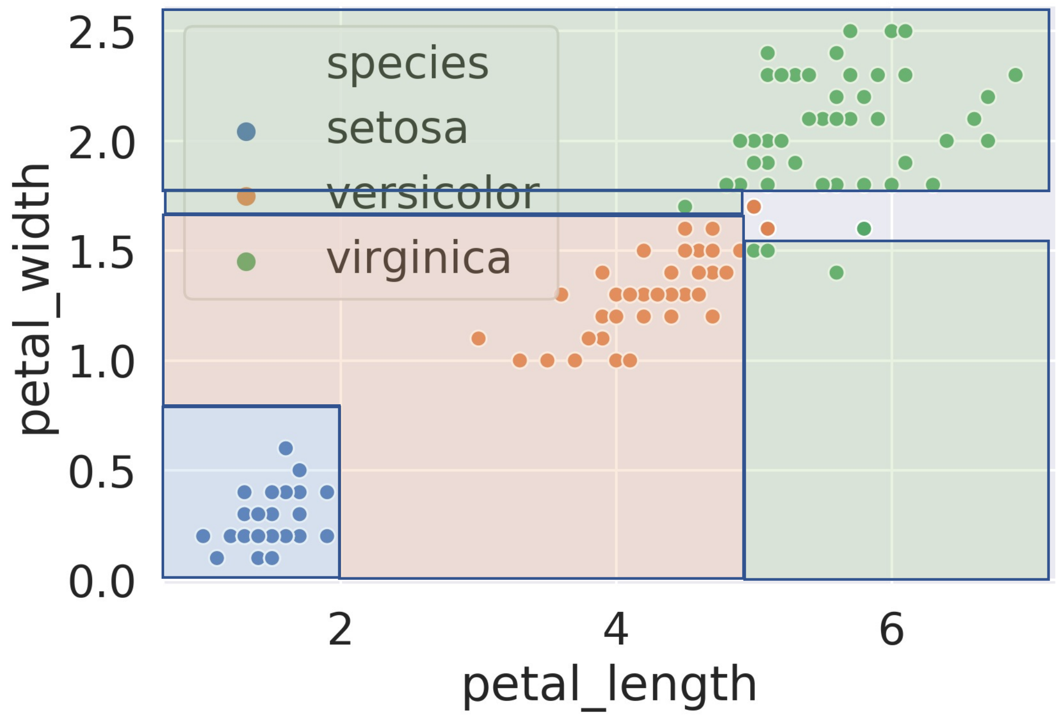
Example: Using Petal Data Only


Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
virginica
no
Is pedal_width <= 1.55 and pedal_length <= 4.95
no
yes
versicolor
Is petal_width <= 1.65 or petal_length >= 4.95
yes
virginica

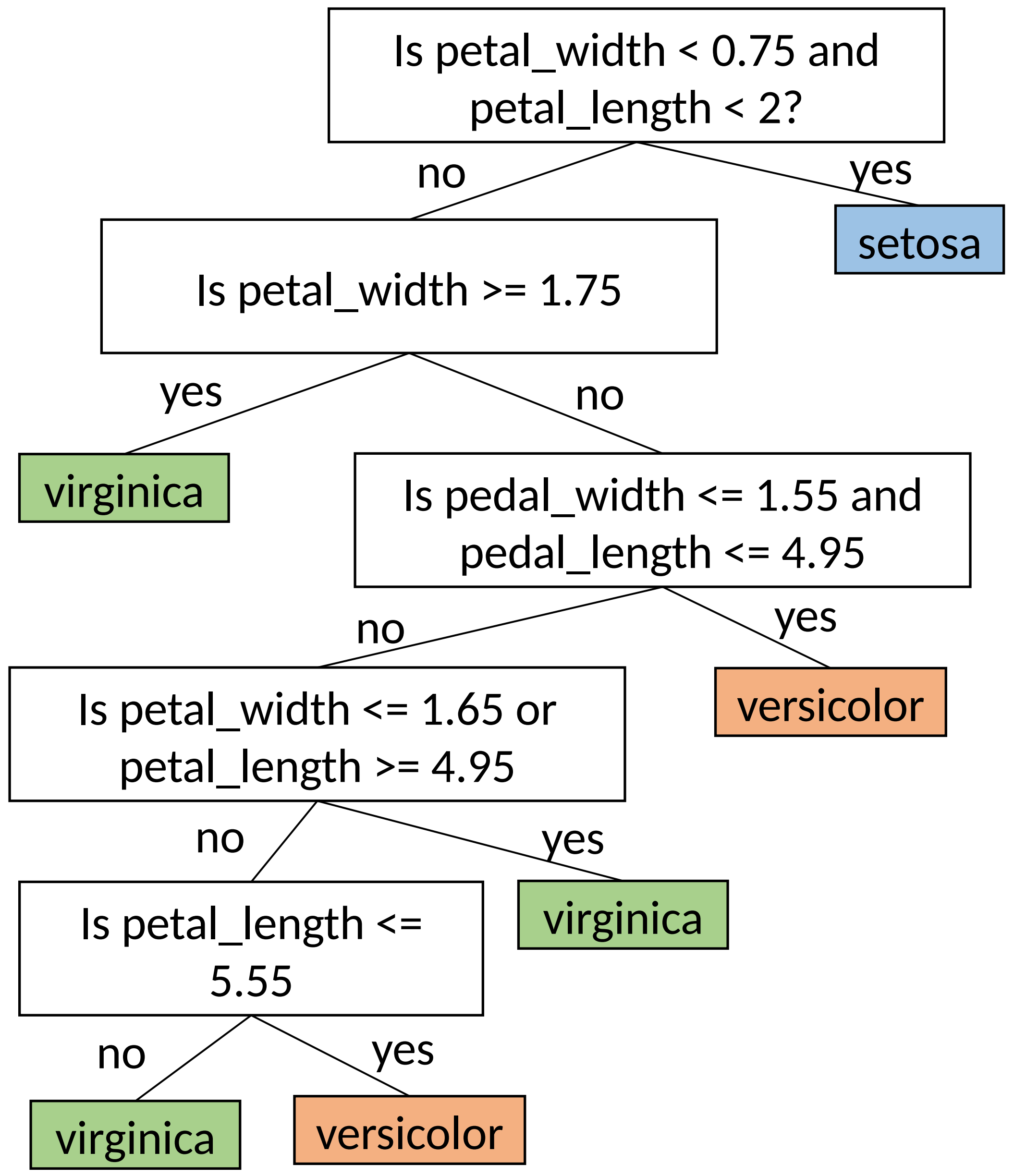


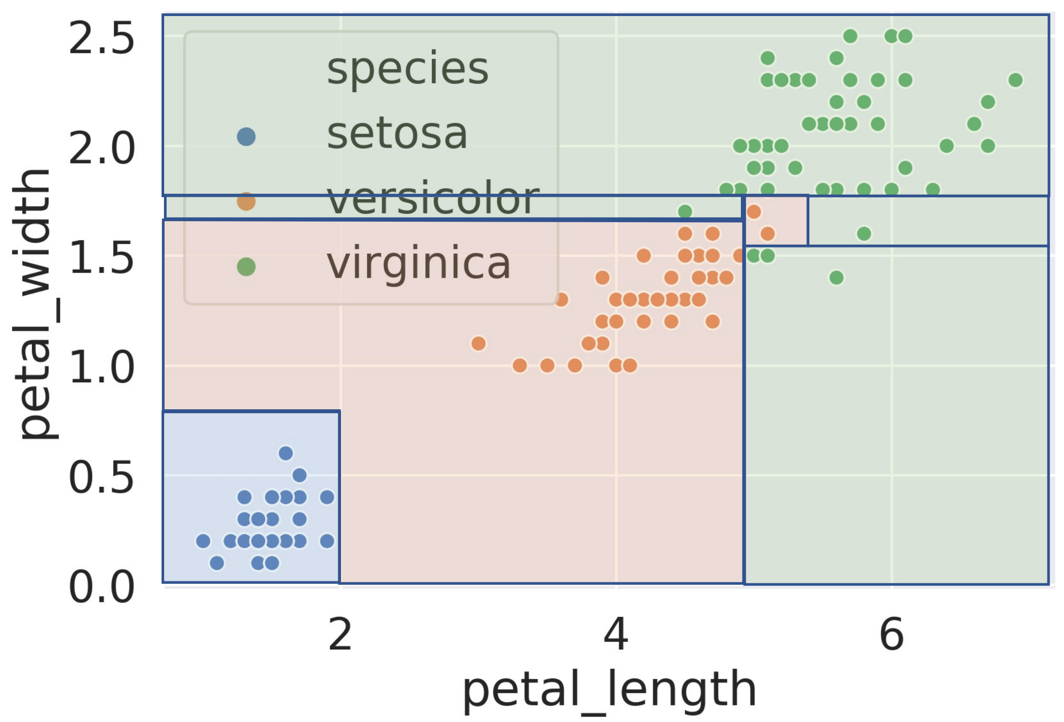
Example: Using Petal Data Only


Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
no
virginica
Is pedal_width <= 1.55 and pedal_length <= 4.95
no
yes
versicolor
Is petal_width <= 1.65 or petal_length >= 4.95
no
yes
virginica
Is petal_length <= 5.55



Example: Using Petal Data Only

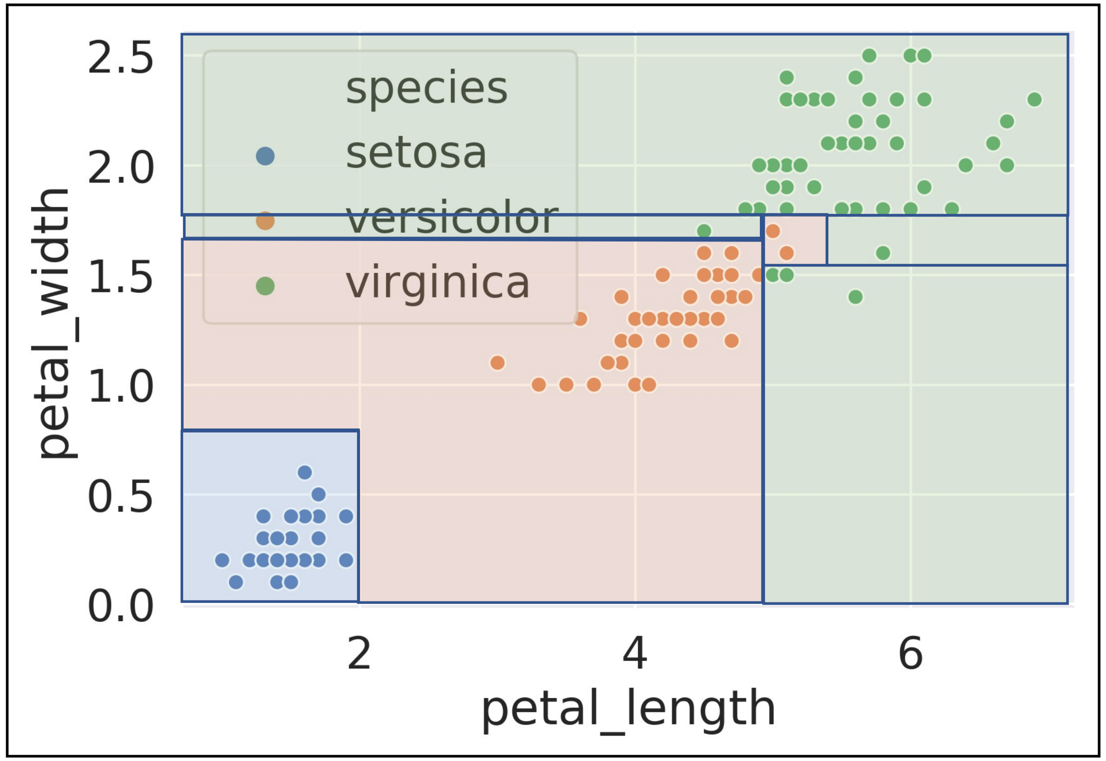

Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
no
virginica
Is pedal_width <= 1.55 and pedal_length <= 4.95
no
yes
versicolor
Is petal_width <= 1.65 or petal_length >= 4.95
no
yes
virginica
Is petal_length <= 5.55
no
yes
virginica
versicolor



Example: Using Petal Data Only

How accurate is our decision tree model on the training data?

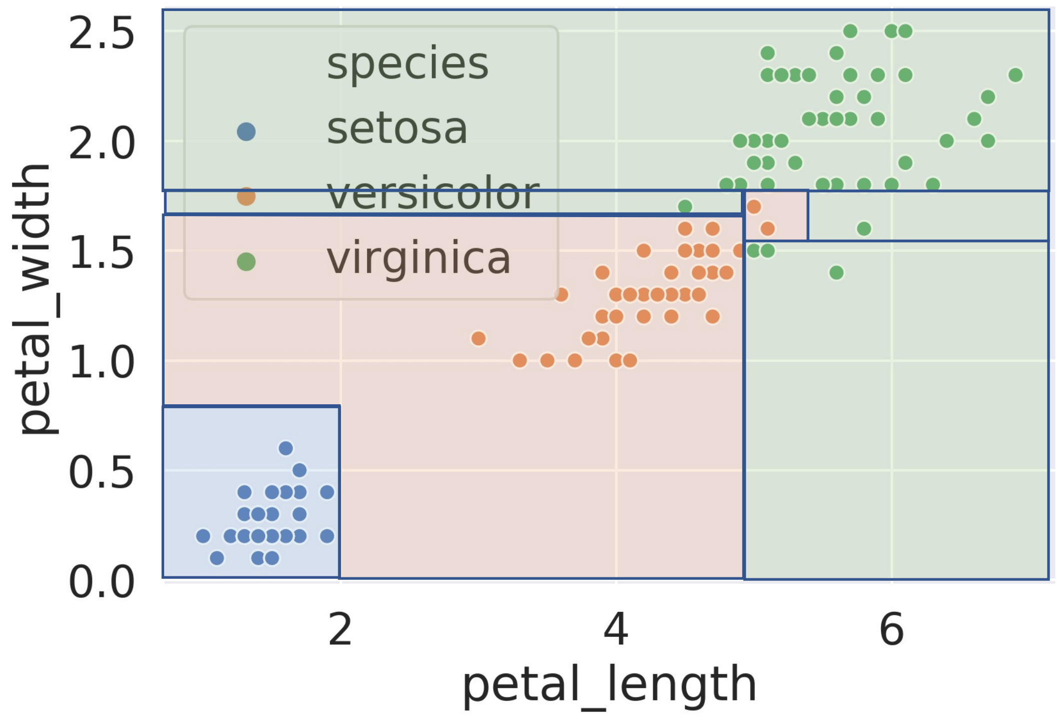
Is this good or bad?



Example: Using Petal Data Only

How accurate is our decision tree model on the training data?

- It seems like it gets every point correct



Is this good or bad?

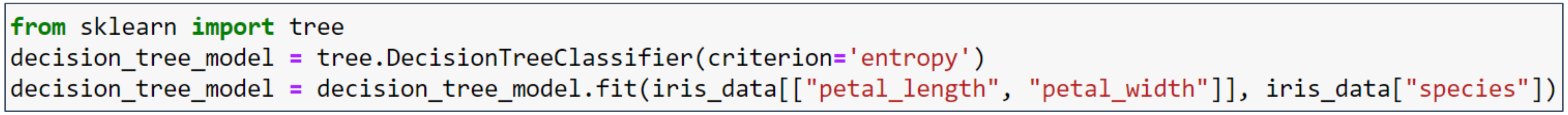
- I'd argue bad
- Seems likely to result in overfitting!

First, let's see how we can build decision trees for classification using `scikit-learn`

Decision Trees in scikit-learn

Decision Tree Models With scikit-learn

The code to build a decision tree model in `scikit-learn` is very similar to what we saw for building linear and logistic regression models:



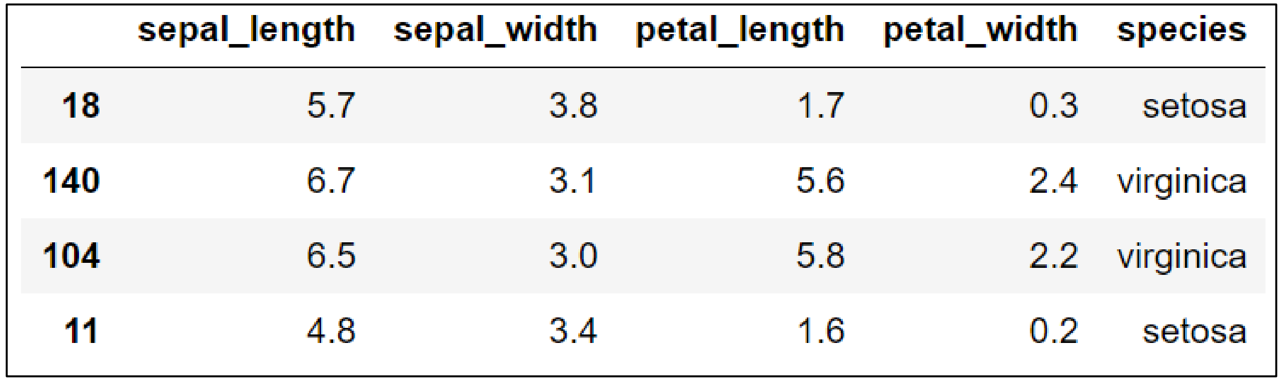
```python
from sklearn import tree
decision_tree_model = tree.DecisionTreeClassifier(criterion='entropy')
decision_tree_model = decision_tree_model.fit(iris_data[["petal_length", "petal_width"]], iris_data["species"])
```

Decision Tree Models With `scikit-learn`

The code to build a decision tree model in `scikit-learn` is very similar to what we saw for building linear and logistic regression models:

```python
from sklearn import tree
decision_tree_model = tree.DecisionTreeClassifier(criterion='entropy')
decision_tree_model = decision_tree_model.fit(iris_data[["petal_length", "petal_width"]], iris_data["species"])
```

```python
four_random_rows = iris_data.sample(4)
four_random_rows
```



	sepal_length	sepal_width	petal_length	petal_width	species
18	5.7	3.8	1.7	0.3	setosa
140	6.7	3.1	5.6	2.4	virginica
104	6.5	3.0	5.8	2.2	virginica
11	4.8	3.4	1.6	0.2	setosa

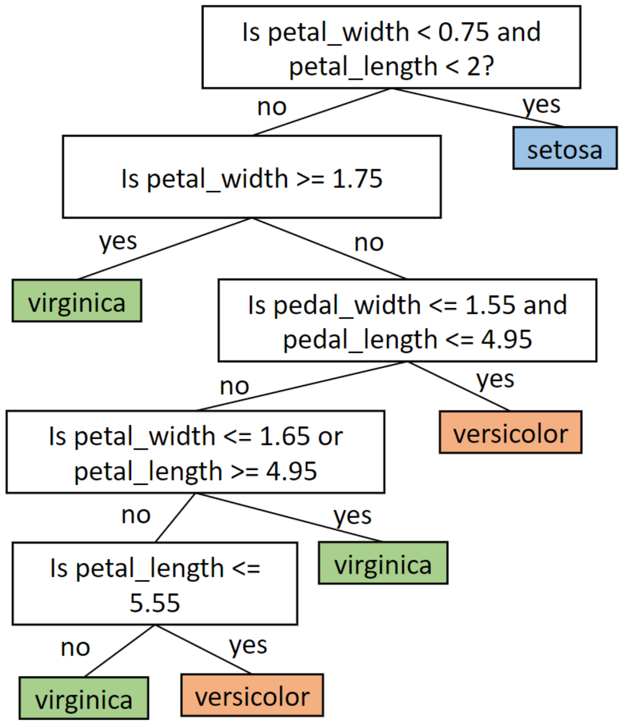
```python
decision_tree_model.predict(four_random_rows[["petal_length", "petal_width"]])
```

```
array(['setosa', 'virginica', 'virginica', 'setosa'], dtype=object)
```

Visualizing Decision Tree Models

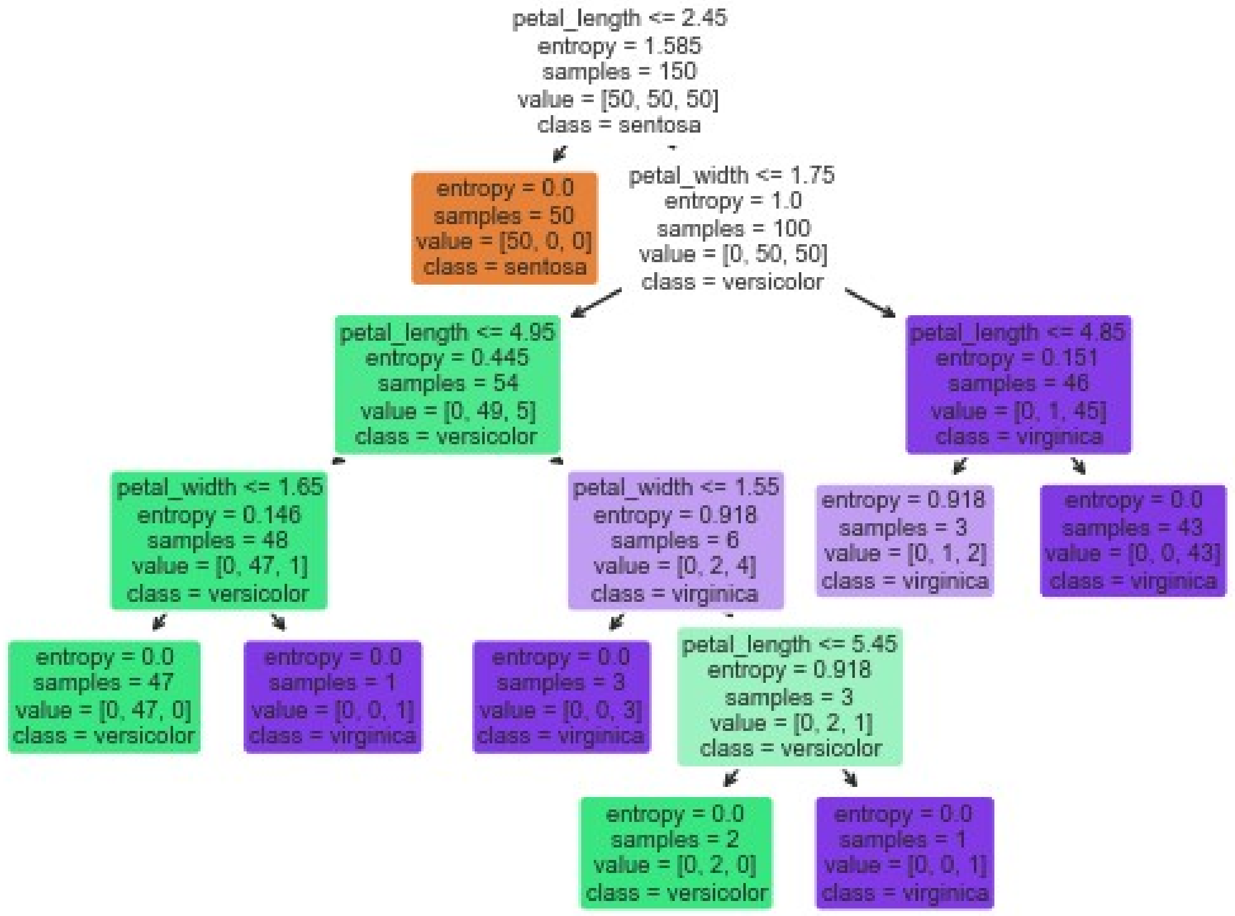
```
from sklearn import tree
decision_tree_model = tree.DecisionTreeClassifier(criterion='entropy')
decision_tree_model = decision_tree_model.fit(iris_data[["petal_length", "petal_width"]], iris_data["species"])
```

Suppose we want to visualize the decision tree, similar to what we saw earlier:


Is petal_width < 0.75 and petal_length < 2?
no
yes
setosa
Is petal_width >= 1.75
yes
no
virginica
Is pedal_width <= 1.55 and pedal_length <= 4.95
no
yes
versicolor
Is petal_width <= 1.65 or petal_length >= 4.95
no
yes
virginica
Is petal_length <= 5.55
no
yes
virginica
versicolor

Visualizing Decision Tree Models – default

```
from sklearn import tree
decision_tree_model = tree.DecisionTreeClassifier(criterion='entropy')
decision_tree_model = decision_tree_model.fit(iris_data[["petal_length", "petal_width"]], iris_data["species"])
```

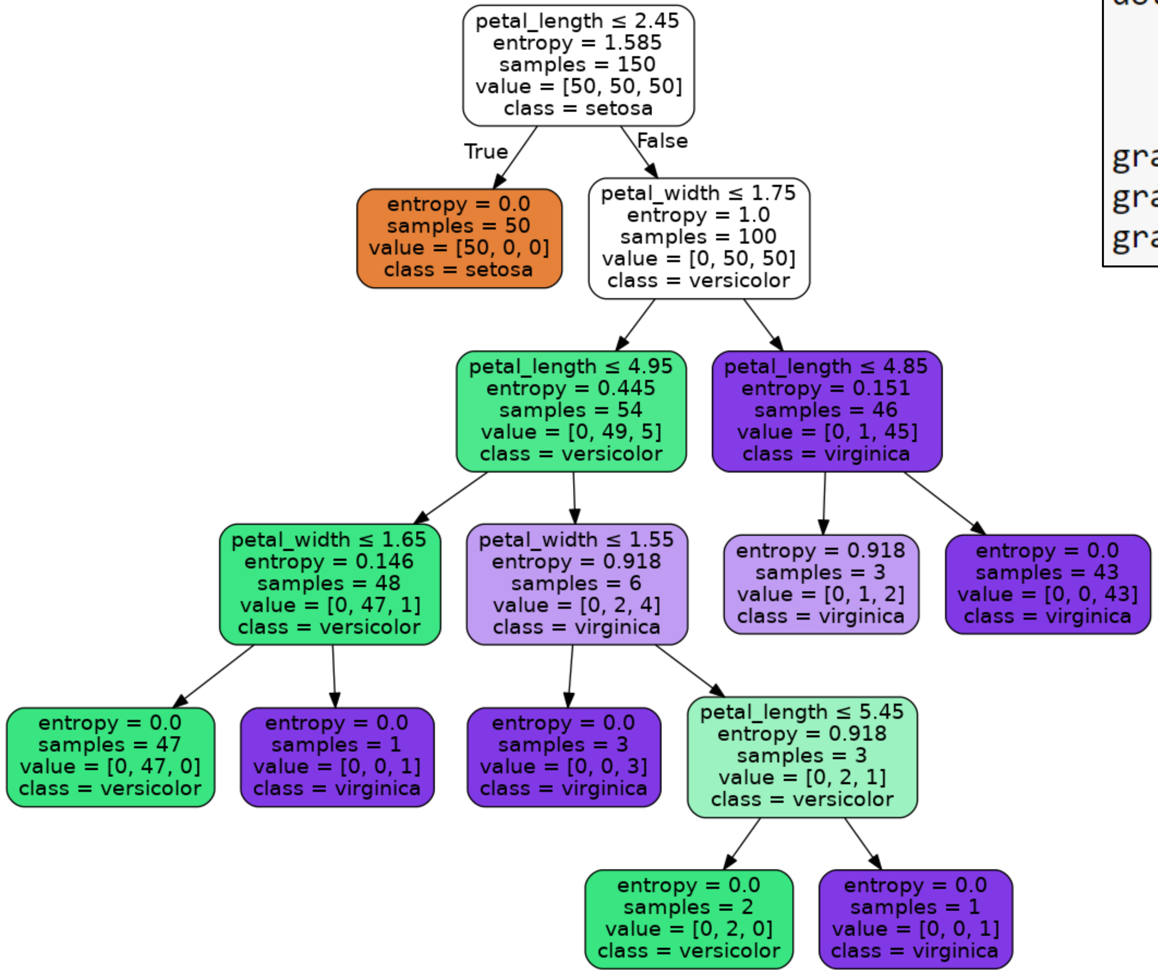


There is a built in `DecisionTree` visualizer

- Unfortunately, it isn't very good

Visualizing Decision Tree Models - use GraphViz

Can use GraphViz to get a much nicer picture.



```python
import graphviz
dot_data = tree.export_graphviz(decision_tree_model, out_file=None,
                                feature_names=["petal_length", "petal_width"],
                                class_names=["setosa", "versicolor", "virginica"],
                                filled=True, rounded=True)
graph = graphviz.Source(dot_data)
graph.render(format="png", filename="iris_tree")
graph
```

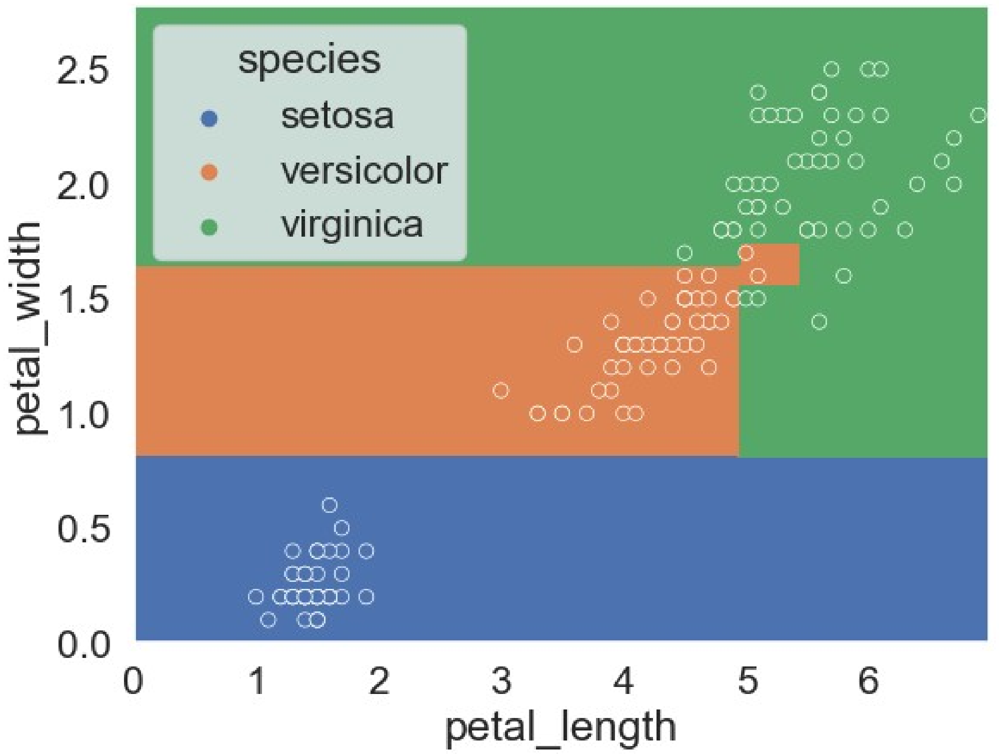
In each box, we see:

- The rule.
- The entropy at that node (more later).
- The number of samples that remain after applying all of the above rules.
- The number of samples that remain.
- The most likely class.

Visualizing Decision Boundary for Decision Tree

Plotting the decision boundaries for decision tree models, we get the results below.

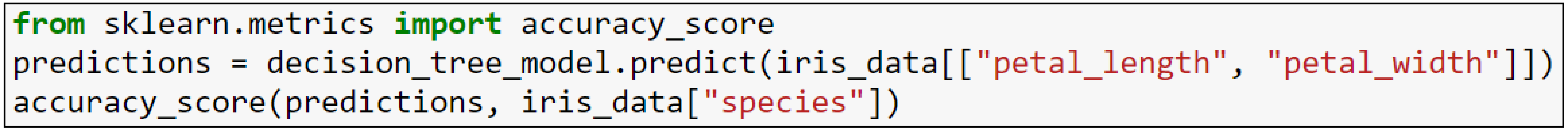
- Decision tree has nonlinear boundary, and appears to get 100% accuracy.
  - Let’s calculate the exact accuracy rather than just relying on our eyes to look at a somewhat complex visualization.



Measuring the Performance of Our Model

Running the code below, we see that we only get 99.3% accuracy

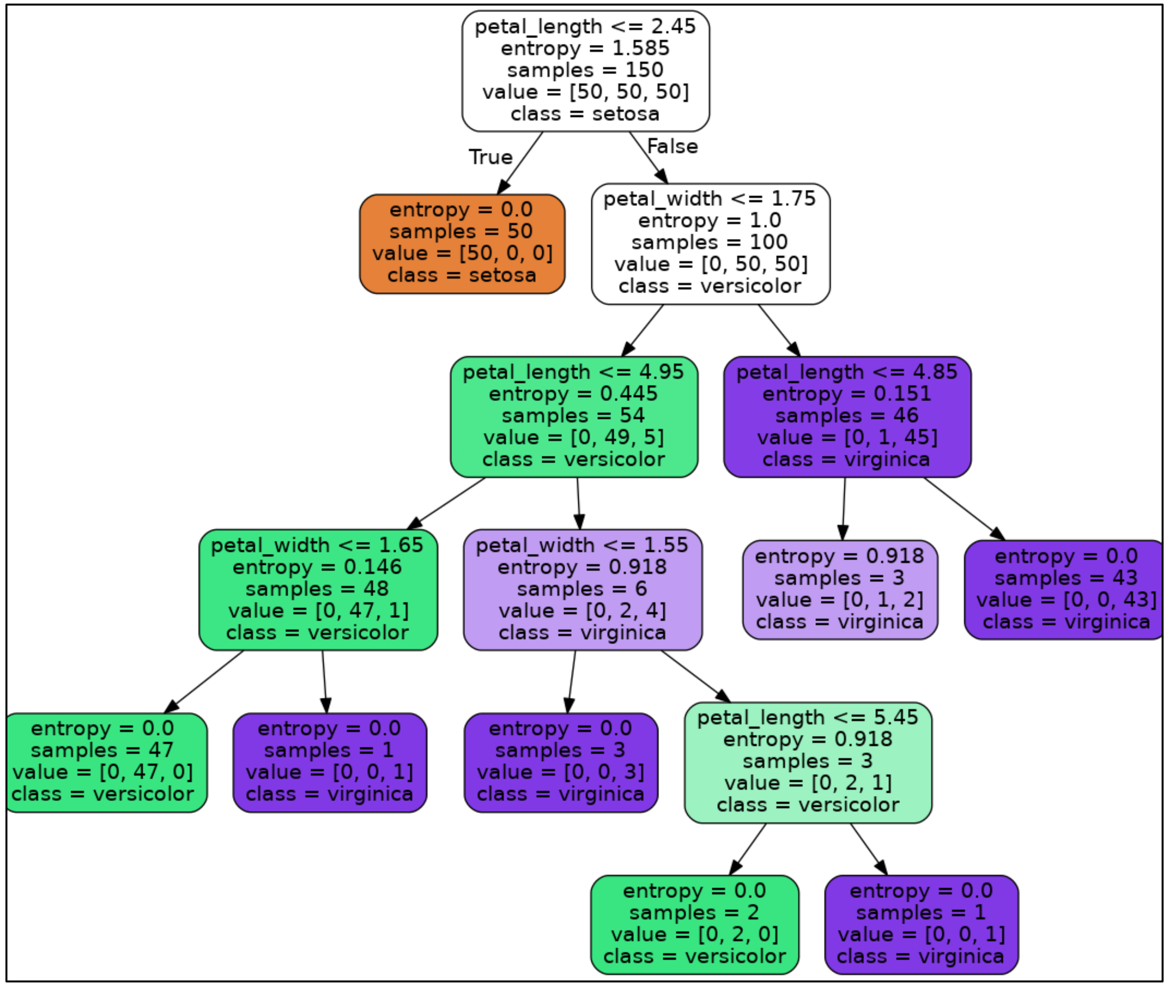
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html



```
from sklearn.metrics import accuracy_score
predictions = decision_tree_model.predict(iris_data[["petal_length", "petal_width"]])
accuracy_score(predictions, iris_data["species"])
```

To understand why, let’s look back at our decision tree model.

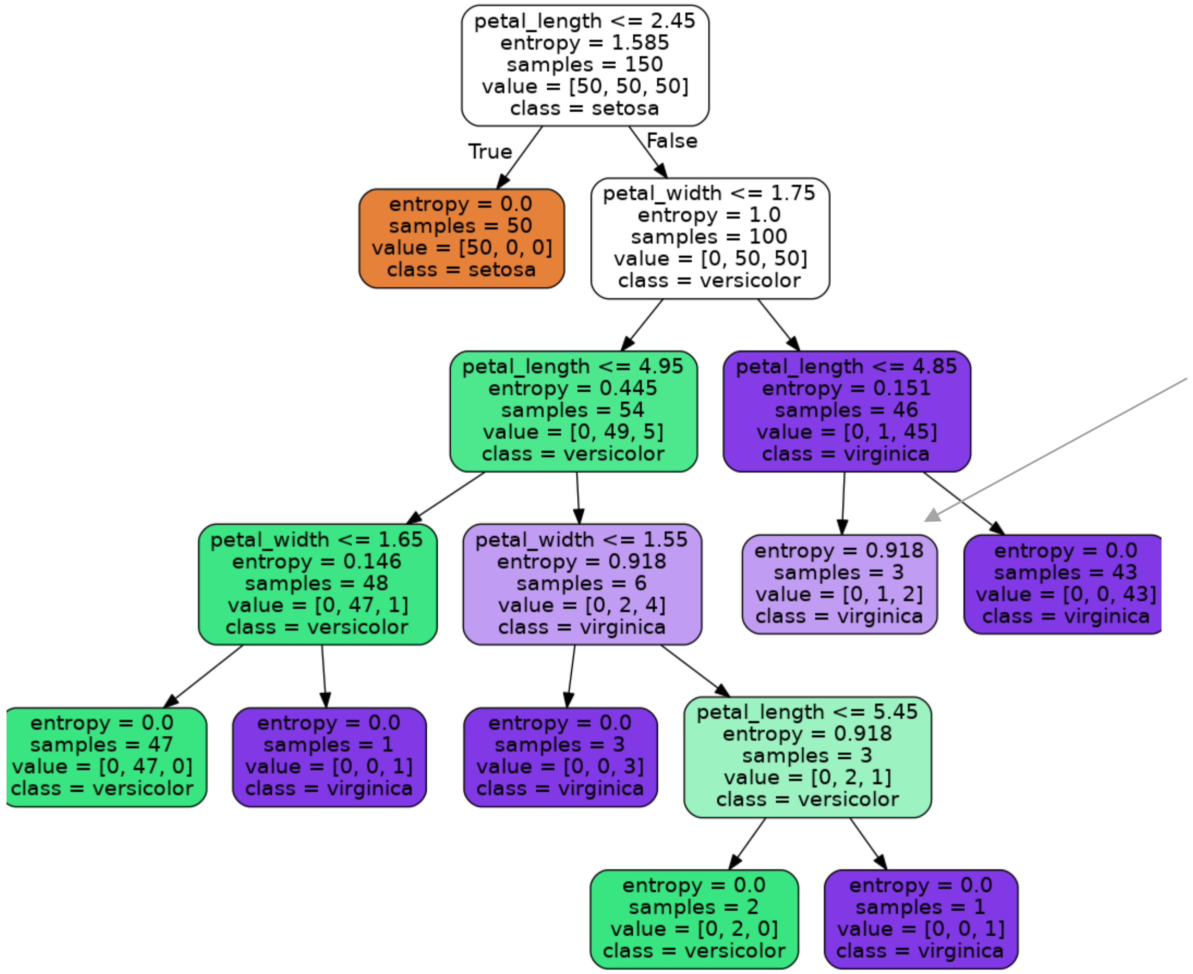
Understanding Our Decision Tree



There is one terminal decision point where there is more than one possible right answer.

Can you find it?

Understanding Our Decision Tree



There is one terminal decision point where there is more than one possible right answer.

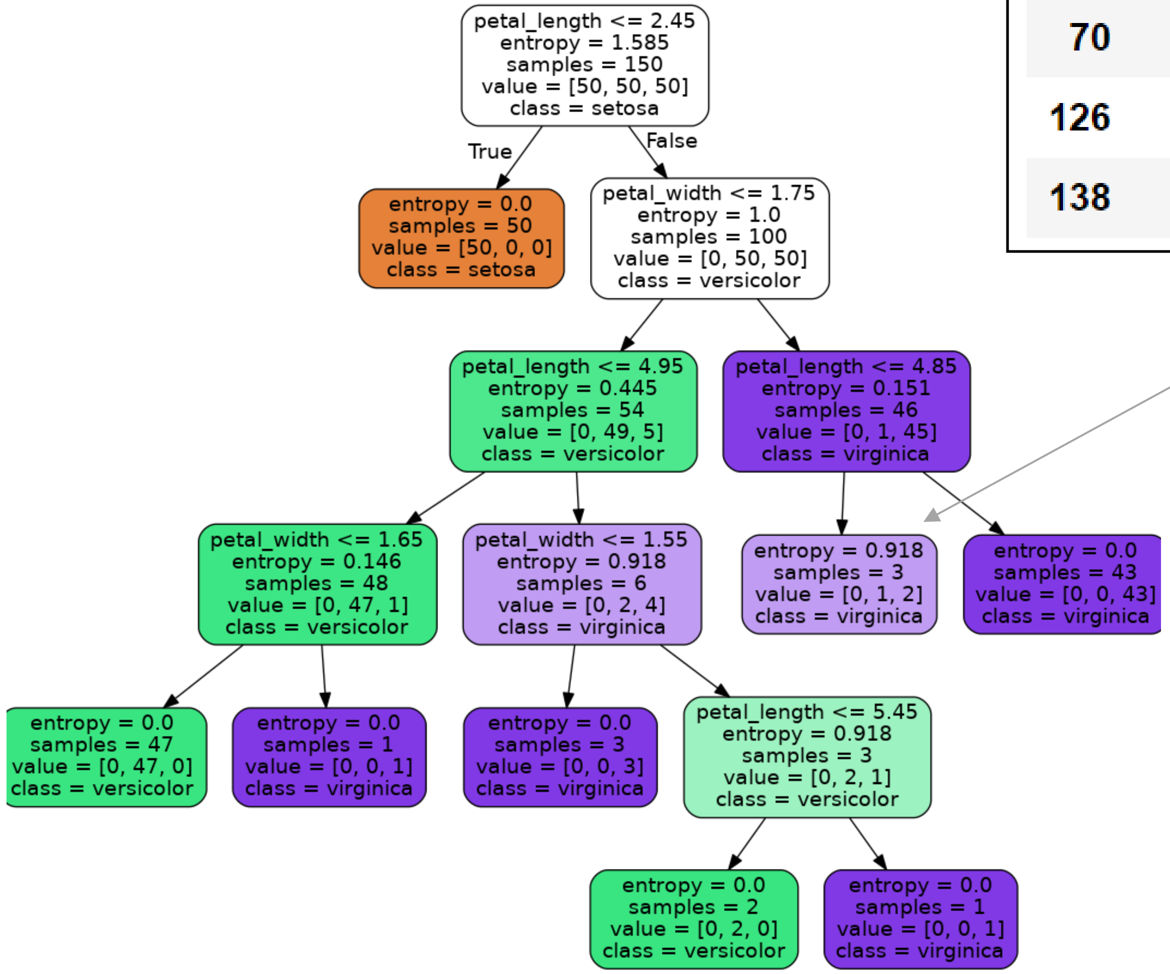
The model was unable to come up with a decision rule to resolve these last 3 samples.

Let’s see why using the query method of the dataframe class.

Understanding Our Decision Tree

```
iris_data.query("petal_length > 2.45 and petal_width > 1.75 and petal_length < 4.85")
```

	sepal_length	sepal_width	petal_length	petal_width	species
70	5.9	3.2	4.8	1.8	versicolor
126	6.2	2.8	4.8	1.8	virginica
138	6.0	3.0	4.8	1.8	virginica

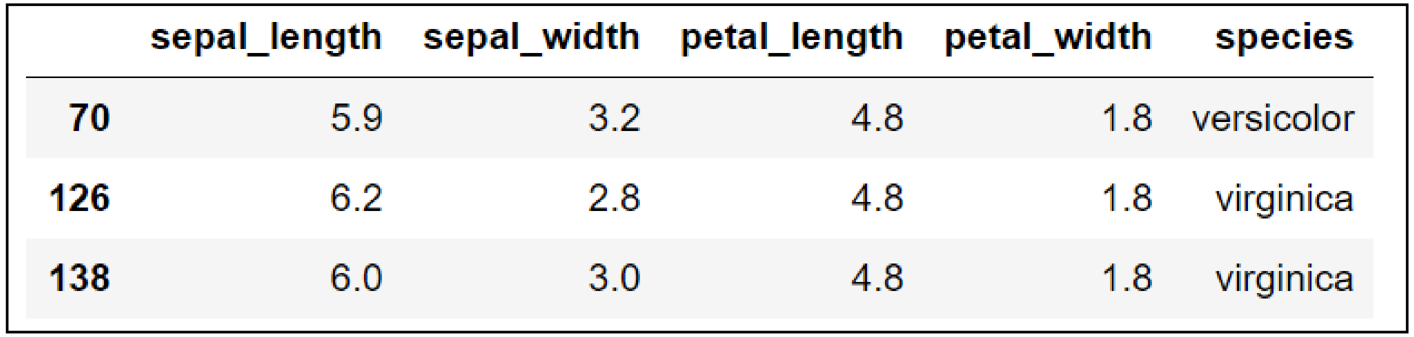


There is one terminal decision point where there is more than one possible right answer.

- In the original data set, there was a versicolor iris with the same petal measurements as two virginicas.

Overfitting

Overfitting and Decision Trees



	sepal_length	sepal_width	petal_length	petal_width	species
70	5.9	3.2	4.8	1.8	versicolor
126	6.2	2.8	4.8	1.8	virginica
138	6.0	3.0	4.8	1.8	virginica

sk-learn decision trees will always have perfect accuracy on the training data, EXCEPT when there are samples from different categories with the exact same features.

- Example: If the versicolor above had a petal_length of 4.8001, we'd have 100% training accuracy.

This tendency for perfect accuracy should give us concern about overfitting.

- Model is extremely sensitive / has high variance.

Sepal Decision Tree

In order to understand how to avoid overfitting, let's first discuss how decision trees are created from data.

Traditional decision tree generation algorithm:

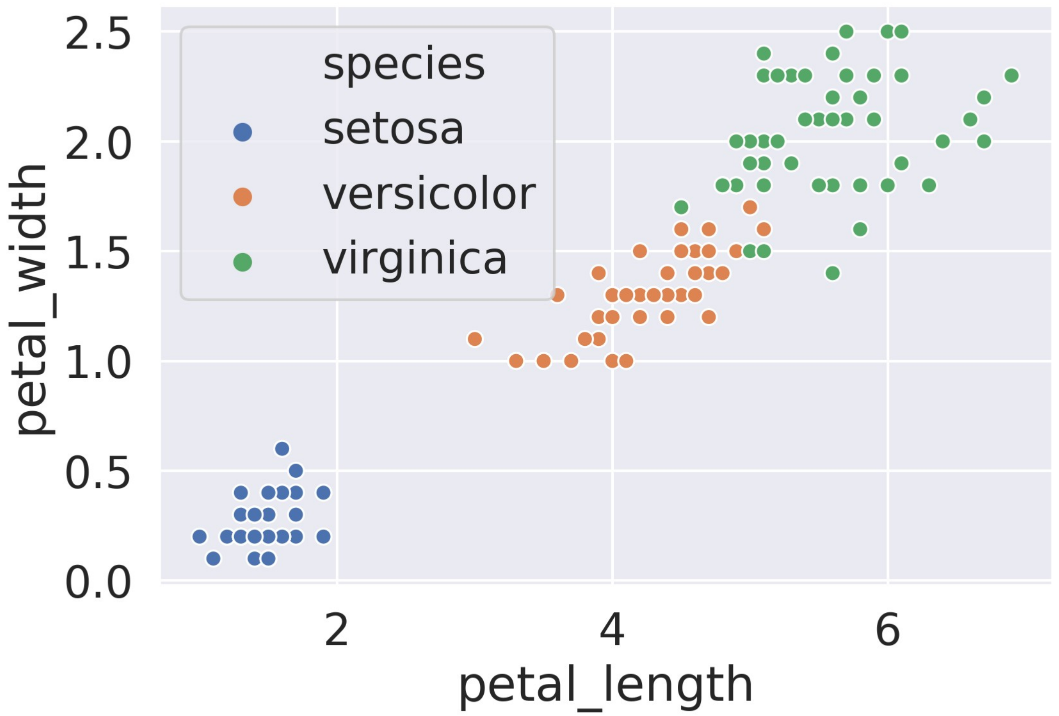
- All of the data starts in the root node.
- **Repeat** until every node is either **pure** or **unsplittable**:
  - **Pick the best feature** x and **best split value** β, e.g. x = petal_length, β = 2.
  - **Split data into two nodes**, one where x < β, and one where x ≥ β.

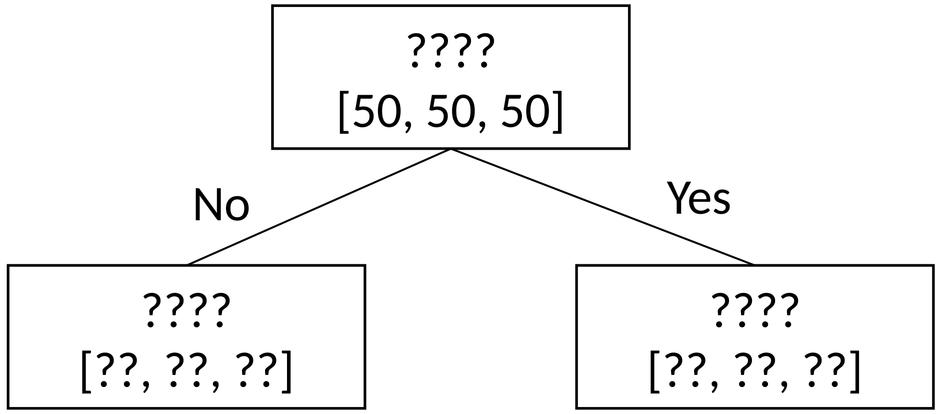
Notes: A node that has only one samples from one class is called a "**pure**" node. A node that has overlapping data points from different classes and thus that cannot be split is called "**unsplittable**".

Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?

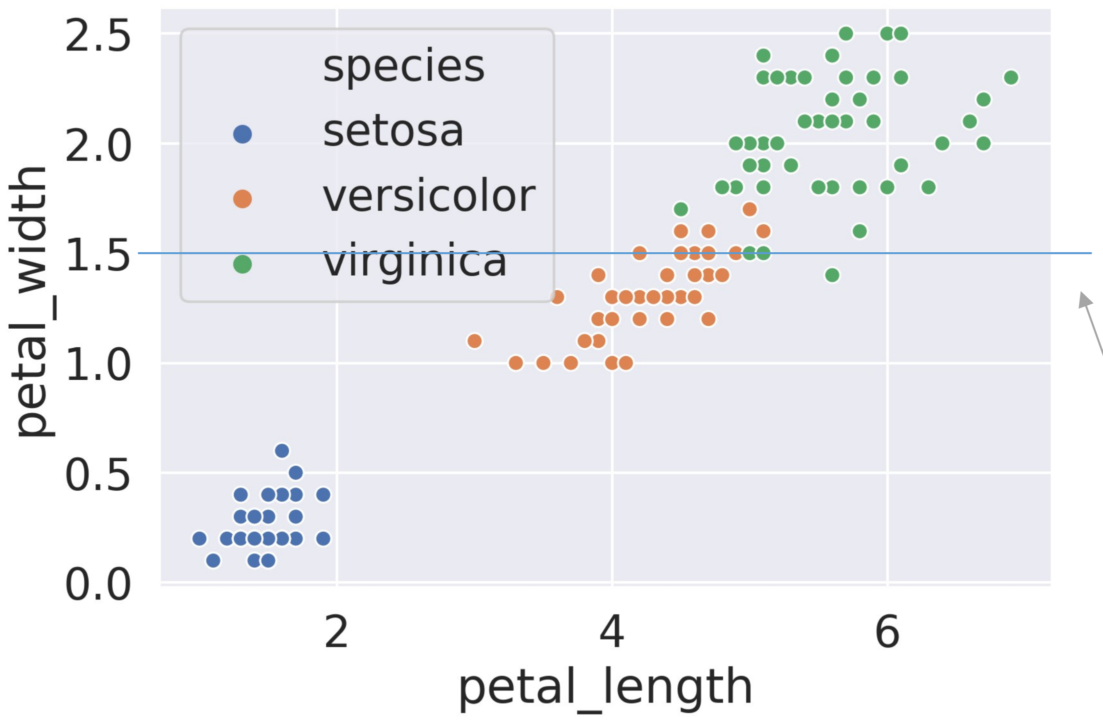


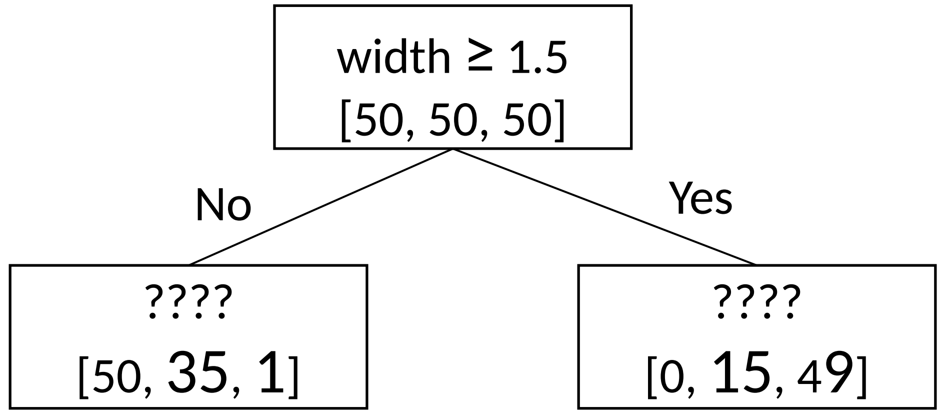


Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?



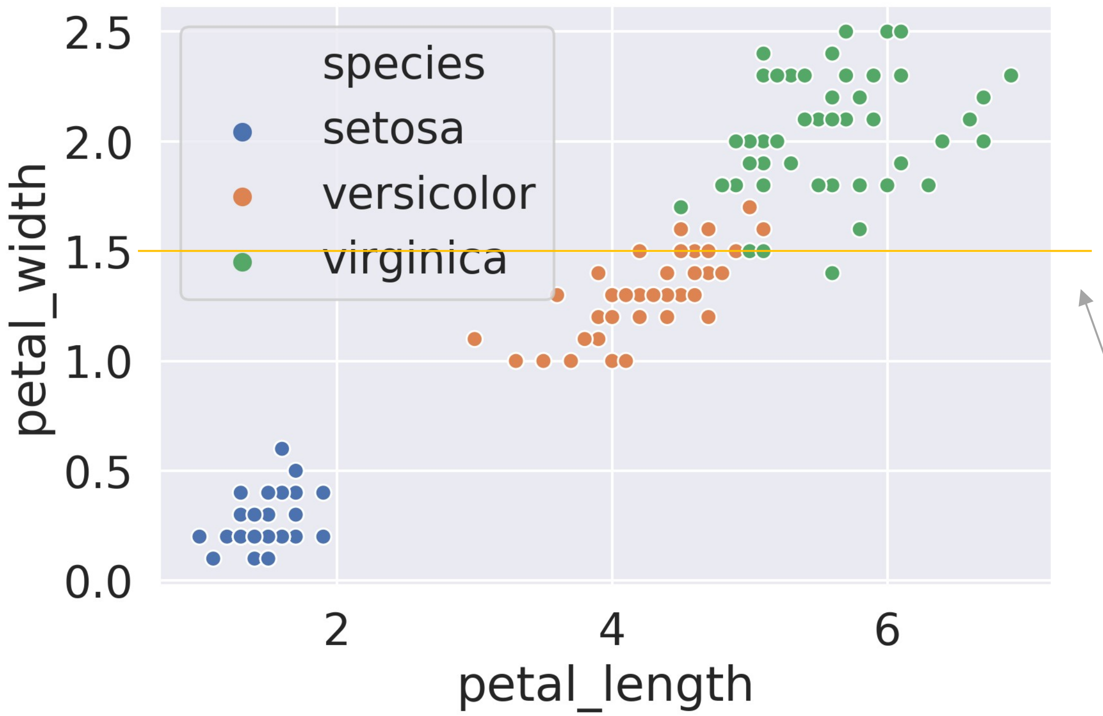


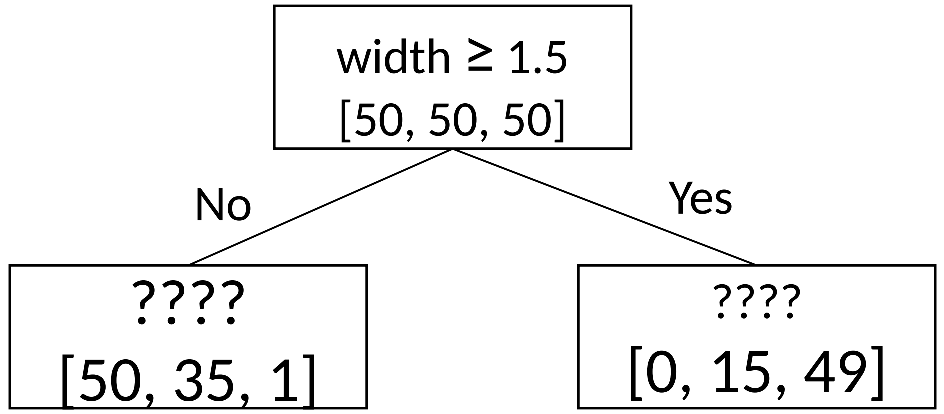
Is this good?

Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?





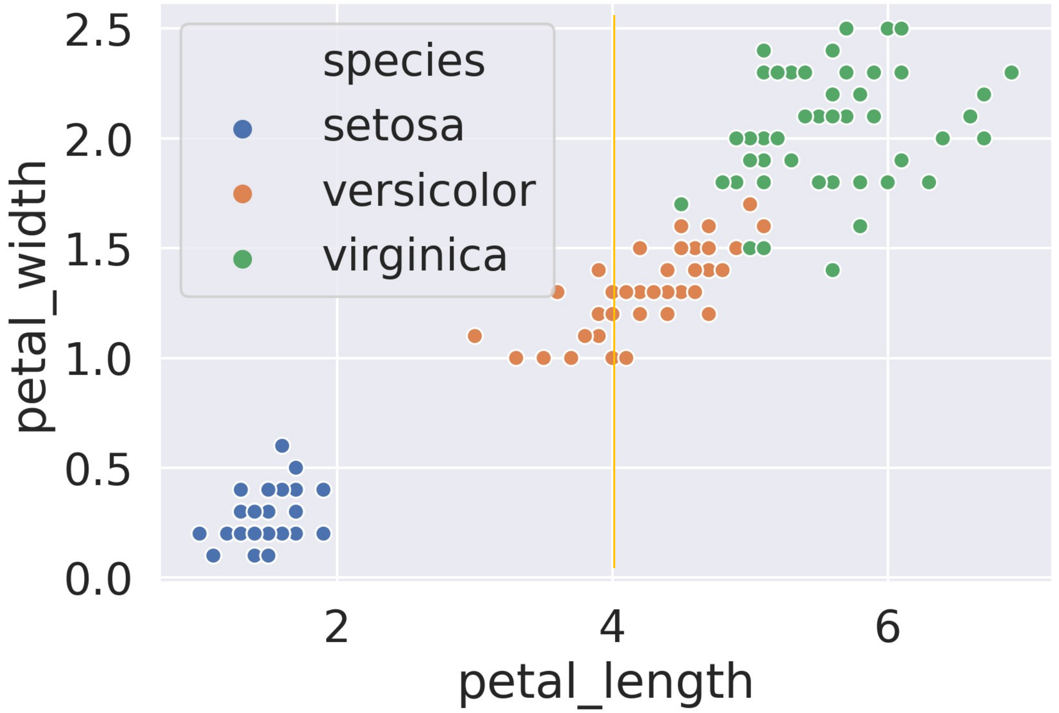
Is this good? It does help, but we could do better!

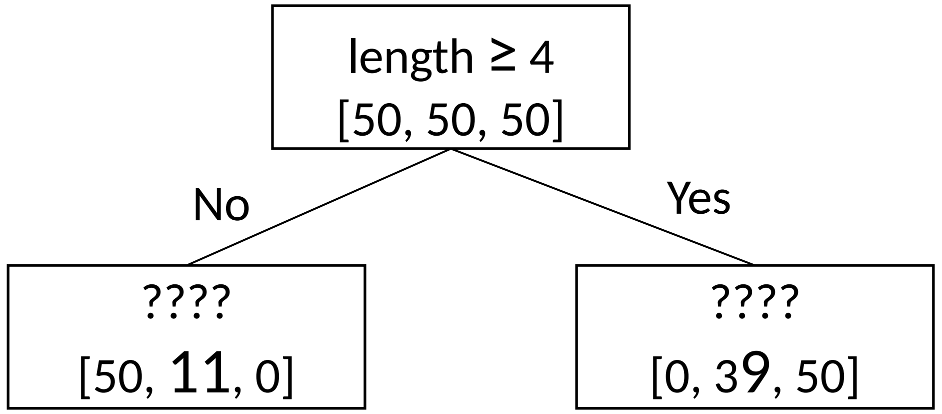
Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?

Better than before??

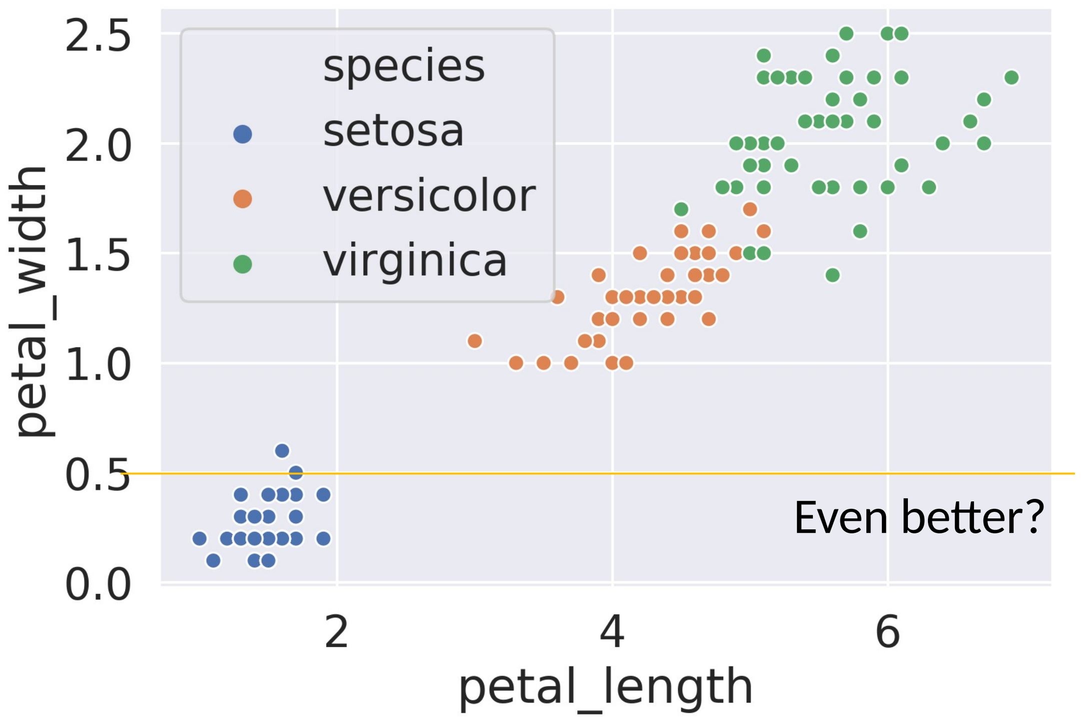


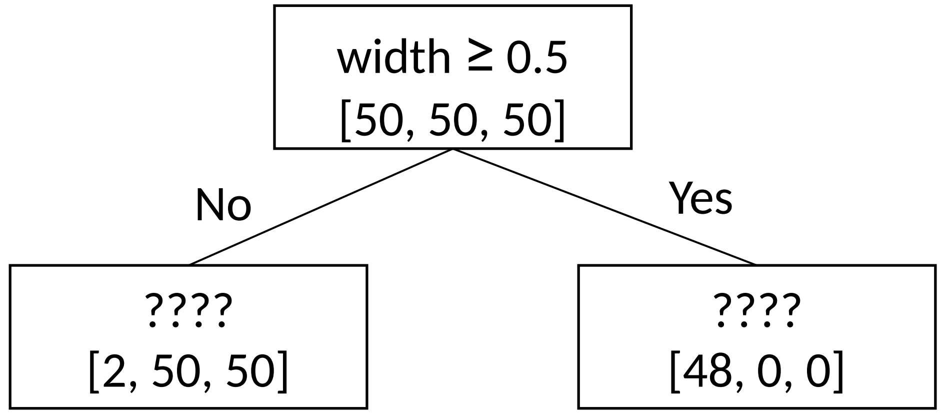


Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?



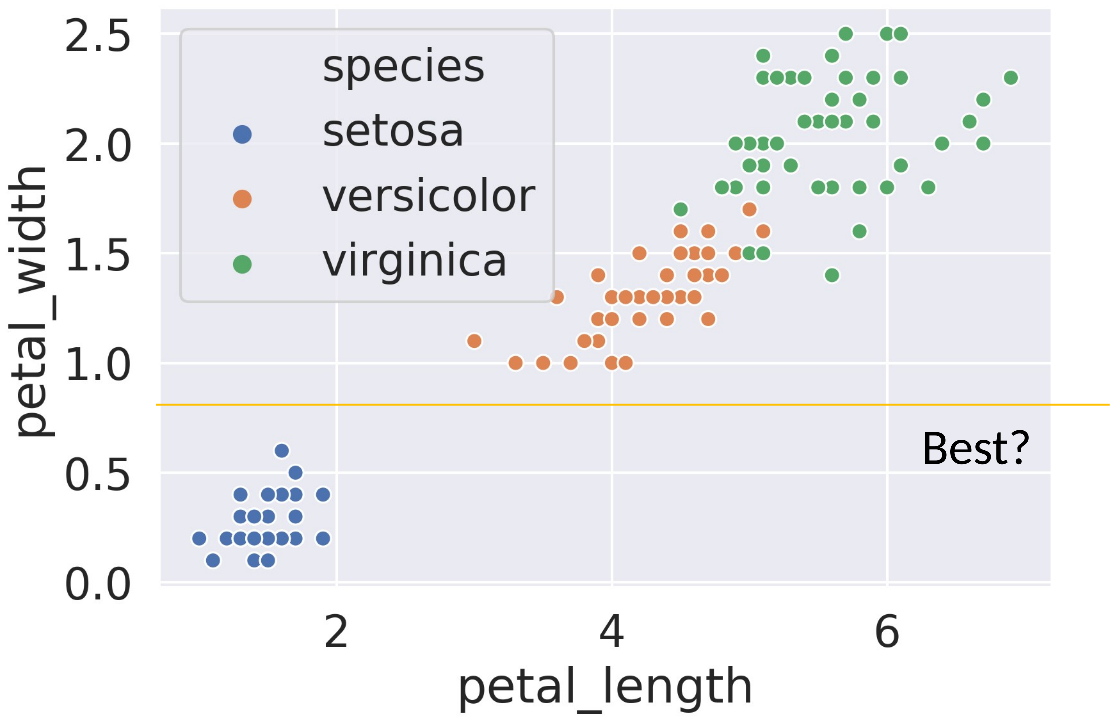


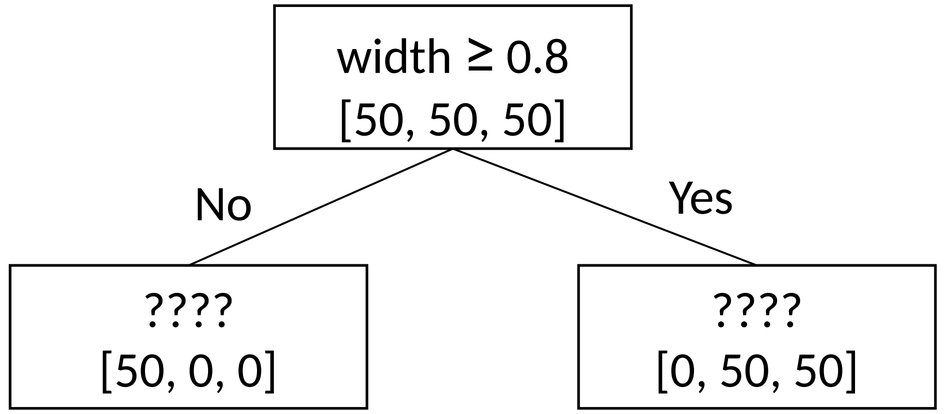
Defining a Best Feature

Question: Which feature and split value is best?

- Equivalently: Which horizontal or vertical line do we want to draw?

We need some sort of rigorous definition for a good split.





Node Entropy

Let p_C be the proportion of data points in a node with label C.

For example, for the node at the top of the decision tree, p_0 = 34/110 = 0.31, p_1 = 36/110 = 0.33, and p_2 = 40/110 = 0.36.

Define the entropy S of a node (in bits) as:

$$S = -\sum_C p_C \log_2 p_C$$

For example, S for the top node is:

$-0.31 \log_2 0.31 - 0.33 \log_2 0.33 - 0.36 \log_2 0.36 = 0.52 + 0.53 + 0.53 = 1.58$ bits

Test Your Understanding

What is the entropy of the node with [31, 4, 1] in each class?

- p_0 = 31/36 = 0.86, p_1 = 4/36 = 0.11, and p_2 = 1/36 = 0.028
- S = $-0.86 \log_2 0.86$
  $- 0.11 \log_2 0.11$
  $- 0.028 \log_2 0.028$ = 0.68 bits

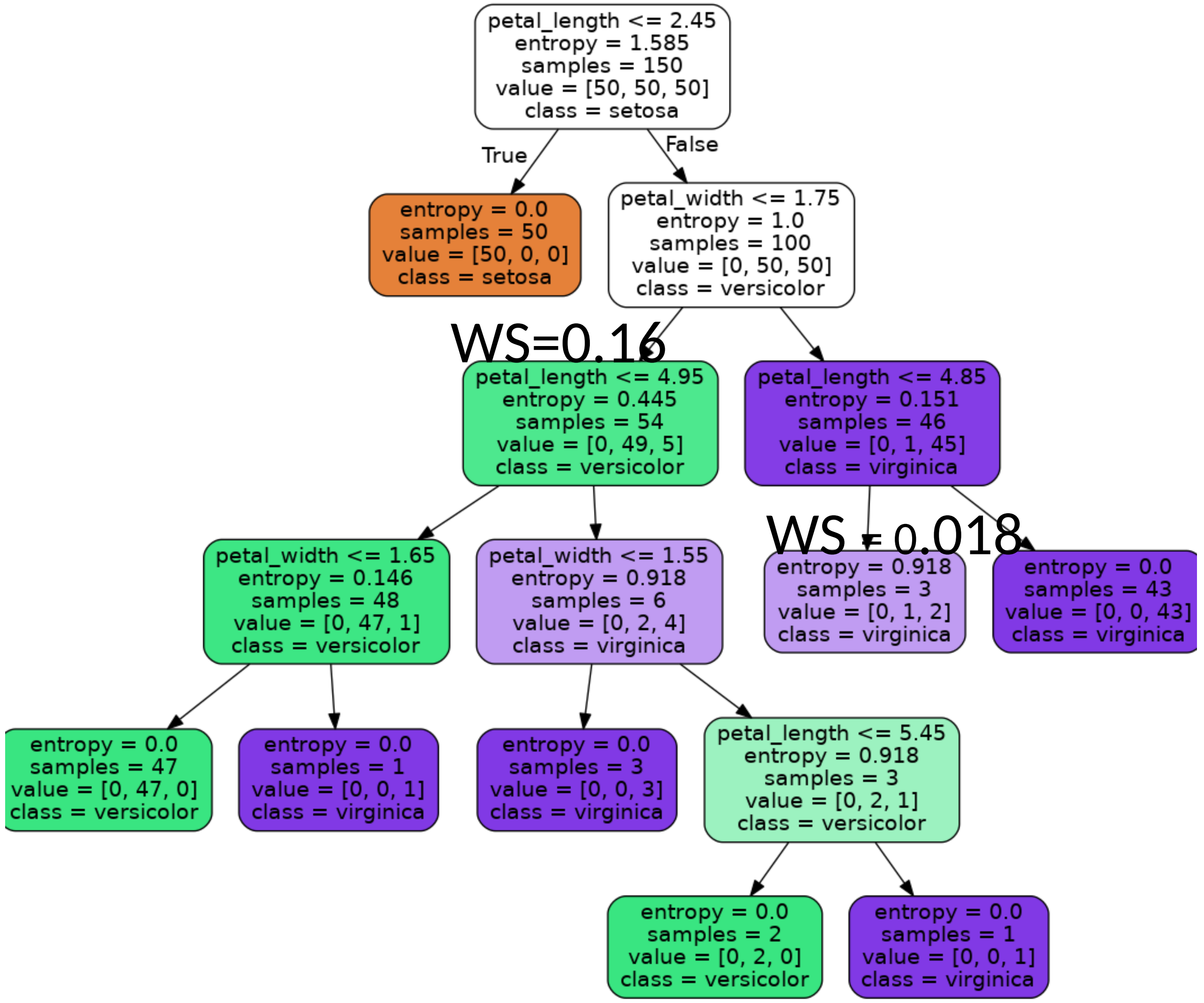
Can think of entropy as how unpredictable a node is. Low entropy means more predictable. High entropy means less predictable.

Weighted Entropy

Define the weighted entropy of a node as its entropy scaled by the fraction of the samples in that node.

Consider our decision tree to the right.

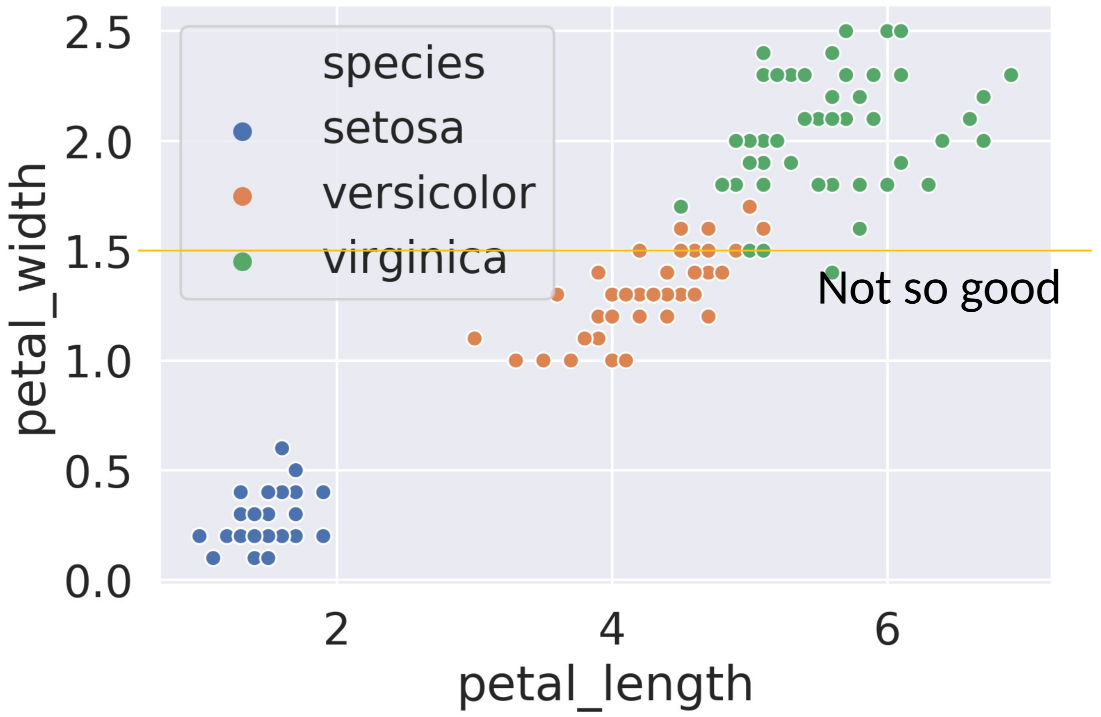
- Weighted entropy of the node with 54 samples is WS = 54/150 * 0.445 = 0.16.
- Weighted entropy of the terminal node with 3 samples is 3/150 * 0.918 = 0.018

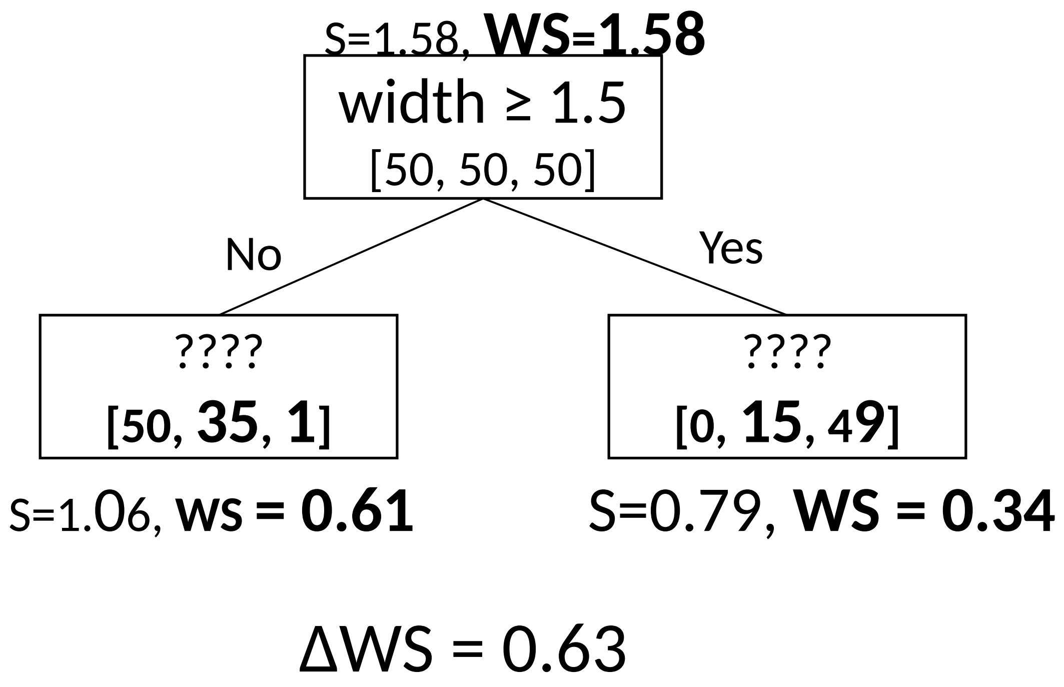


Defining a Best Feature

Split choice #1: width ≥ 1.5. Compute entropy of child nodes:

- entropy([50, 50, 50]) =1 , weighted entropy = 150/150 * 1 = 1.58
- entropy([50, 35, 1]) = 1.06, weighted entropy = 86/150 * 1.06 = 0.61
- entropy([15, 49]) = 0.79, weighted entropy = 64/150 * 0.79 = 0.34
- ΔWS = **1.58 - 0.61 - 0.34** = 0.63.

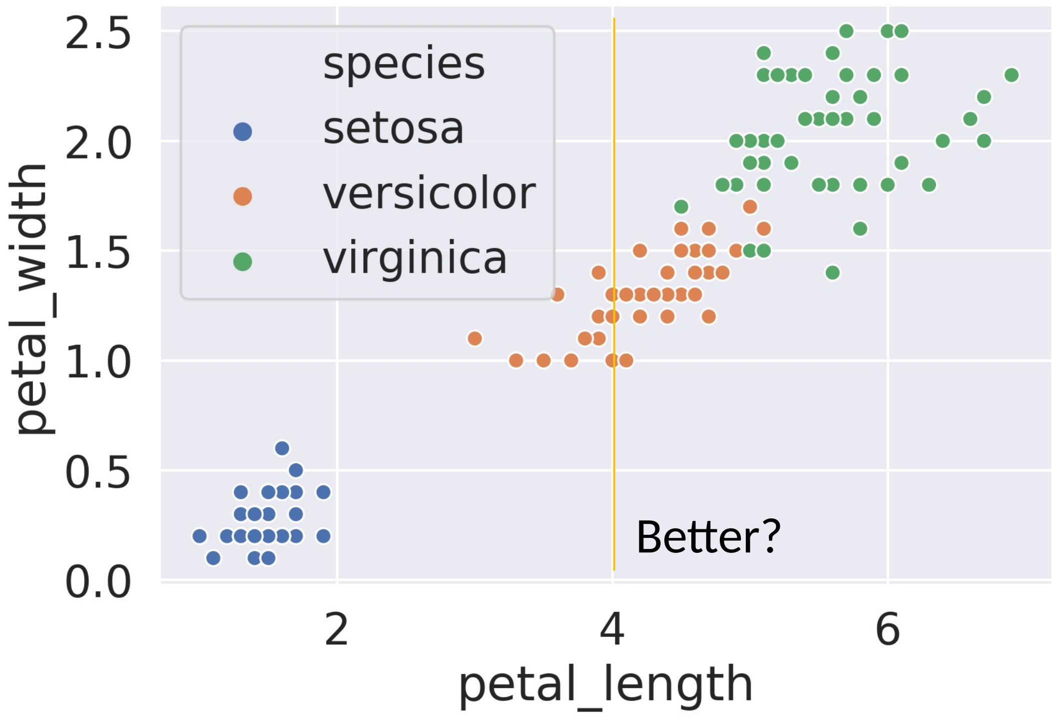


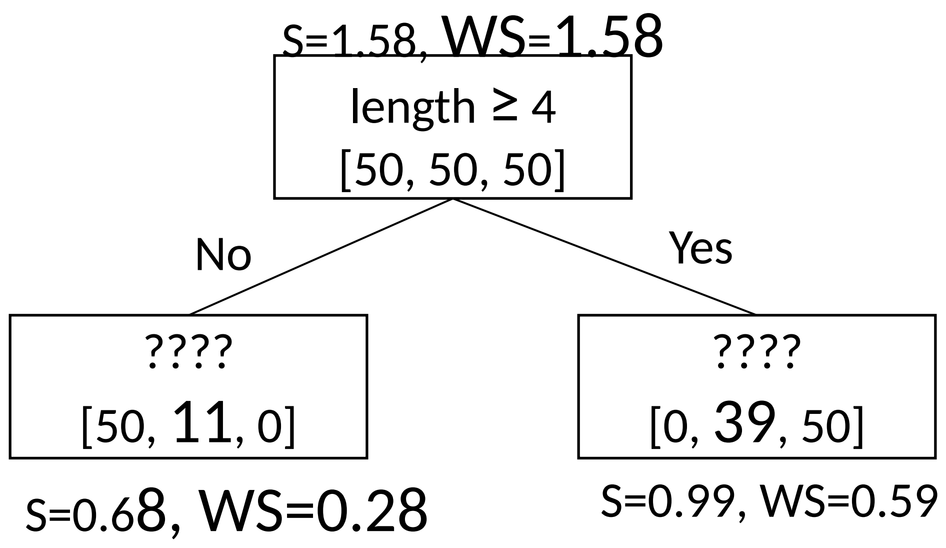


Defining a Best Feature

Split choice #2: length ≥ 4. Compute entropy of child nodes:

- entropy([50, 11]) = 0.68, weighted entropy = 61/150 * 0.68 = 0.28
- entropy([39, 50]) = 0.99, weighted entropy = 89/150 * 0.99 = 0.59
- ΔWS = **1.58 - 0.28 - 0.59** = 0.71. Better than split choice #1 (had ΔWS = 0.63).



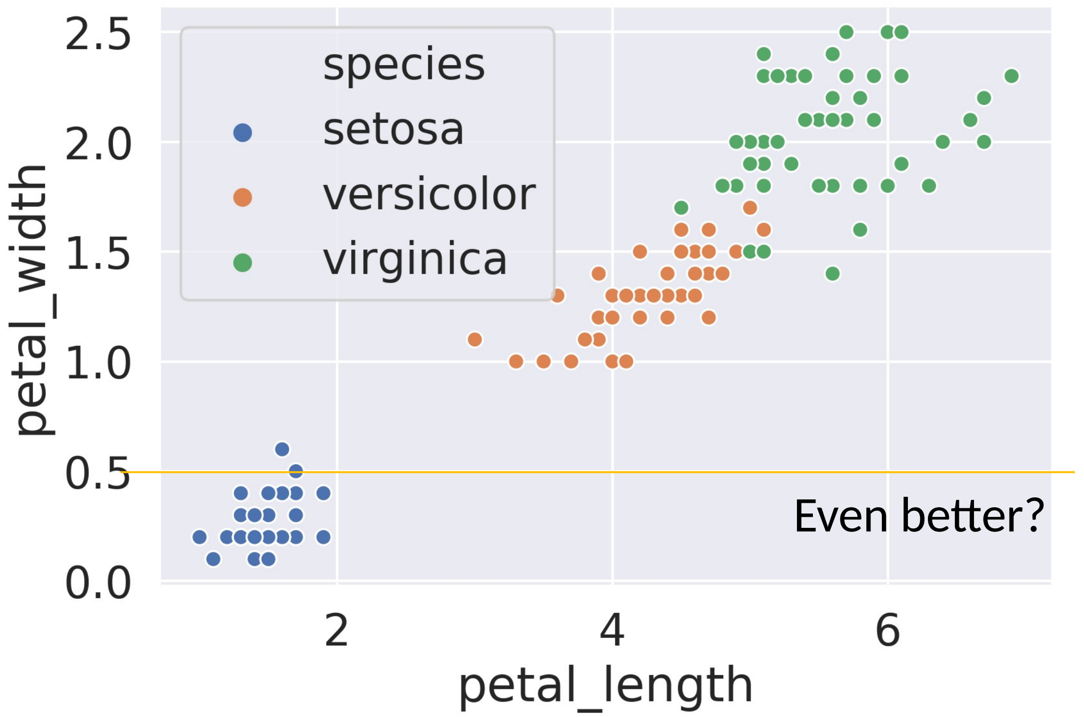


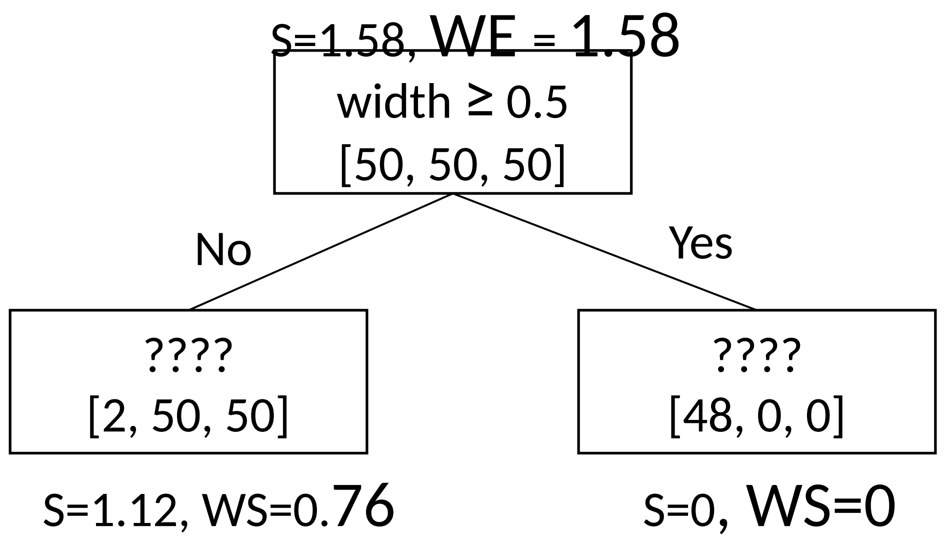
ΔWS = 0.71

Defining a Best Feature

Split choice #3: width ≥ 0.5. Compute entropy of child nodes:

- entropy([2, 50, 50]) = 1.12, weighted entropy = 0.68 * 1.12 = 0.76
- entropy([48]) = 0, weighted entropy = 0
- ΔWS = **1.58 - 0.76** = 0.82. Better than split choice #2 (had ΔWS = 0.71).



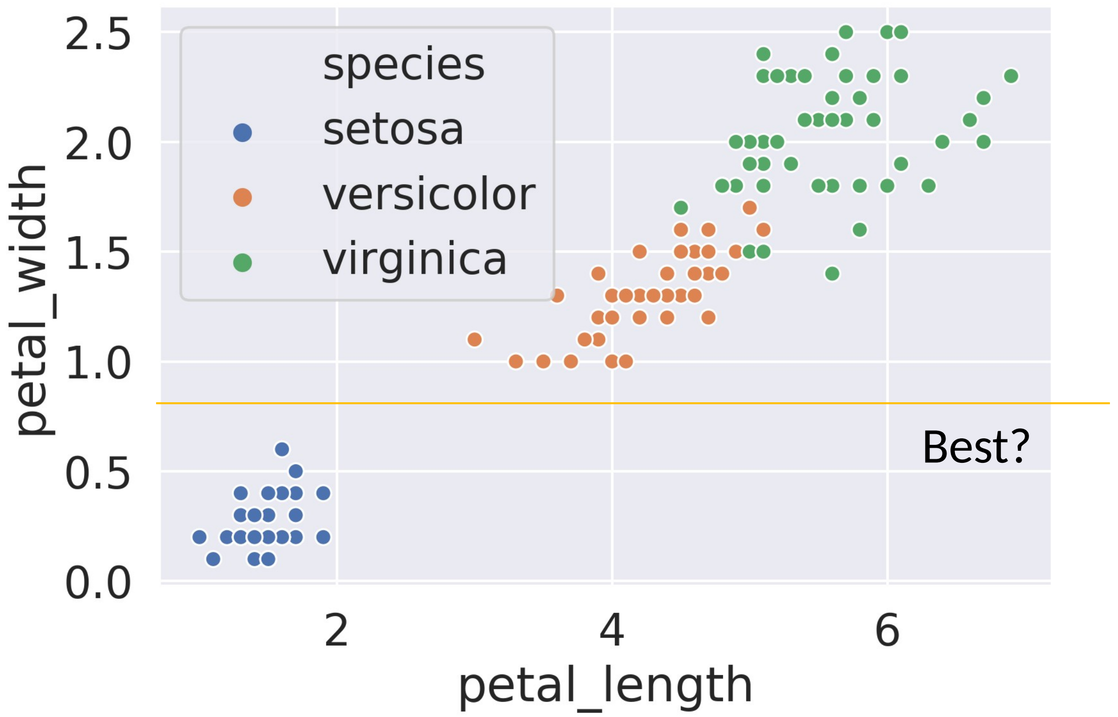


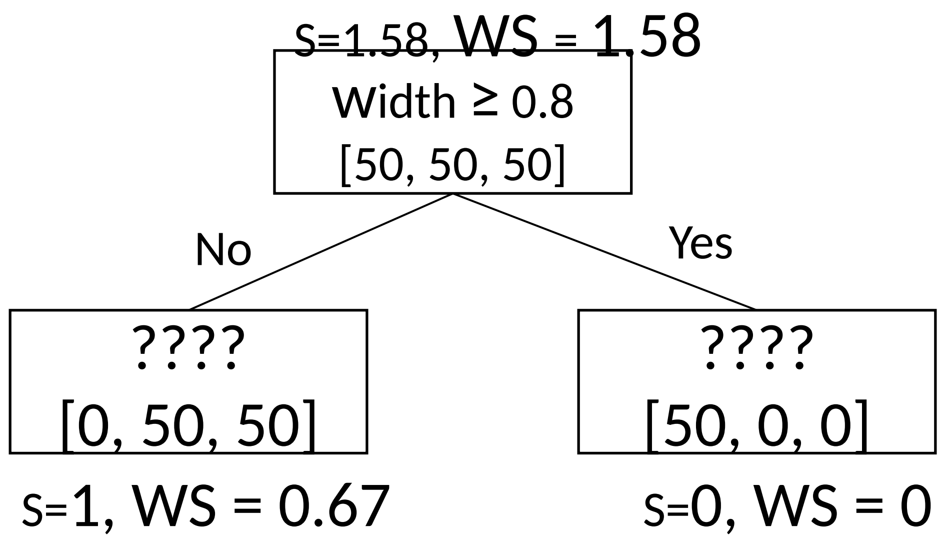
ΔWS = 0.82

Defining a Best Feature

Split choice #4: width ≥ 0.8. Compute entropy of child nodes:

- entropy([50, 50]) = 1, weighted entropy = 100/150 * 1 = 0.67
- entropy([50]) = 0, weighted entropy = 50/150 * 0 = 0
- ΔWS = **1.58 - 0.67** = 0.91. Better than split choice #3 (had ΔWS = 0.82).





ΔWS = 0.91

Decision Tree Generation Algorithm

Traditional decision tree generation algorithm:

- All of the data starts in the root node.
- Repeat until every node is either **pure** or **unsplittable**:
  - **Pick the best feature** x and **split value** β such that the **ΔWS** is **maximized**, e.g. x = petal_width, β = 0.8 has ΔWS **=** 0.91.
  - **Split data into two nodes**, one where x < β, and one where x ≥ β.

Notes: A node that has only one samples from one class is called a “**pure**” node. A node that has overlapping data points from different classes and thus that cannot be split is called “**unsplittable**”.

Let’s now turn our attention to avoiding overfitting.

Overfitting and Our Algorithm

A “fully grown” decision tree built with our algorithm runs the risk of overfitting.

One idea to avoid overfitting: Don’t allow fully grown trees.

- Approach 1: Set rules to prevent full growth.
  - Don’t split nodes with < 1% of the samples
  - Don’t allow nodes to be more than 7 levels deep in the tree
- Approach 2: Allow full growth then prune branches afterwards.
  - Branch have many rules that only affect few points

Won’t discuss these in any great detail.

- There’s a completely different idea called a “random forest” that is more popular and more beautiful. Won’t discuss in our course.