Decision Trees

Tree-based methods for classification and regression.

Sean Kang

What is it, and why use it?

- ML Algorithm used for classification and regression
- It is a non-parametric supervised learning algorithm – it means that it is not built or designed for any pre-determined set of data.

- Why use it?
- It is easy to understand and implement so it is a great tool to use for beginners in data science.

- Some aspect of the algorithm
- Working principles
- Different types of decision trees
- Building a decision tree
- Evaluating and optimizing decision tree

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.
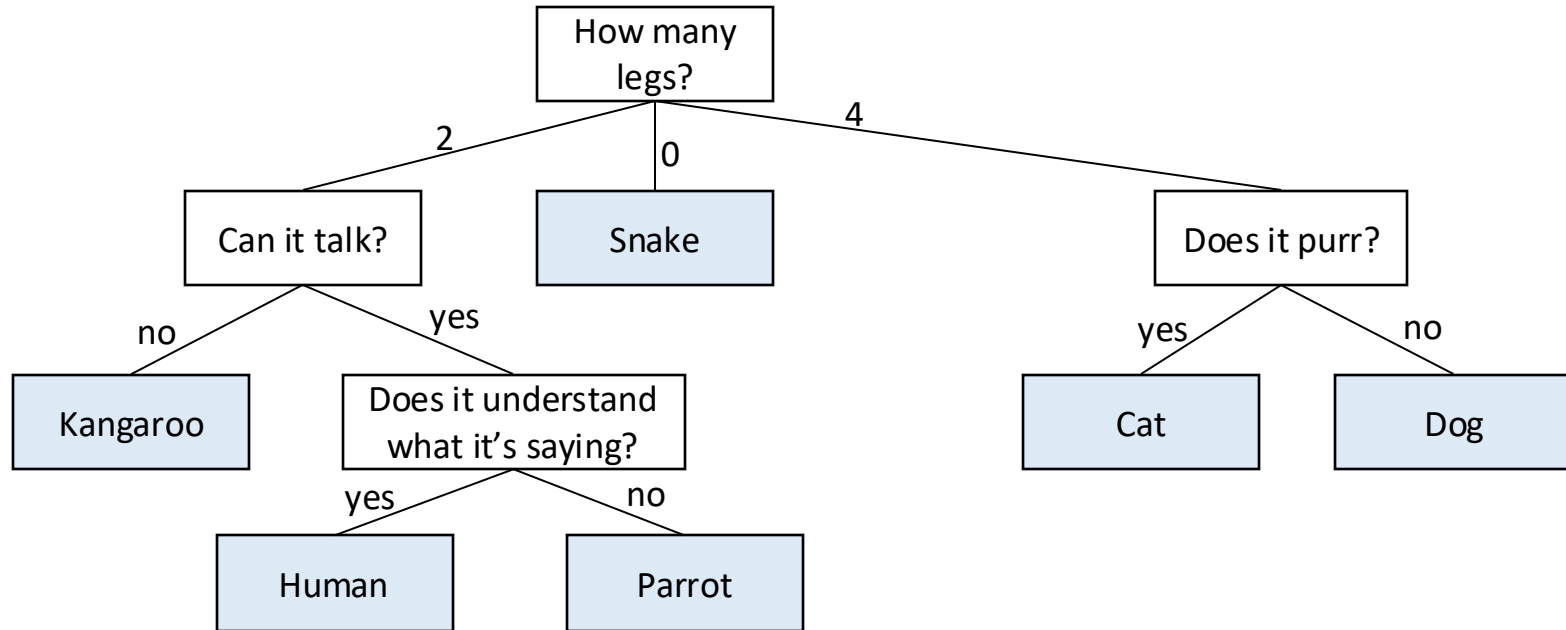
Decision Tree Terminologies

- Root node – all data population starts from here

- Decision nodes – represents intermediate decision or condition within tree

- Leaf nodes – final decision or terminal nodes

- Subtree – a subsection of tree or specific portion of tree (aka branch)

- Pruning – removing portions of the tree – reducing overfitting

- Parent and Child nodes – parent represents the prior condition or decision, whereas child represents the further processing, outcome or decision

Example: Flower Classification

The [Iris flower data set](https://en.wikipedia.org/wiki/Sepal) 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.

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

sepal

petal
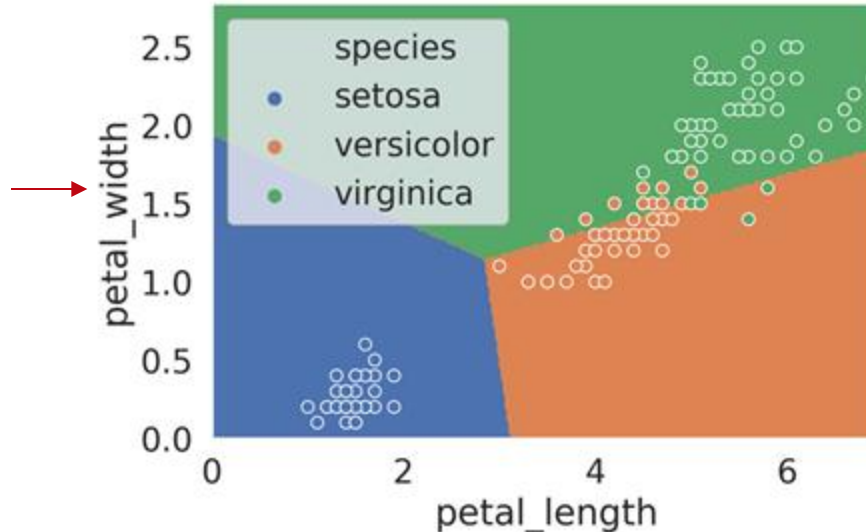
Classifying Irises Using Linear Models

Example: Logistic Regression 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.
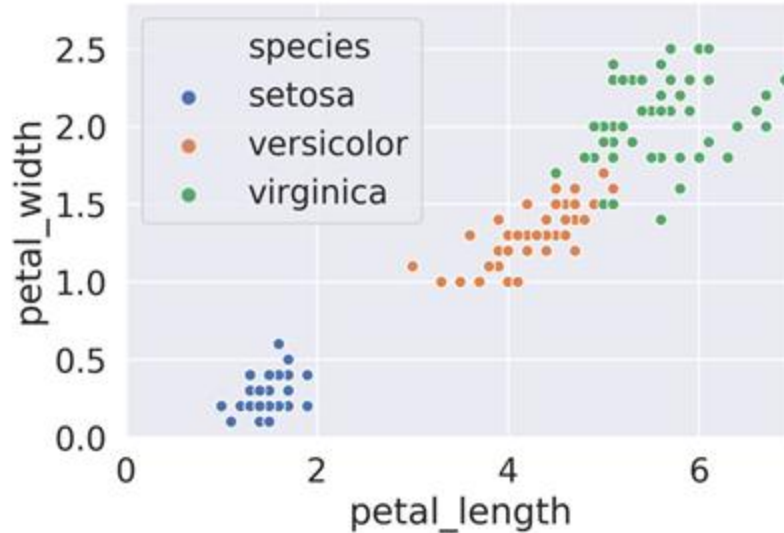
Example: Logistic Regression Using Petal Data Only

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

- If we train a 3 class logistic regression model on this data, we end up with the model below.

Because logistic regression models are linear, the decision boundaries are linear.
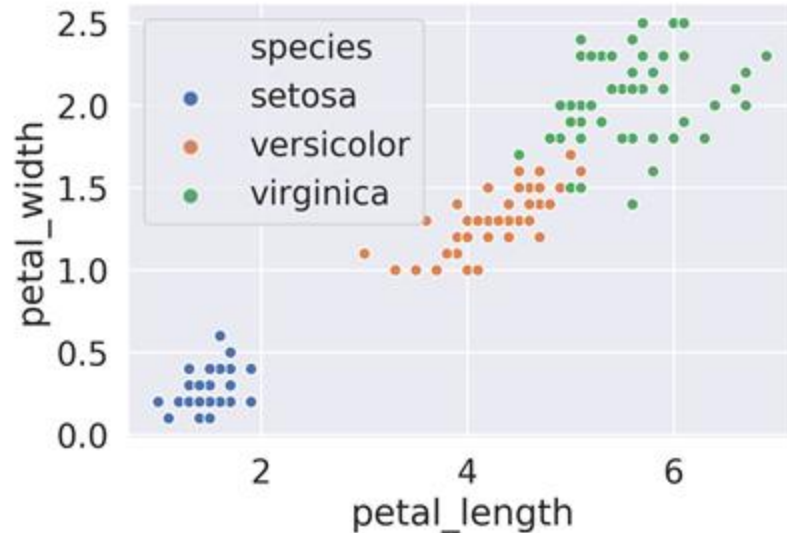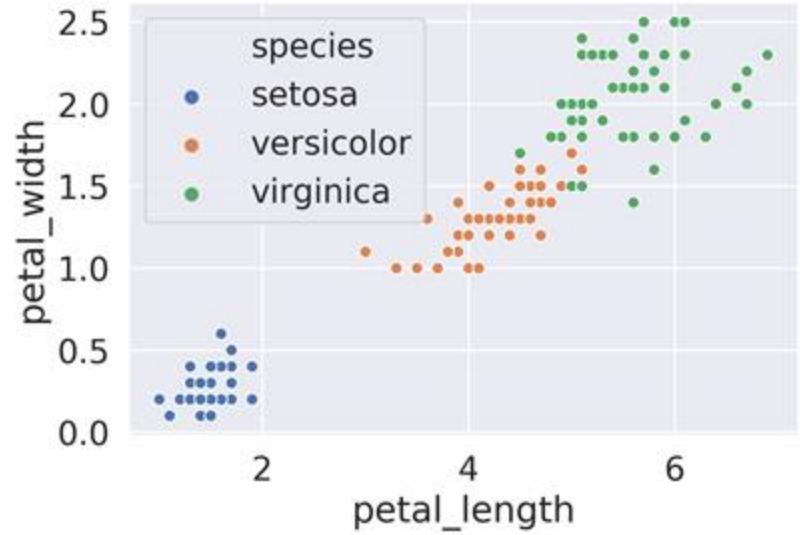
Decision Tree Basics

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.
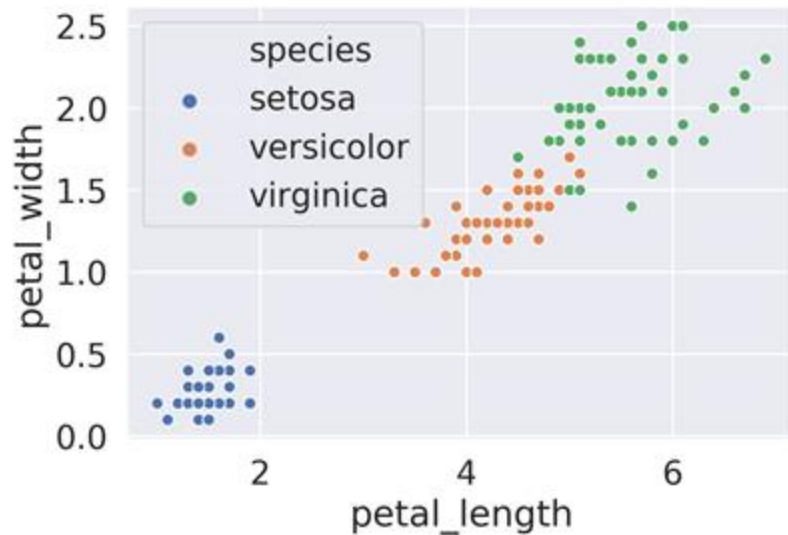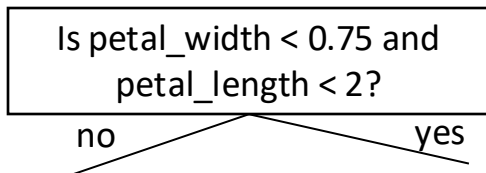
We can build a decision tree manually just by looking at this picture.
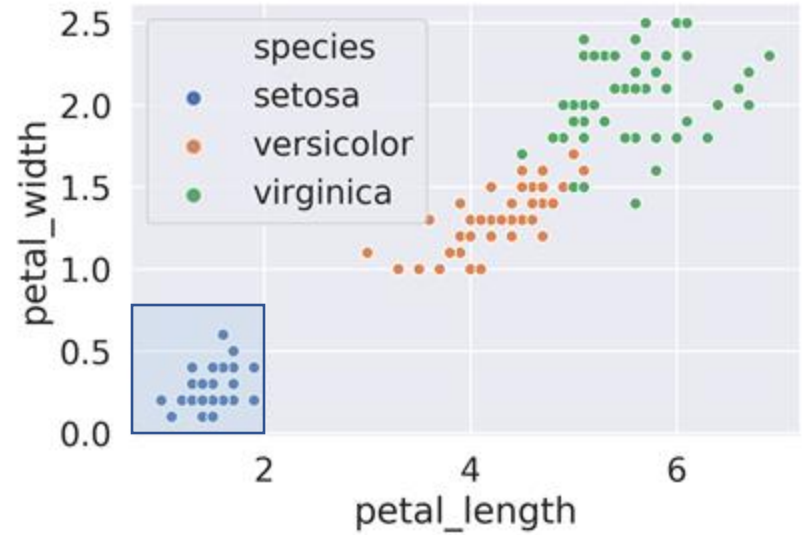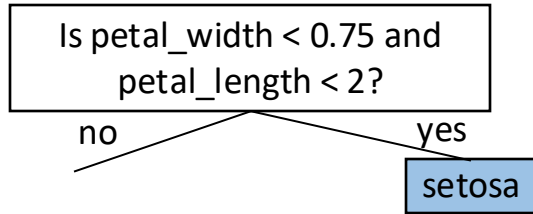
Example: Using Petal Data Only

Example: Using Petal Data Only

Is petal_width < 0.75 and petal_length < 2?

no   yes

Example: Using Petal Data Only

Is petal_width < 0.75 and petal_length < 2?

no — yes

setosa

Example: Using Petal Data Only

Is petal_width < 0.75 and petal_length < 2?
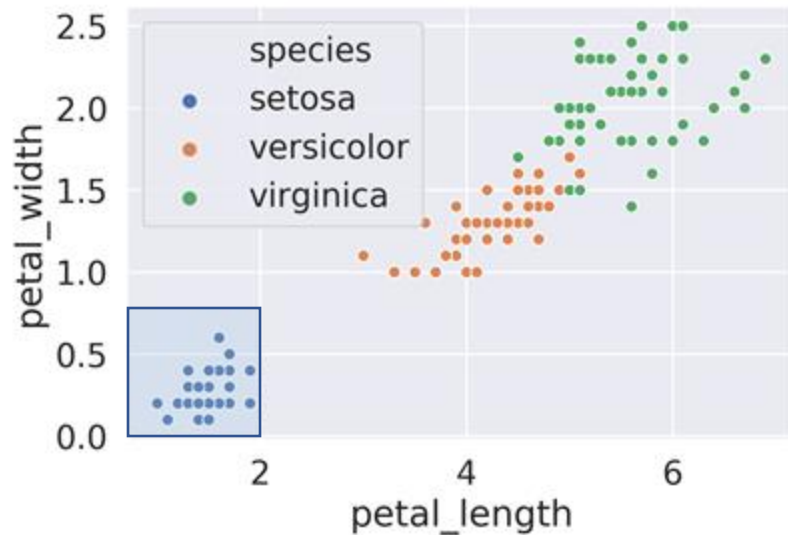
no   yes

Is petal_width >= 1.75

setosa

Example: Using Petal Data Only

Example: Using Petal Data Only

Example: Using Petal Data Only

Decision tree:
- Is petal_width < 0.75 and petal_length < 2?
  - yes → setosa
  - no → Is petal_width >= 1.75
    - yes → virginica
    - no → Is pedal_width <= 1.55 and pedal_length <= 4.95
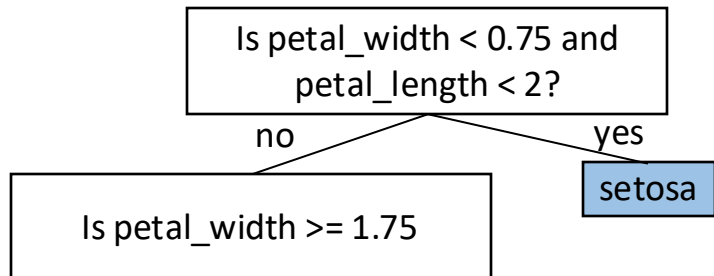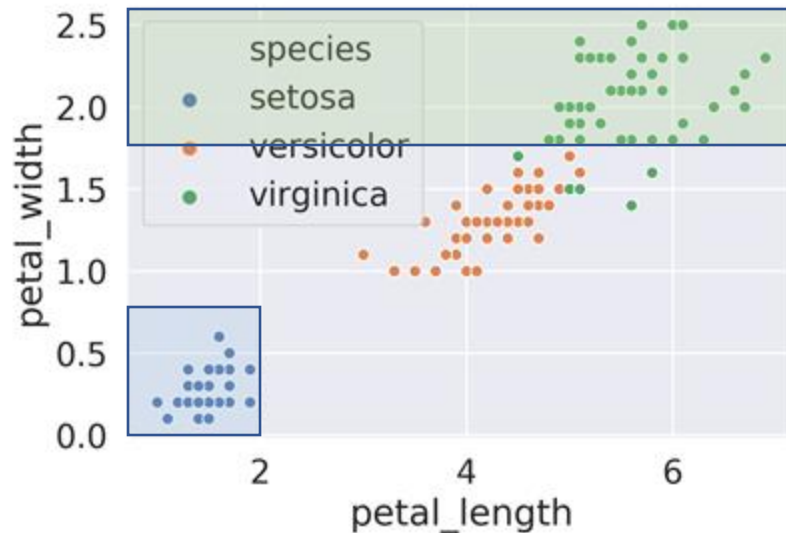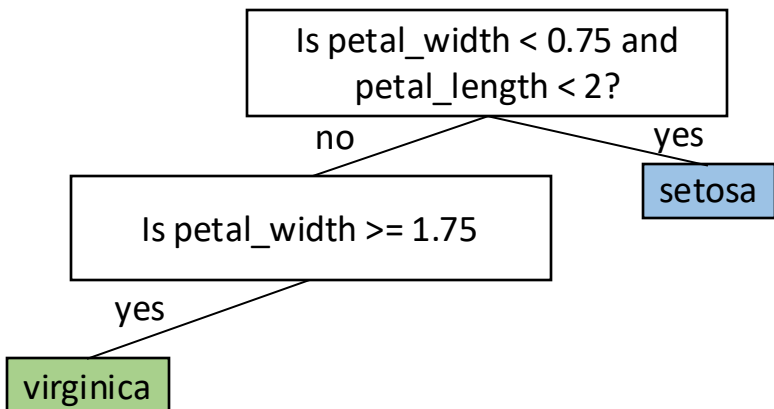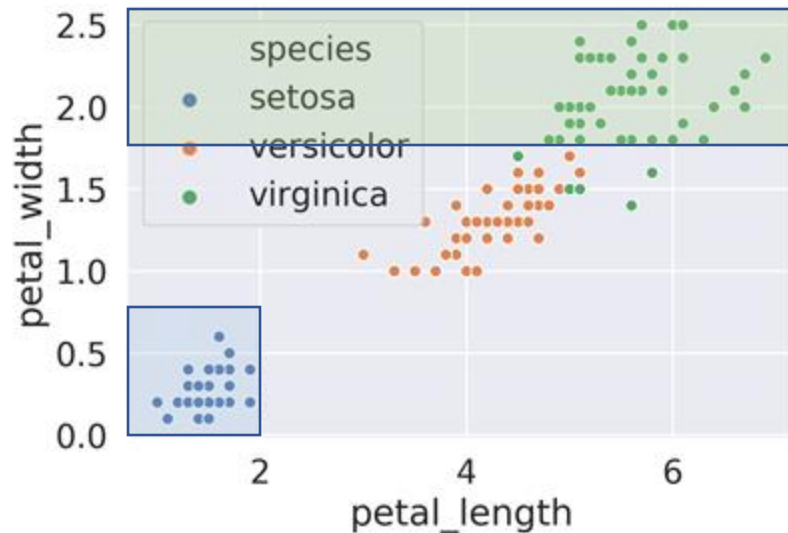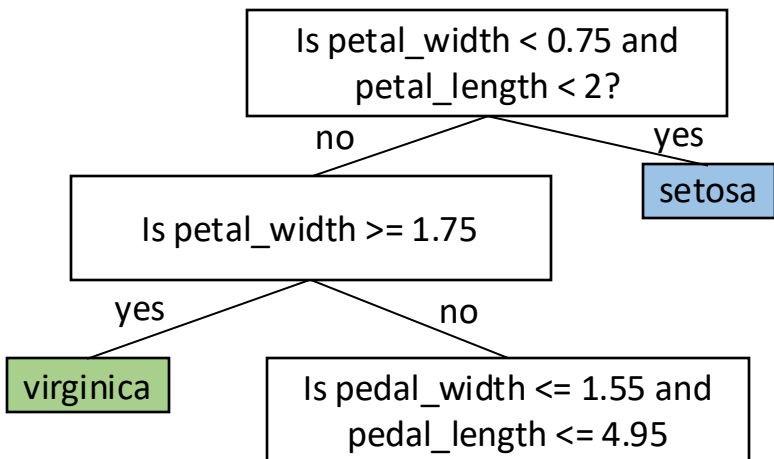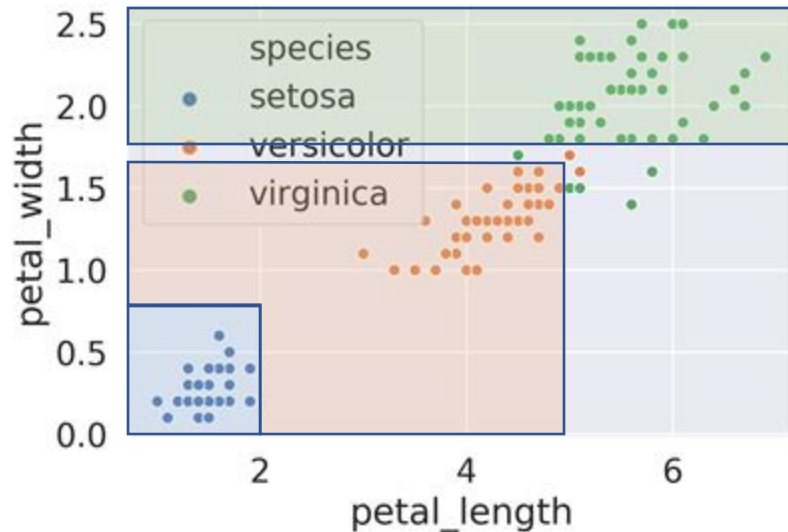   - yes → versicolor
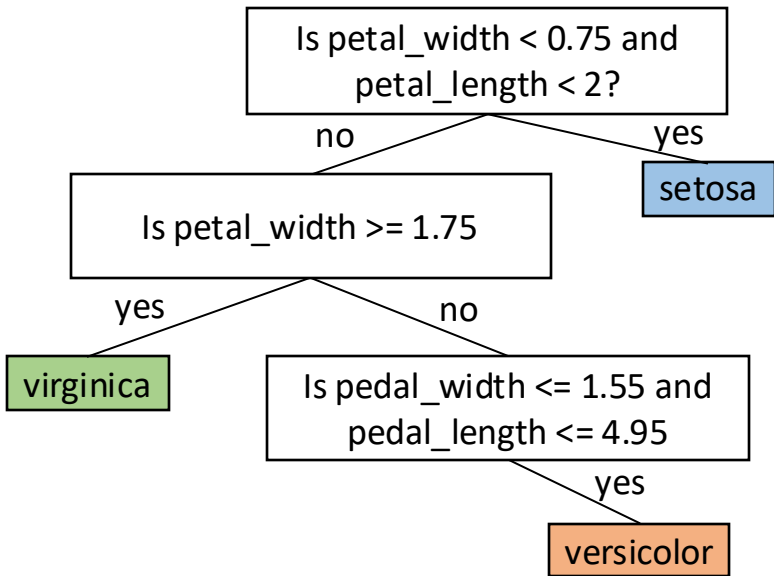
Example: Using Petal Data Only
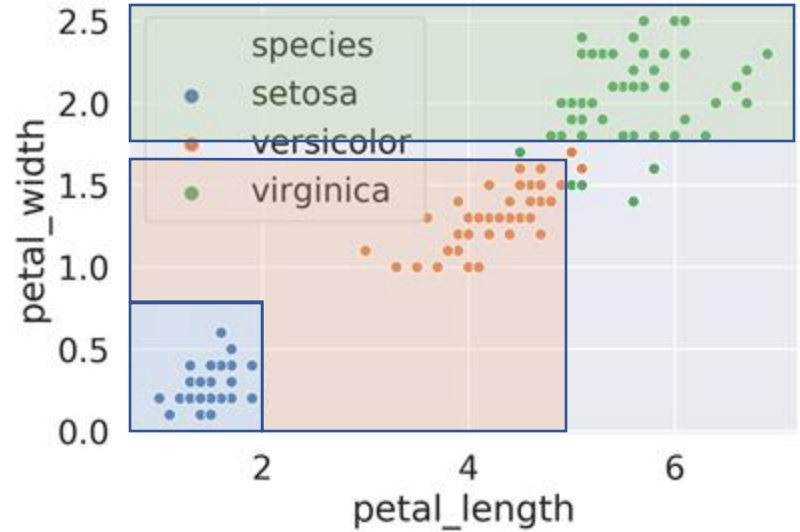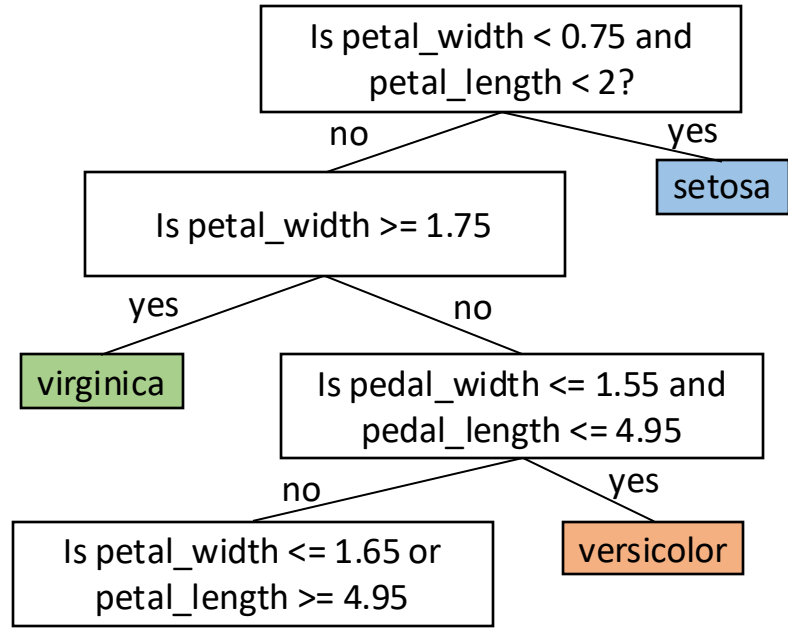
Example: Using Petal Data Only
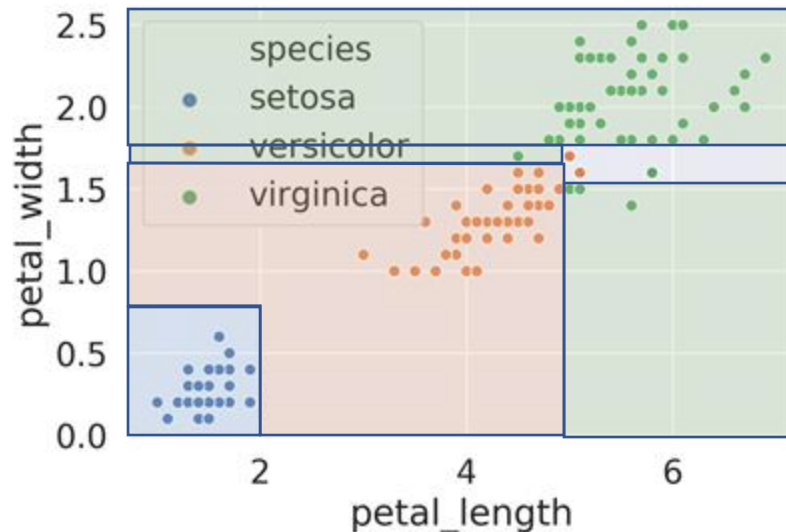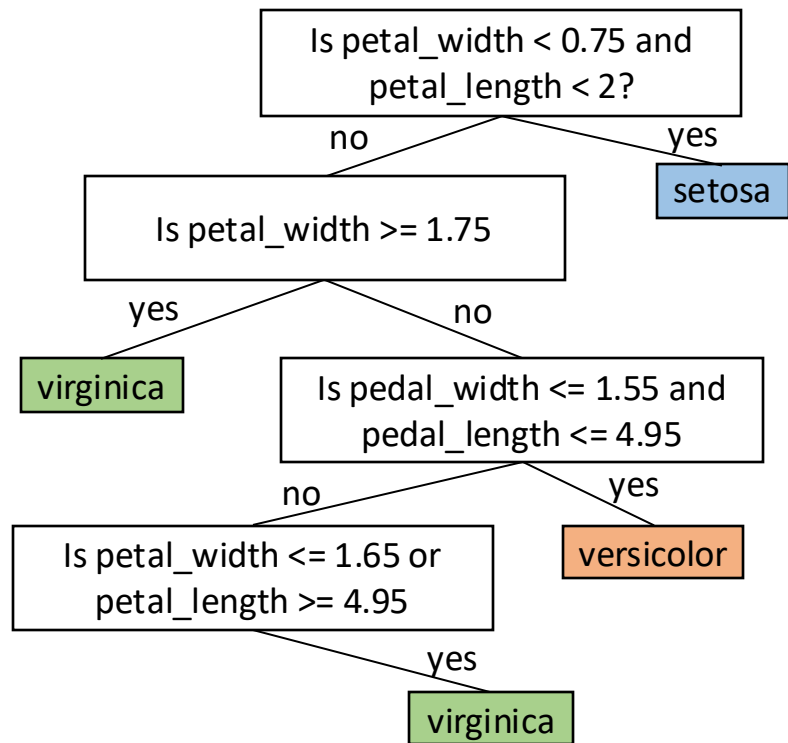
Example: Using Petal Data Only
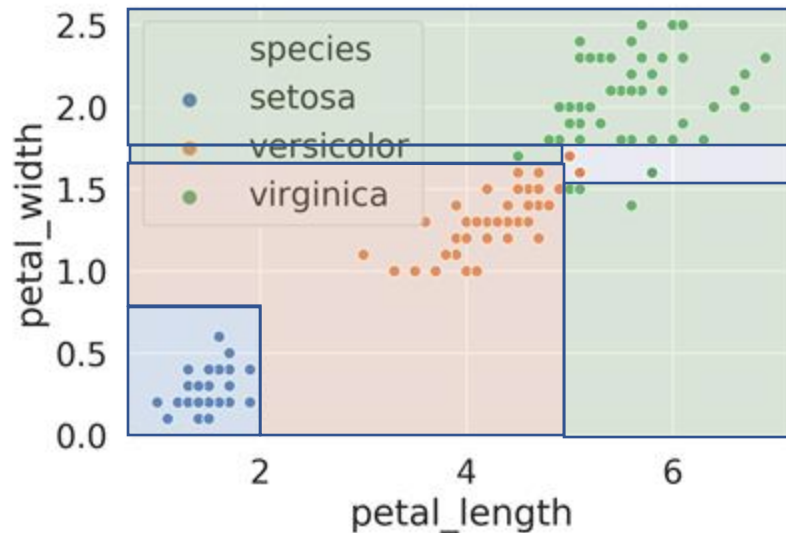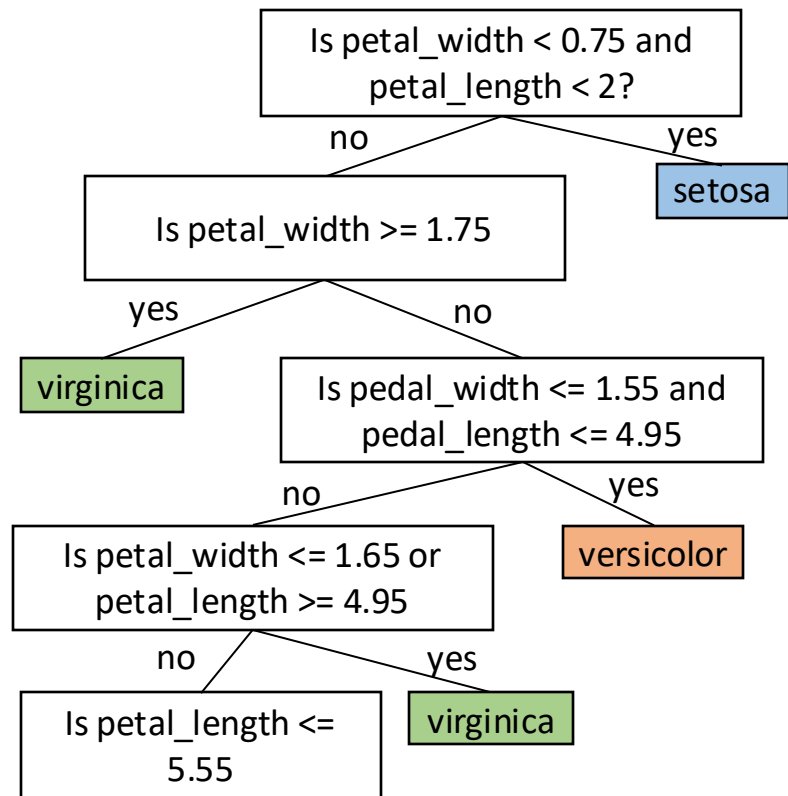
Example: Using Petal Data Only
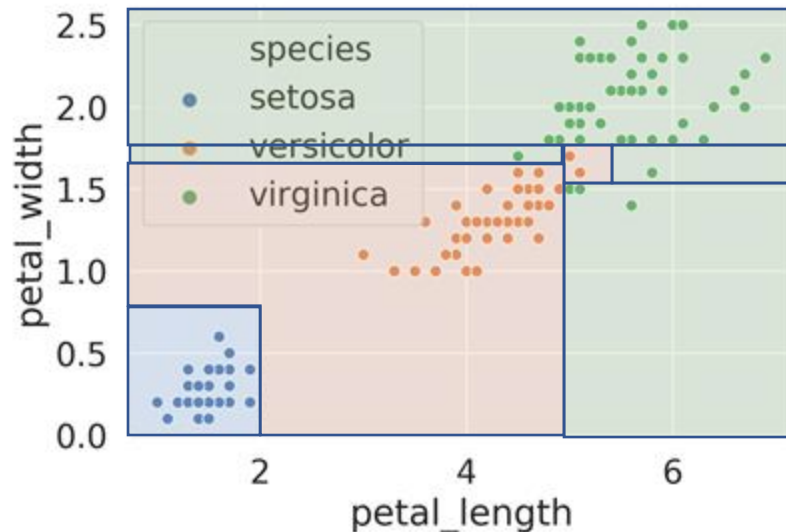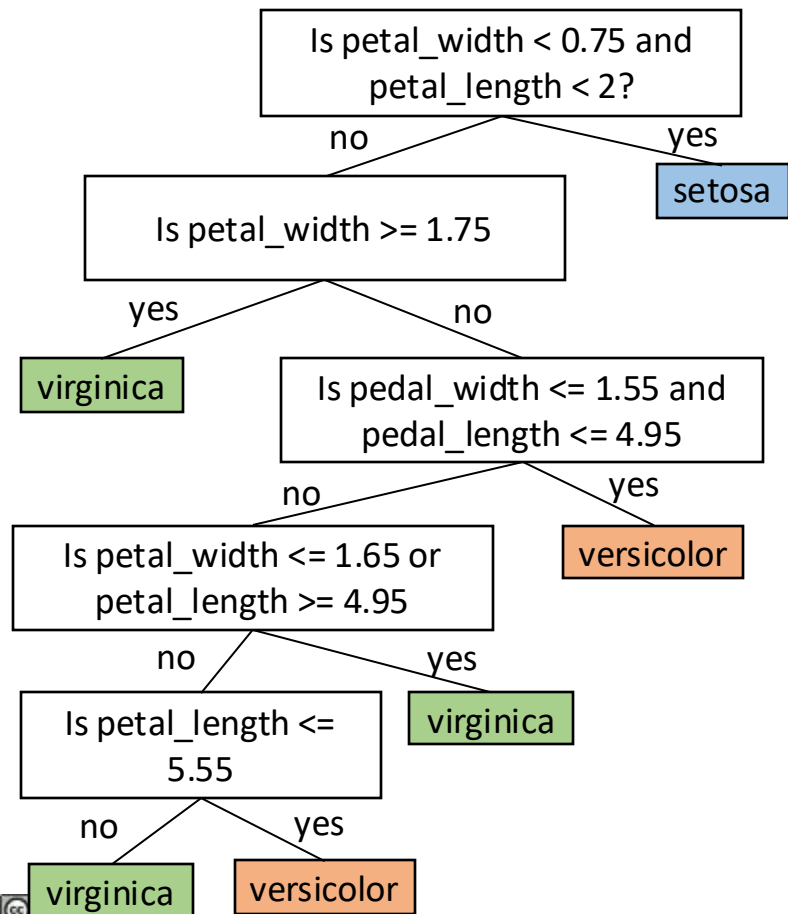
Example: Using Petal Data Only

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

Is this good or bad?
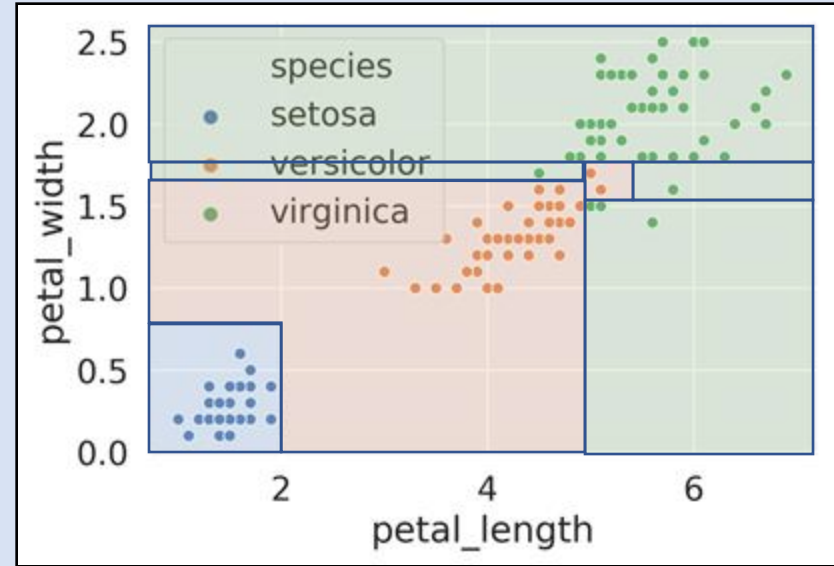
Example: Using Petal Data Only

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`
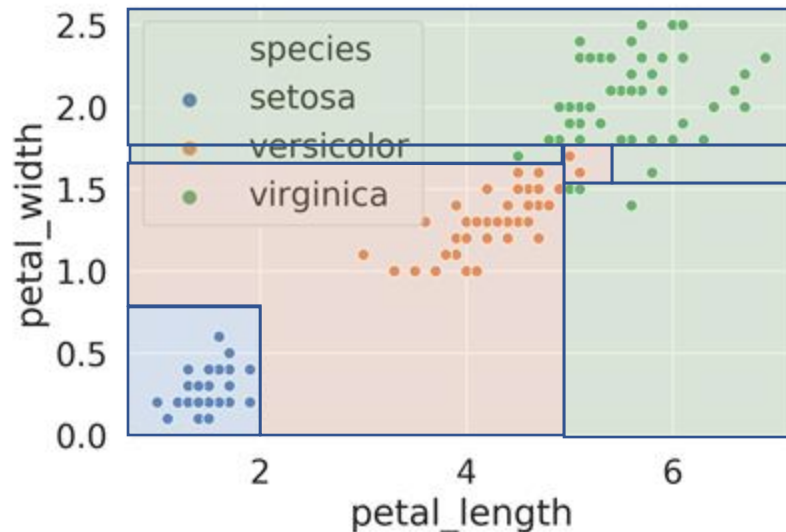
Decision Trees in scikit-learn

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()
decision_tree_model = decision_tree_model.fit(iris_data[["petal_length", "petal_width"]], iris_data["species"])
```

See `lec20-decision-trees.ipynb` if you want to try it out

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()
decision_tree_model = decision_tree_model.fit(iris_data[["petal_length", "petal_width"]], iris_data["species"])
```

See `lec20-decision-trees.ipynb` if you want to try it out.

```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"]])
```

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

Visualizing Decision Tree Models

```
from sklearn import tree
decision_tree_model = tree.DecisionTreeClassifier()
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:

Visualizing Decision Tree Models

```python
from sklearn import tree
decision_tree_model = tree.DecisionTreeClassifier()
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
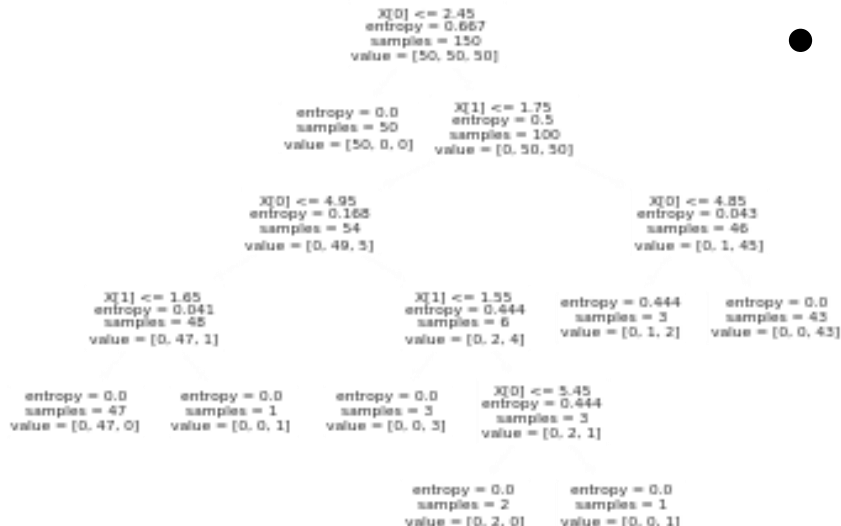
```python
tree.plot_tree(decision_tree_model)
```

Visualizing Decision Tree Models

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,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

In each box, we see:
- The rule
- The gini impurity (chance that a sample would be misclassified if randomly assigned at this point)
- The number of samples still unclassified
- The number of samples in each class still unclassified
- The most likely class

Visualizing Decision Tree Models

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,
                        special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

In each box, we see:
- The rule
- The gini impurity (chance that a sample would be misclassified if randomly assigned at this point)
- The number of samples still unclassified
- The number of samples in each class still unclassified
- The most likely class

petal_length ≤ 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginia

gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginia

gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginia

petal_length ≤ 5.45
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor

= 0.0
les = 2
= [0, 2, 0]

gini = 0.0
samples = 1
value = [0, 0, 1]

Visualizing Decision Tree Models

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,
                    special_characters=True)
graph = graphviz.Source(dot_data)
graph
```
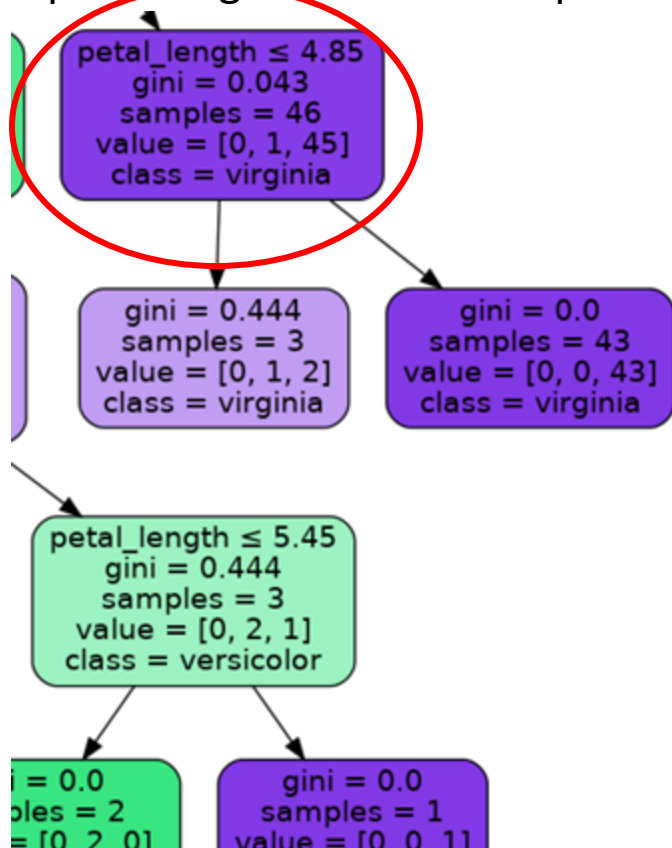

In each box, we see:
- The rule
- The gini impurity (chance that a sample would be misclassified if randomly assigned at this point)
- The number of samples still unclassified
- The number of samples in each class still unclassified
- The most likely class

Visualizing Decision Tree Models

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,
                        special_characters=True)
graph = graphviz.Source(dot_data)
graph
```
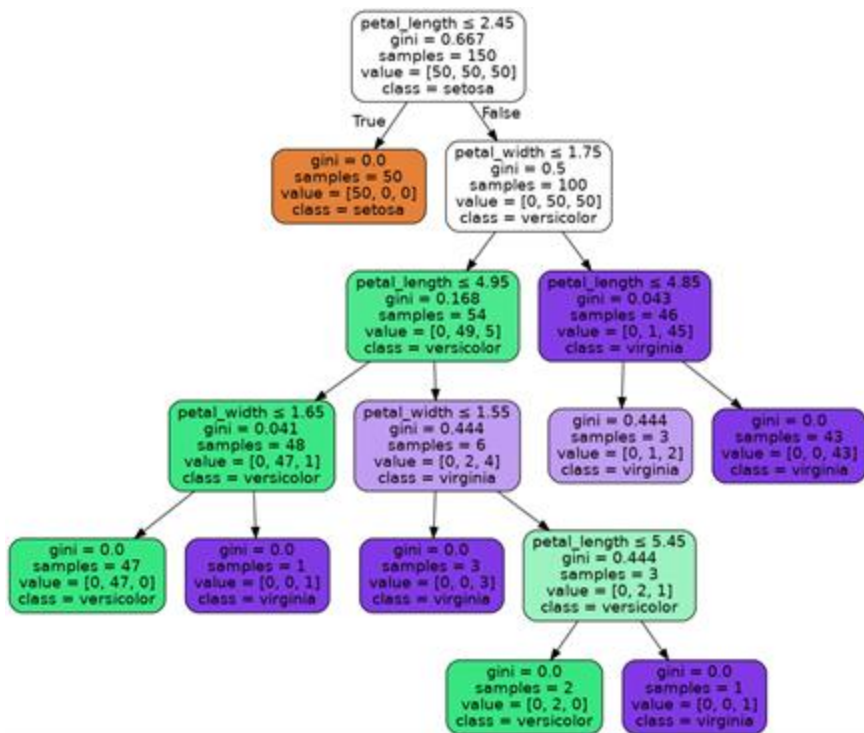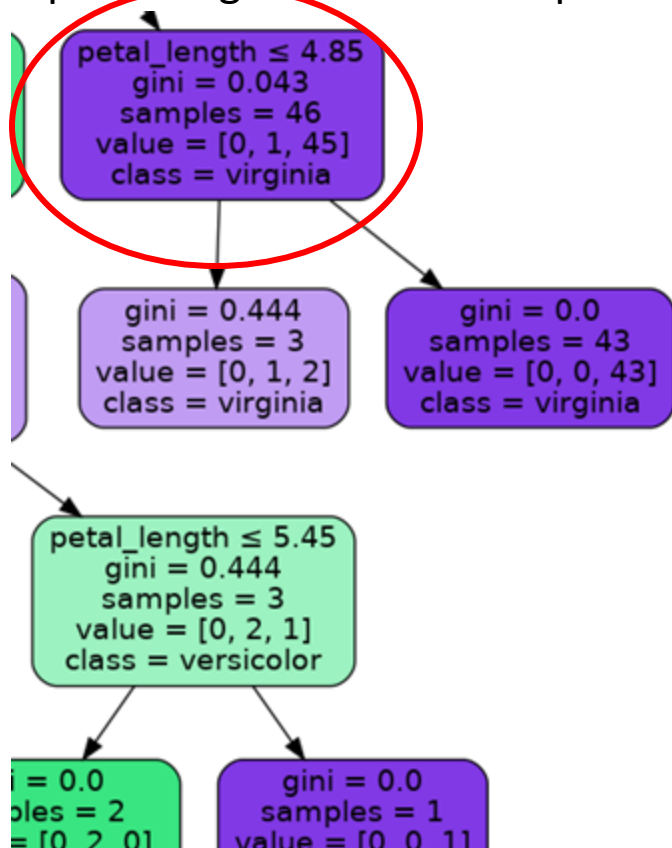
In each box, we see:
- The rule
- The gini impurity (chance that a sample would be misclassified if randomly assigned at this point)
- The number of samples still unclassified
- The number of samples in each class still unclassified
- The most likely class

petal_length ≤ 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
class = virginia

gini = 0.444
samples = 3
value = [0, 1, 2]
class = virginia

gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginia

petal_length ≤ 5.45
gini = 0.444
samples = 3
value = [0, 2, 1]
class = versicolor

i = 0.0
oles = 2
= [0, 2, 0]

gini = 0.0
samples = 1
value = [0, 0, 1]

Visualizing Decision Tree Models

Plotting the decision boundaries for our logistic regression & decision tree models, yields these results:

- Decision tree has nonlinear boundary, and appears to get 100% accuracy
  - Let's calculate the exact accuracy rather than just relying on our eyes
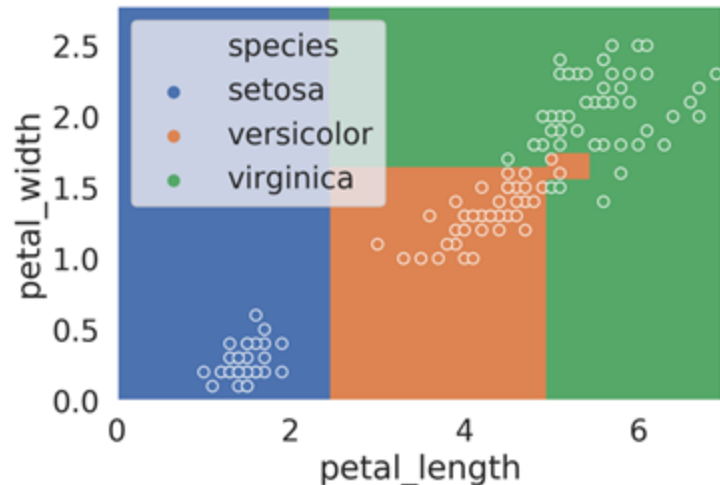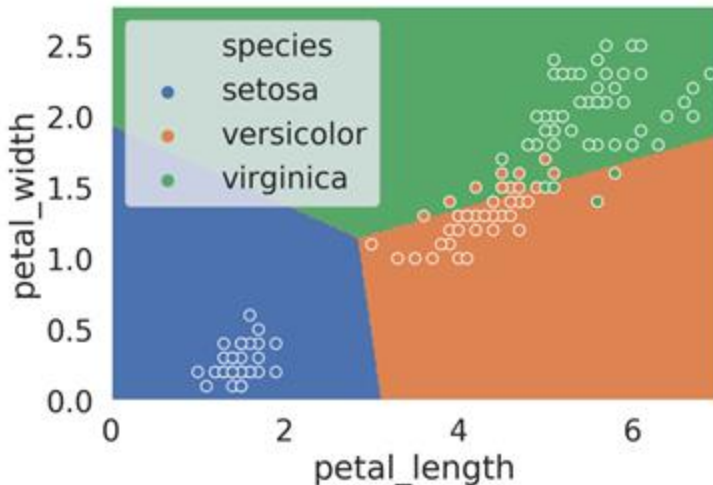
Measuring the Performance of Our Model

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

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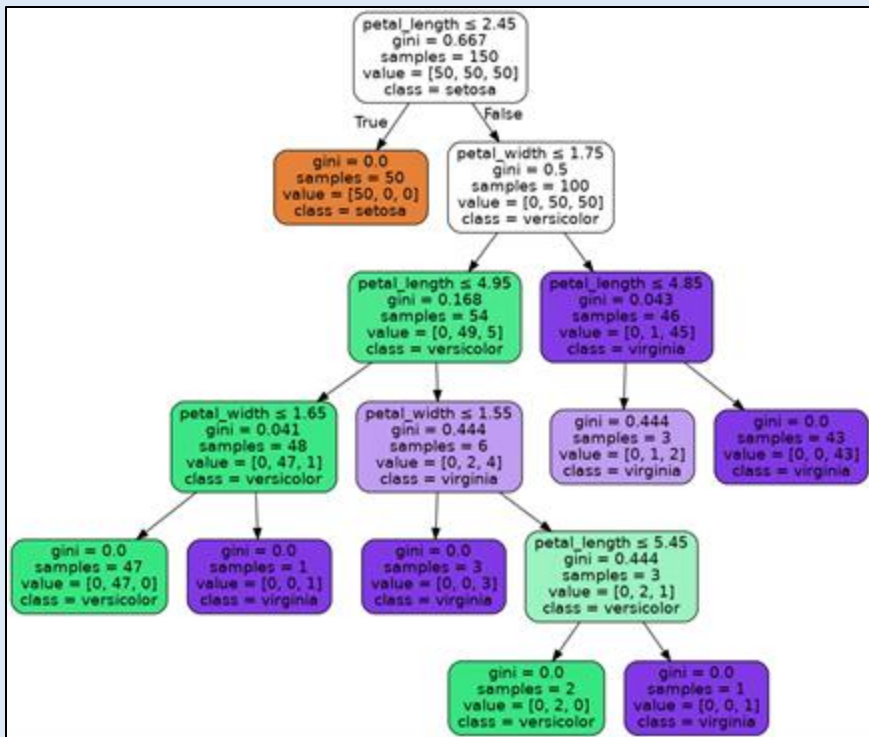
There is one terminal decision point where there is more than one possible right answer

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")
```
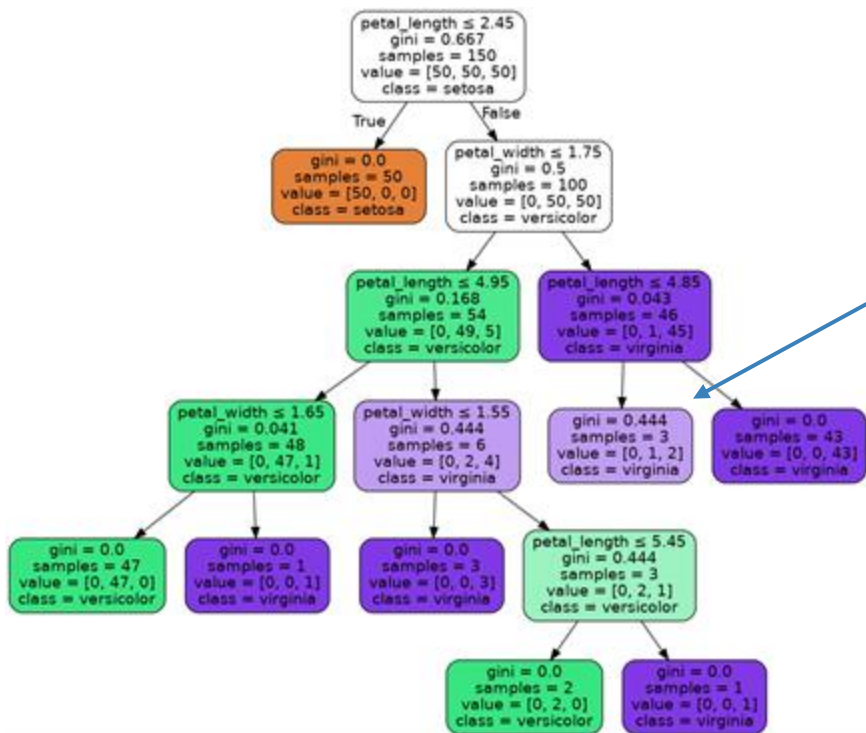
	sepal_length	sepal_width	petal_length	petal_width	species
70	5.9	3.2	4.8	1.8	versicolor
126	6.2	2.8	4.8	1.8	virginica
138	6.0	3.0	4.8	1.8	virginica
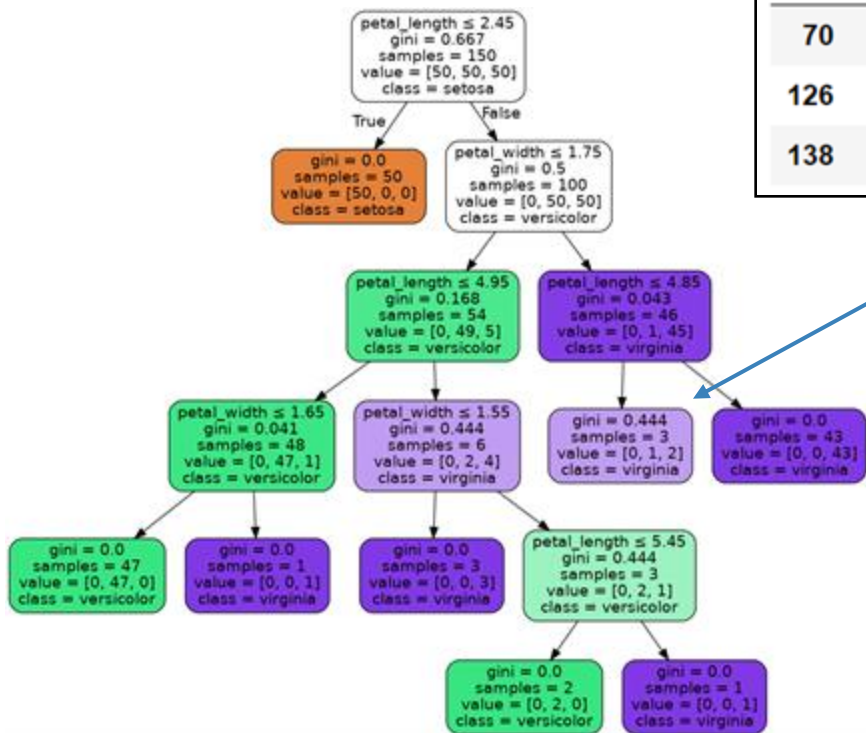
There is one terminal decision point where there is more than one possible right answer
- In the original data set, there was a versicolor iris with the same petal measurements as two virginicas

Overfitting and Decision Trees

Bottom line: `scikit-learn` makes it easy to generate 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

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

- That is, if the versicolor above had a petal_length of 4.800001, we'd have 100% training accuracy

This tendency for perfect accuracy should give
    us concern about overfitting

Decision Trees and Overfitting

Overfitting and Decision Trees

With the exception of samples that have the exact same data, we can always get 100% accuracy. This should give us concern about possible overfitting

Let's see how bad things are by separating our data into train and test sets

- We'll use 110 training points and 40 testing points
- Performance on test set gives us an estimate of how well we generalize

```
train_iris_data, test_iris_data = np.split(iris_data.sample(frac=1), [110])
```

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

Visualizing Our New Model

Comparing our model trained on 150 vs. 110 data points, we see slight differences in the generated model

- Naturally, both models get very high accuracy on the data that they use for training (100% for non-overlapping points)

Trained on all 150 points.
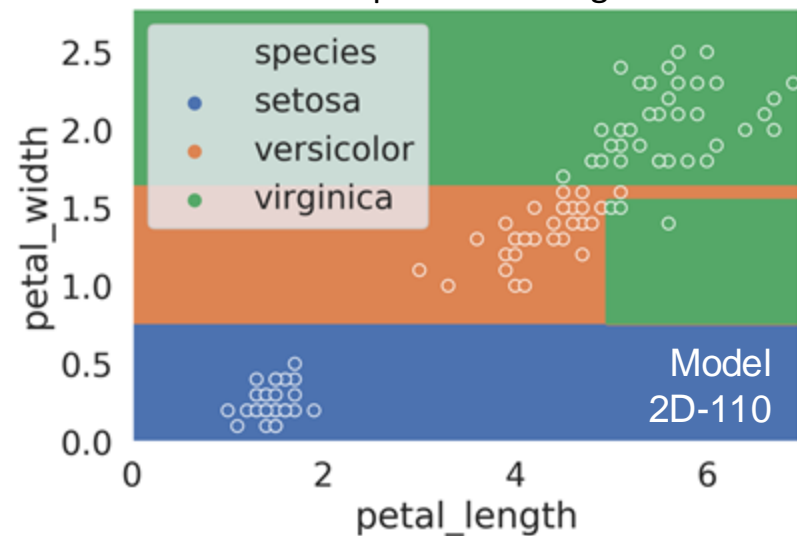Trained on 110 points.
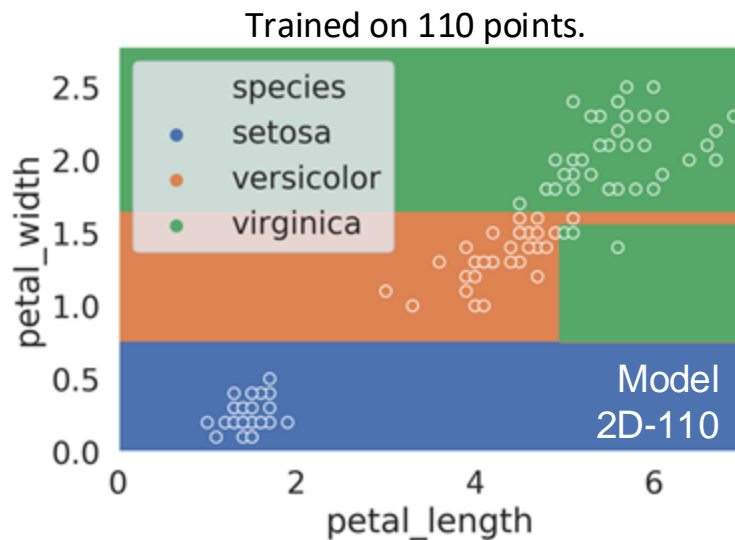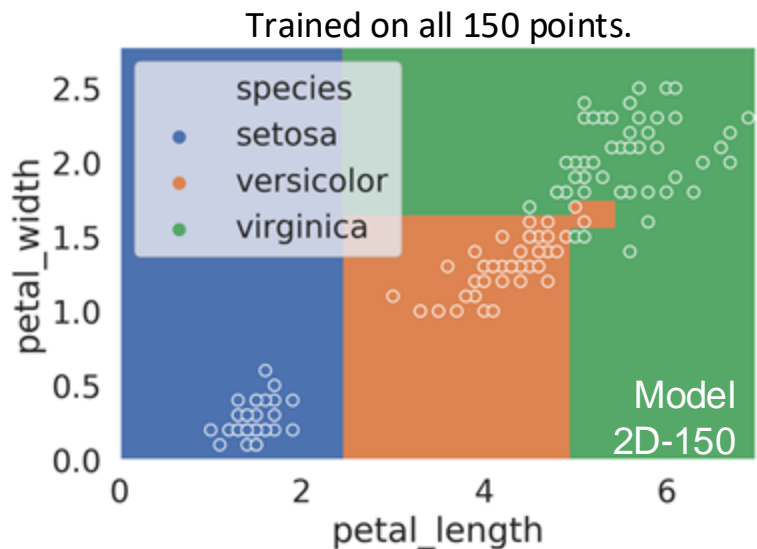
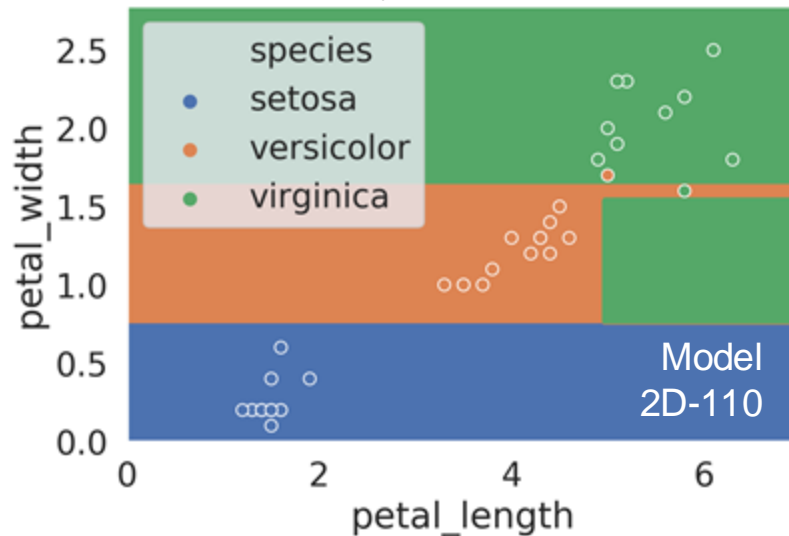# Performance of Our New Model on the Test Set

When we look at the performance of Model 2D-110 on the test set, we see that we make some errors that aren't just from overlapping data

- 99% accuracy on training set (2 overlapping points). 95% accuracy on test set

Trained on 110 points. Training set.   Trained on 110 points. Test set.

Multidimensional Decision Trees

Naturally, we can include even more features. For example, if we want to use the petal AND sepal measurements, we simply train the decision tree on all four columns of the data

```
decision_tree_model_4d = tree.DecisionTreeClassifier()
decision_tree_model_4d = decision_tree_model_4d.fit(train_iris_data[["petal_length", "petal_width",
                                                    "sepal_length", "sepal_width"]], train_iris_data["species"])
```

The resulting model Model 4D-110 gets:

- 100% accuracy on the training set (no overlapping data points)
- 95% accuracy on the test set

Cannot easily visualize Model 4D-110's decision
boundaries because the prediction space
is 4 dimensional

Model 2D-150 vs. Model 4D-110 Tree Visualization
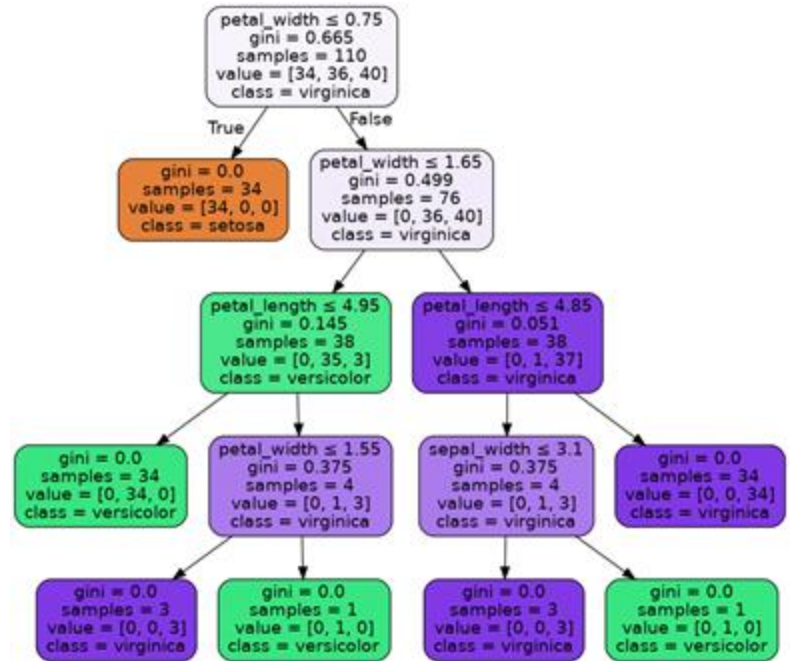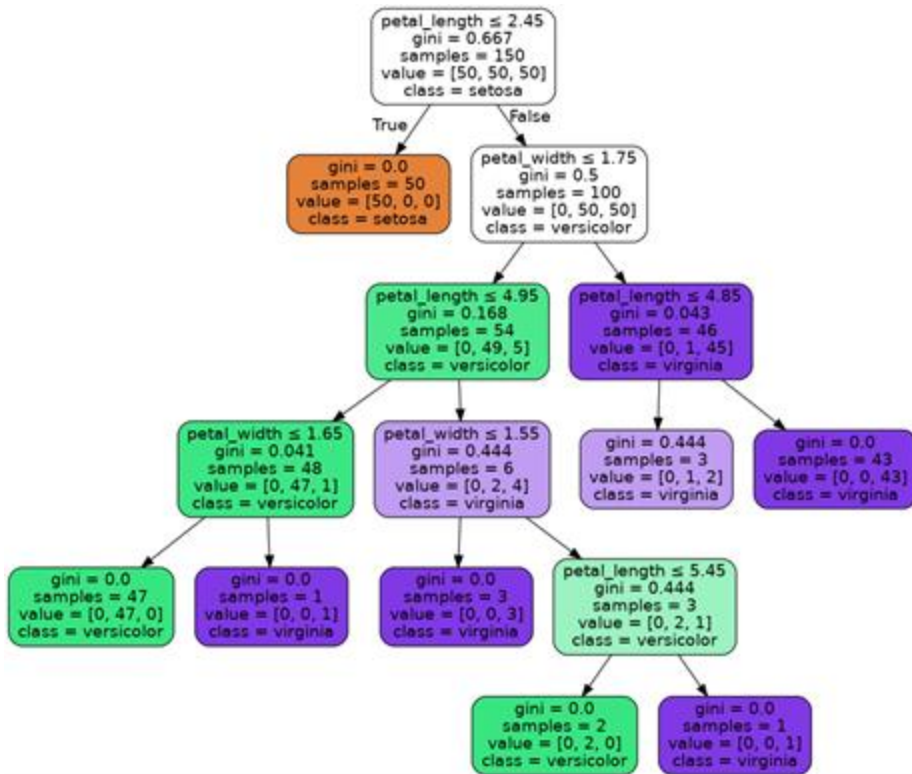
What is different about these two models?

Model 2D-150 vs. Model 4D-110 Tree Visualization

Models are quite similar:

- Model 4D-110 uses only 110 samples.
- The decision rules are nearly identical, except that Model4D-110 uses the sepal_width exactly once to resolve the case that we couldn't resolve with petal_length and petal_width

Model 4D-110 seems marginally better to me, but both got only 95% accuracy on test set.
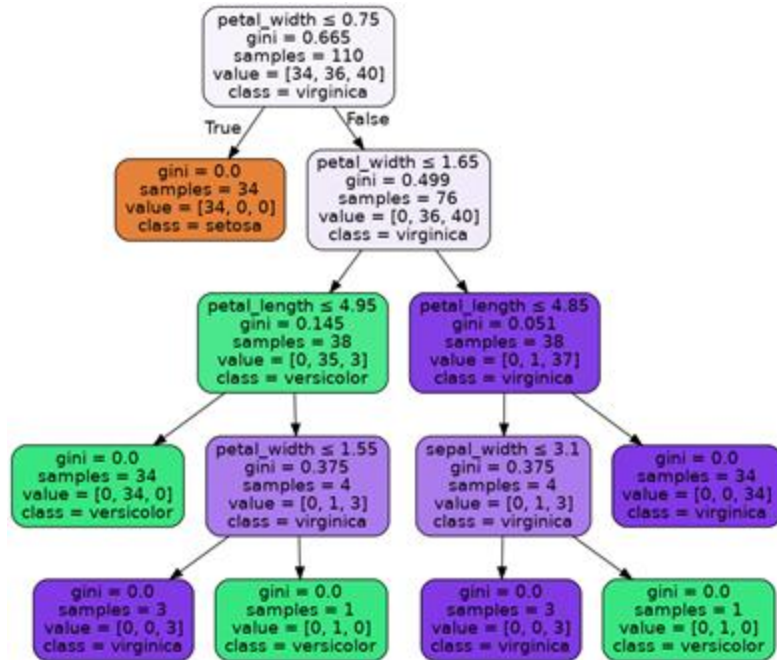
- Need more data to know for sure

Model 2D-150 vs. Model 4D-110 Tree Visualization

Models are quite similar:

- Model 4D-110 uses only 110 samples.
- The decision rules are nearly identical, except that Model4D-110 uses the sepal_width exactly once to resolve the case that we couldn't resolve with petal_length and petal_width

Model 4D-110 seems marginally better to me, but both got only 95% accuracy on test set.
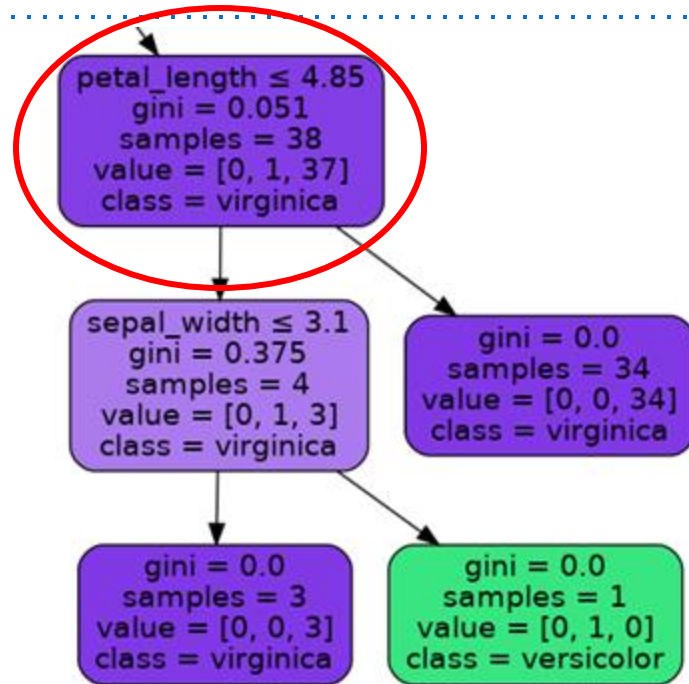
- Need more data to know for sure

Overfitting and Sepal Data

If we use the sepal data only (no petal data), we will run into serious overfitting issues

Below, we see our three flowers plotted in the sepal space
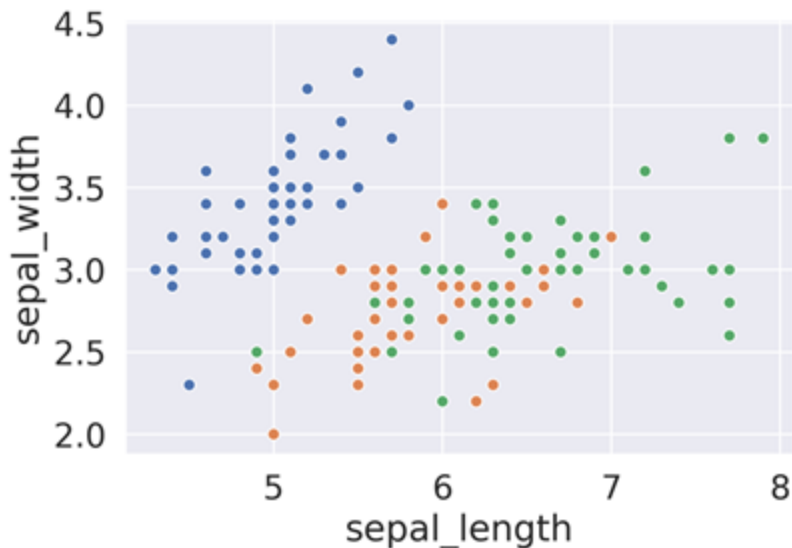
Sepal Decision Boundaries

After training on 110/150 points, we get the decision boundaries below

- Decision boundaries are erratic!
- Many overlapping points leads to only 93% accuracy on training set

Trained on 110 points. Training set.
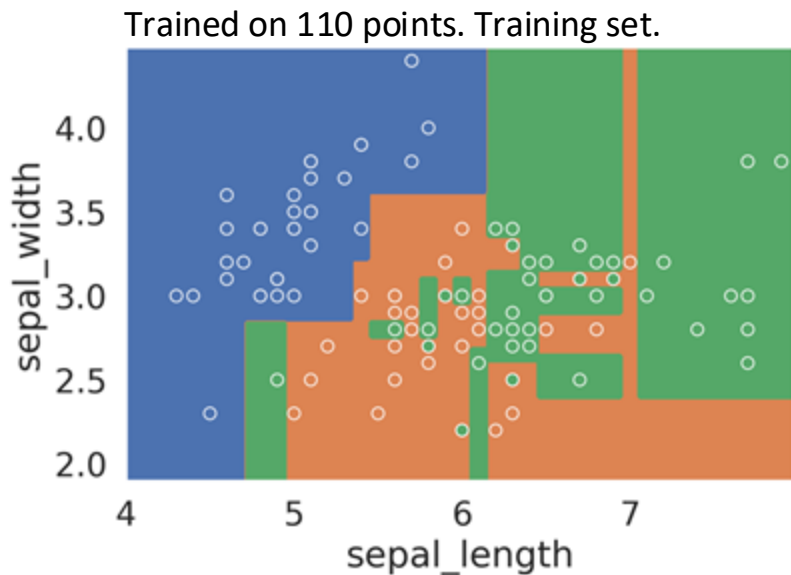
Sepal Decision Boundaries and Test Set

Performance on test set is quite poor

- Only 70% accuracy: 28/40 predictions are correct

Trained on 110 points. Test set.
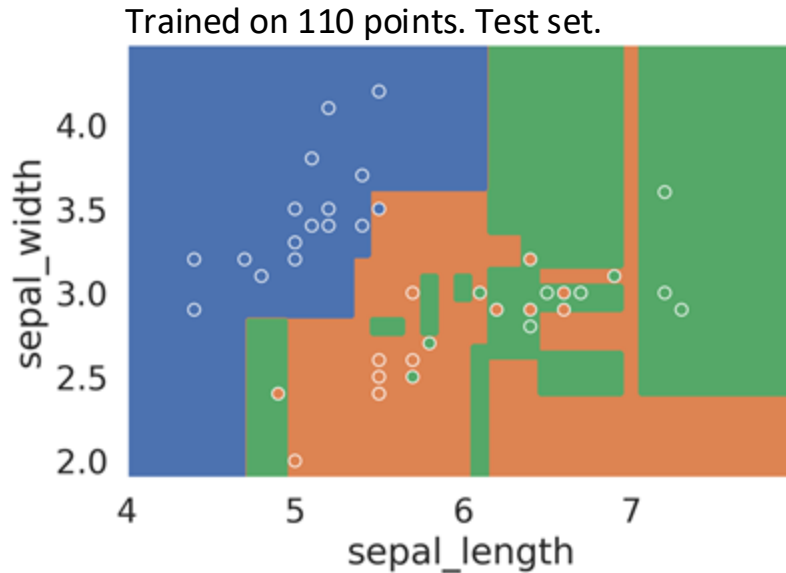
Overfitting Visualized

The decision boundaries for our sepal model were quite complex

- Or drawn out as a tree, we also see a highly complex structure

Next, we'll discuss strategies for preventing overfitting

Model 2D-Sepal-110
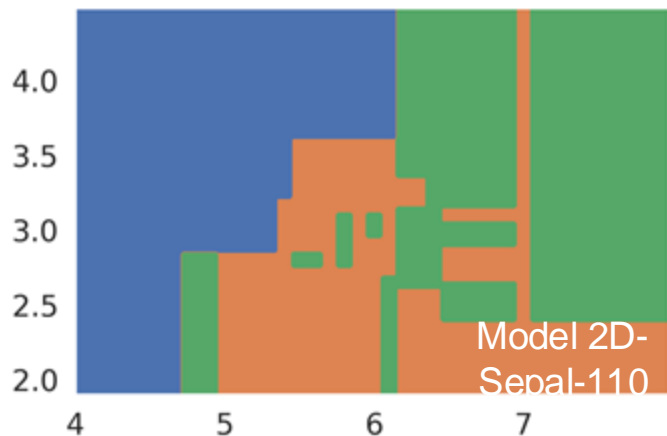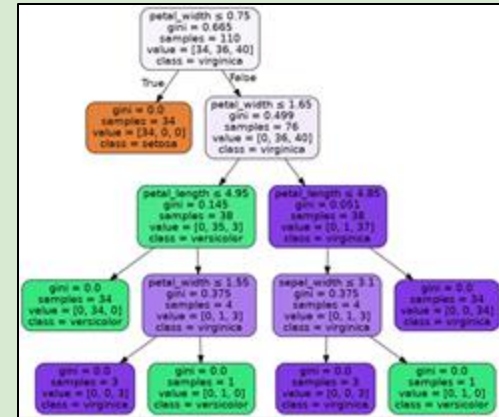
Basic Decision Tree Generation
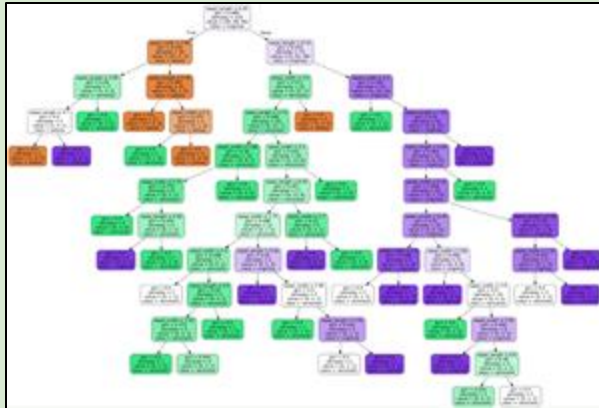
Decision Tree Generation

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 $\geq$ β

Notes: A node that has only one type is called a "**pure**" node. A node that has duplicate data 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?

width > 1.5
[50, 50, 50]

No   Yes

????
[50, 46, 3]

????
[0, 4, 47]

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?

width > 1.5
[50, 50, 50]

No

Yes

????
[50, 46, 3]

????
[0, 4, 47]

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??

[50, 50, 50]

No   Yes

????
[50, 9, 0]

????
[0, 41, 50]

Defining a Best Feature

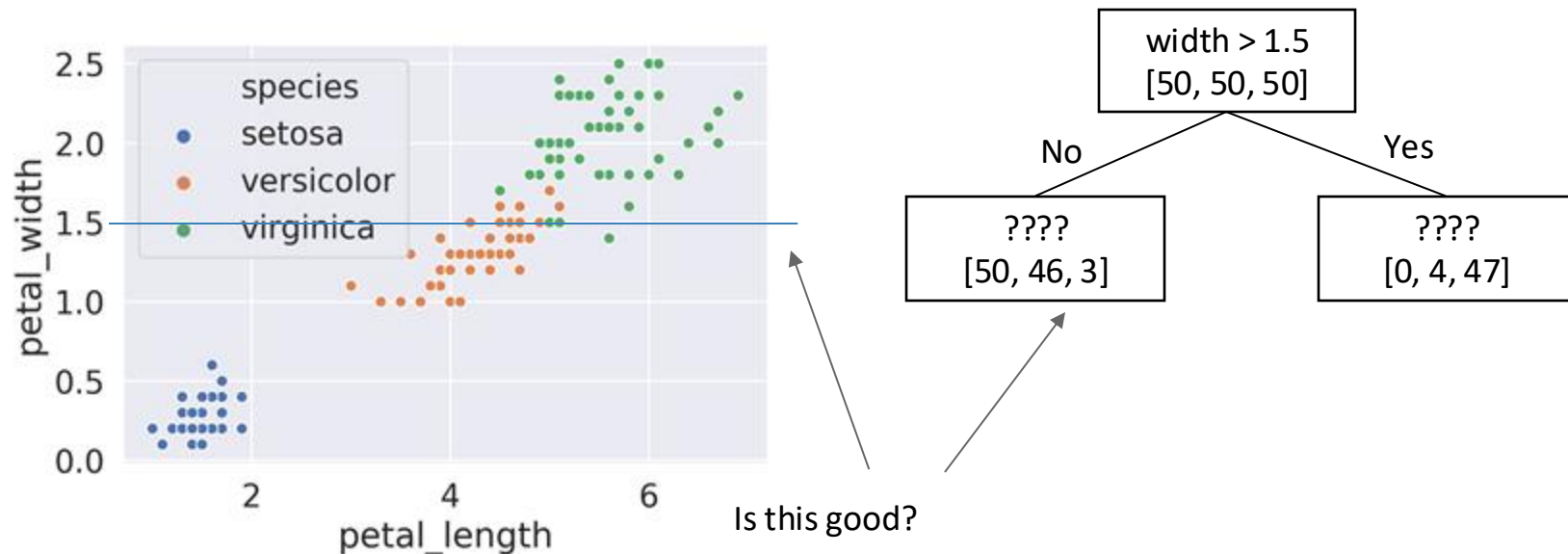Question: Which feature and split value is best?

- 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

width > 0.8
[50, 50, 50]

No — ???? [50, 0, 0]

Yes — ???? [0, 50, 50]

Maximum uncertainity about the next outcome.
It also means that the system is perfectly uniform,
the probability of the two possible outcomes is the same

$$H(X) = -p \log_2 p - (1-p) \log_2 (1-p)$$

Entropy

Probability

P(Y=1)
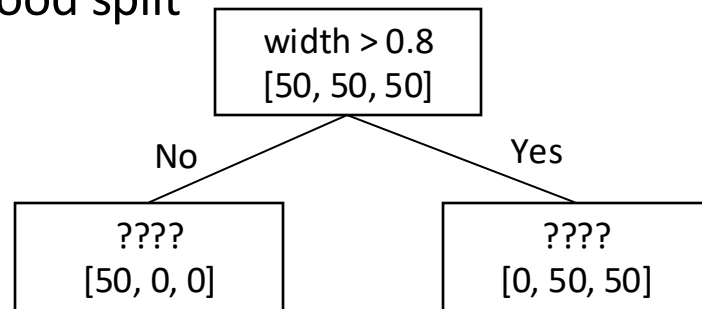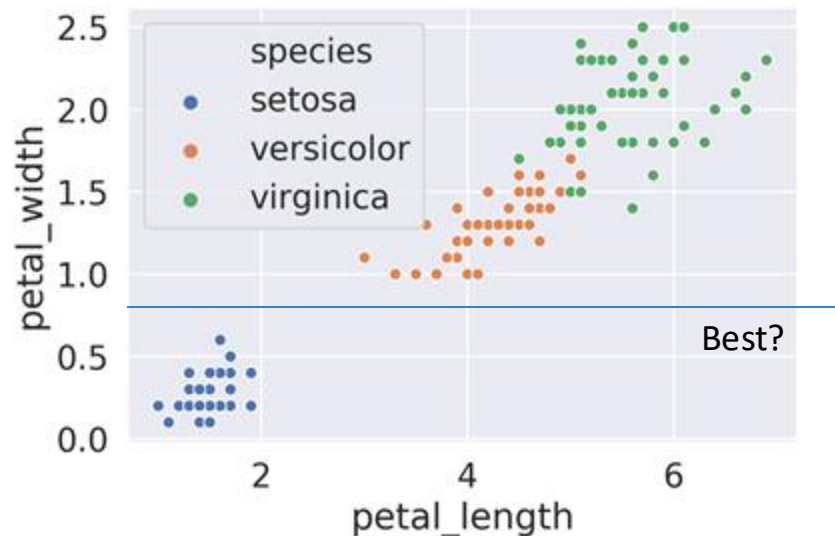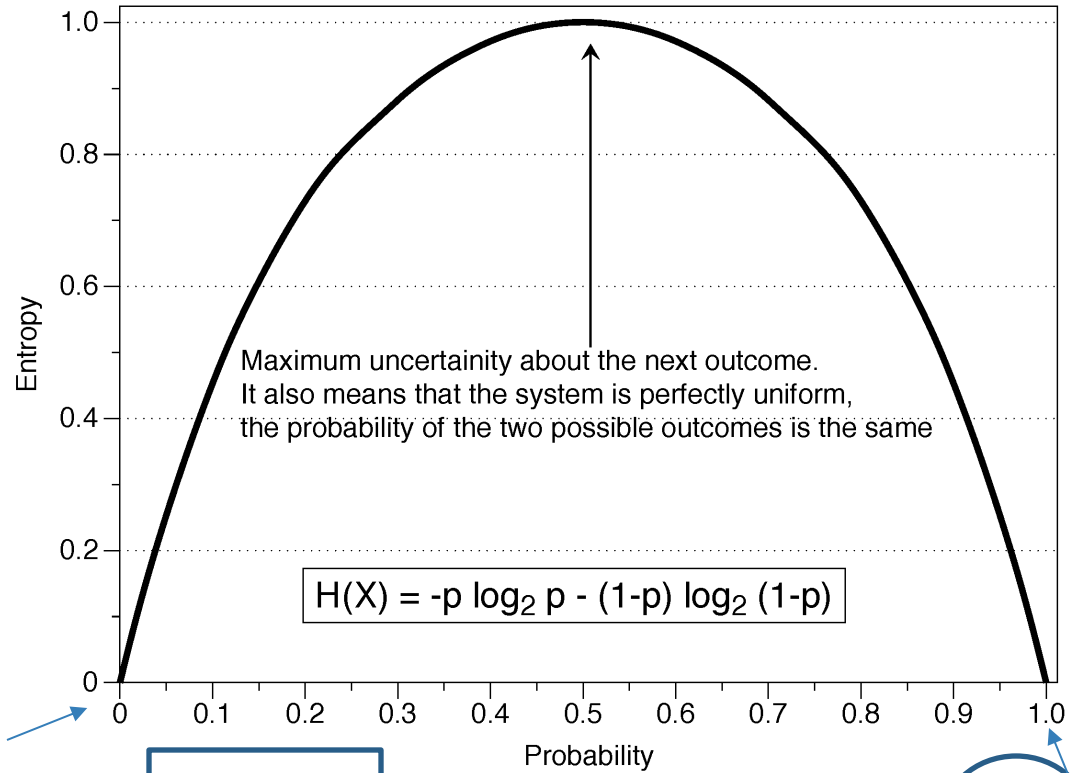
Node Entropy

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 as:

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

For example, S for the top node is:
$-0.31 \log_2 0.31 - 0.33 \log_2 0.33 - 0.36 \log_2 0.36$
$= 0.52 + 0.53 + 0.53 = 1.58$


```
sepal_length ≤ 5.45
gini = 0.665
samples = 110
value = [34, 36, 40]
class = virginica
```

True   S=1.58   False

```
sepal_width ≤ 2.85
gini = 0.245
samples = 36
value = [31, 4, 1]
class = setosa
```

```
sepal_length ≤ 6.15
gini = 0.534
samples = 74
value = [3, 32, 39]
class = virginica
```
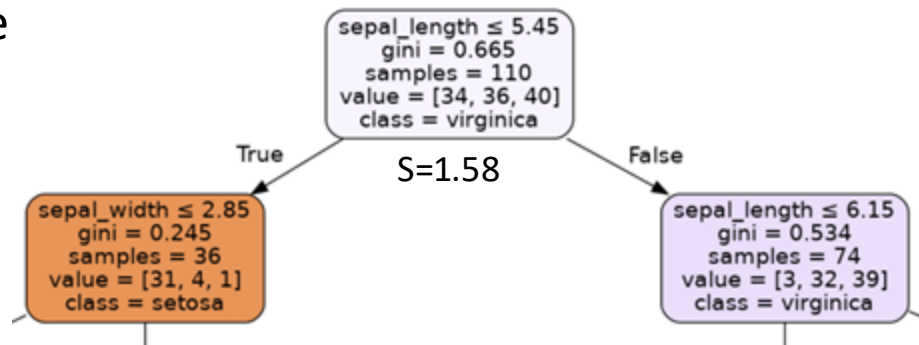
How unpredictable (uncertain).a node is

Test Your Understanding

What is p_0?


```
sepal_length ≤ 5.45
gini = 0.665
samples = 110
value = [34, 36, 40]
class = virginica
```

True   S=1.58   False

```
sepal_width ≤ 2.85
gini = 0.245
samples = 36
value = [31, 4, 1]
class = setosa
```

```
sepal_length ≤ 6.15
gini = 0.534
samples = 74
value = [3, 32, 39]
class = virginica
```
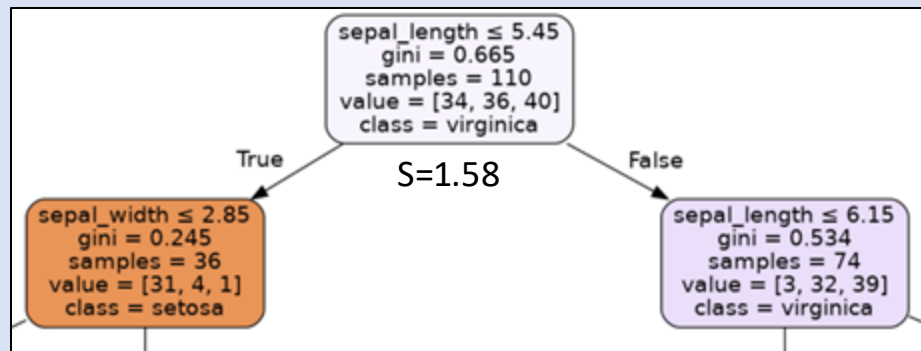
What is the entropy of the node on the left with [31, 4, 1] in each class?

- Try writing out an expression, or try to compute an exact value

$$S = -\sum_{c} p_c \log_2 p_c$$

Test Your Understanding

What is the entropy of the node on the left 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

Define the entropy S of a node as:

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

```
sepal_length ≤ 5.45
gini = 0.665
samples = 110
value = [34, 36, 40]
class = virginica
```

True   S=1.58   False

```
sepal_width ≤ 2.85
gini = 0.245
samples = 36
value = [31, 4, 1]
class = setosa
```

```
sepal_length ≤ 6.15
gini = 0.534
samples = 74
value = [3, 32, 39]
class = virginica
```

S=0.68

Can think of entropy as how unpredictable a node is:

- Low entropy means more predictable
- High entropy means more unpredictable
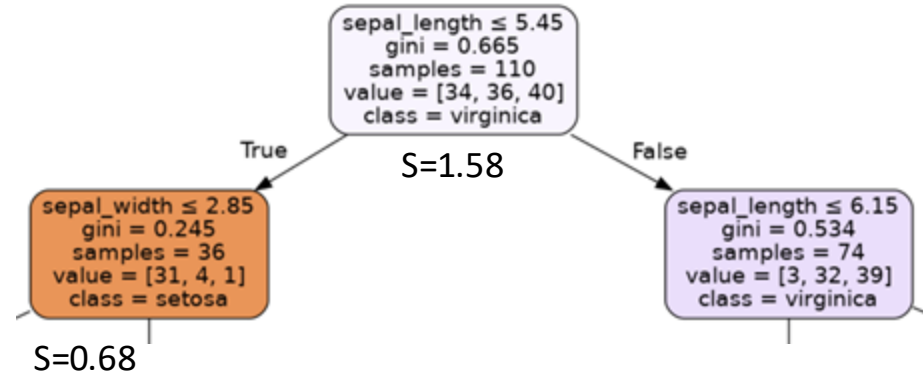
Exploring Entropy

Observations about entropy:

- A node where all data are part of the same class has zero entropy

  $-1 \log_2 1 = 0$

- A node where data are evenly split between two classes has entropy 1

  $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

- A node where data are evenly split between 3 classes has entropy 1.58
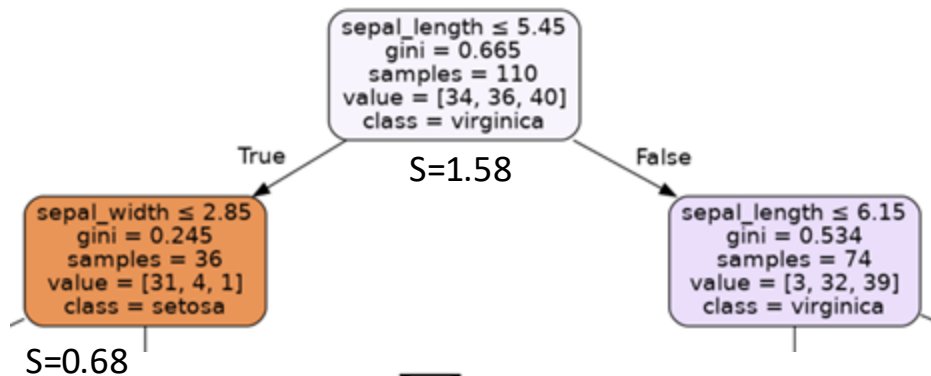
  $3 \times (-0.33 \log_2 0.33) = 1.58$

- A node where data are evenly split into C classes has entropy $\log_2 C$

  $C \times (-1/C \log_2 1/C) = -\log_2 1/C = \log_2 C$

sepal_length ≤ 5.45
gini = 0.665
samples = 110
value = [34, 36, 40]
class = virginica

True   S=1.58   False

sepal_width ≤ 2.85
gini = 0.245
samples = 36
value = [31, 4, 1]
class = setosa

S=0.68

sepal_length ≤ 6.15
gini = 0.534
samples = 74
value = [3, 32, 39]
class = virginica

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

Weighted Entropy as a Loss Function

We can use Weighted Entropy as a loss function in helping us decide which split to take
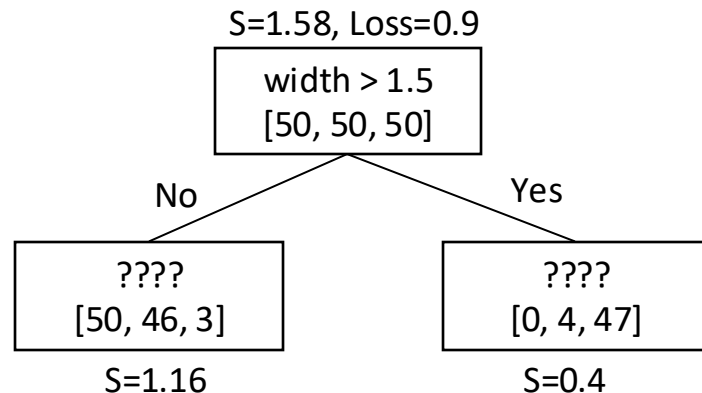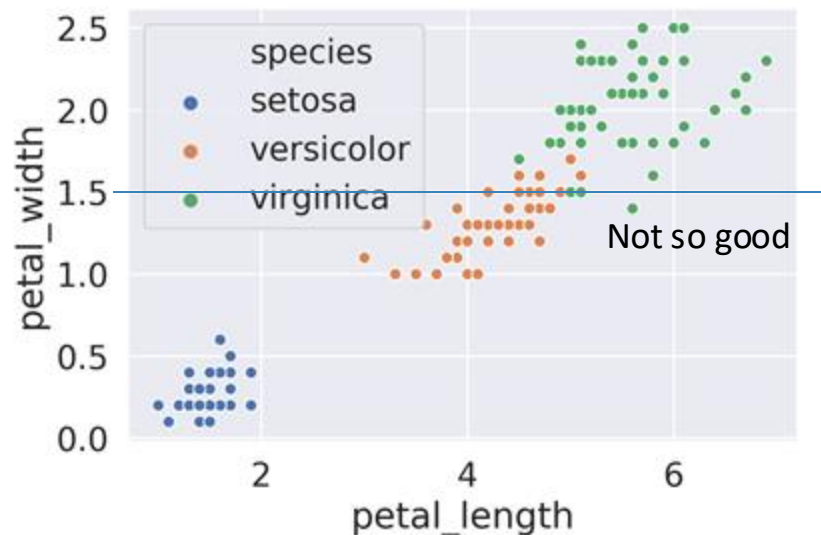
Suppose a given split results in two nodes X and Y with N_1 and N_2 total samples each. The loss of that split is given by:

$$L = \frac{N_1 S(X) + N_2 S(Y)}{N_1 + N_2}$$

Defining a Best Feature

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

- entropy([50, 46, 3]) = 1.16
- entropy([4, 47]) = 0.4
- Weighted average: 99/150 × 1.16 + 51/150 × 0.4 = **0.9**

Defining a Best Feature

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

- entropy([50, 9]) = 0.62
- entropy([41, 50]) = 0.99
- Weighted Average: **0.84**: Better than split choice #1!

S=1.58, Loss=0.84

[50, 50, 50]

No   Yes

????
[50, 9, 0]

S=0.62

????
[0, 41, 50]

S=0.99
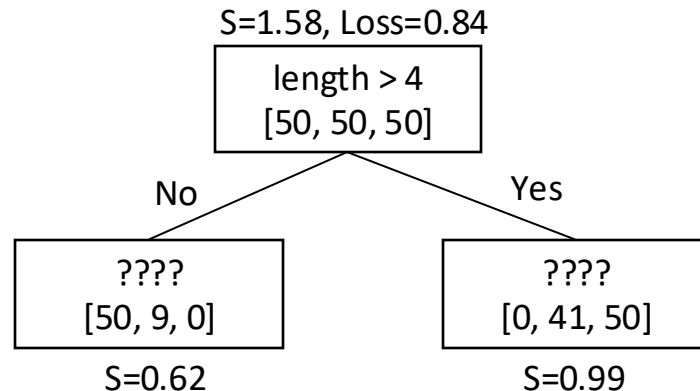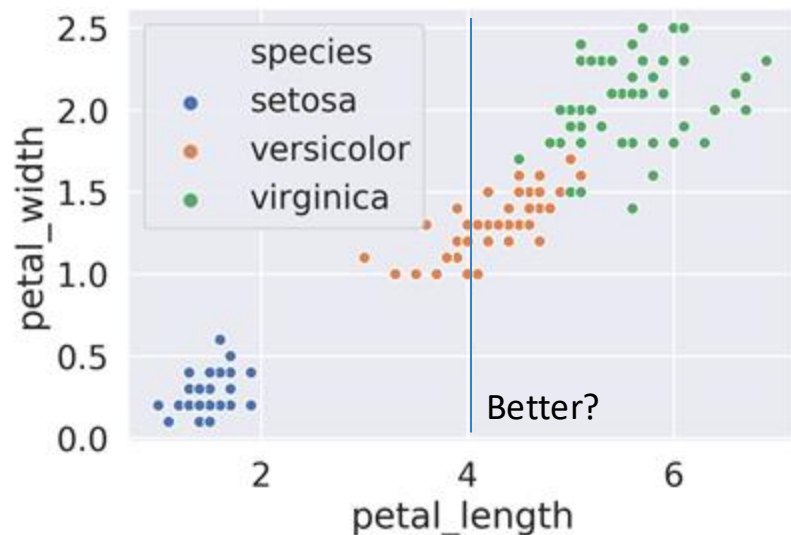
Defining a Best Feature

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

- entropy([2, 50, 50]) = 1.12
- entropy([48]) = 0
- Weighted average: **0.76**: Lower than split choice #2!
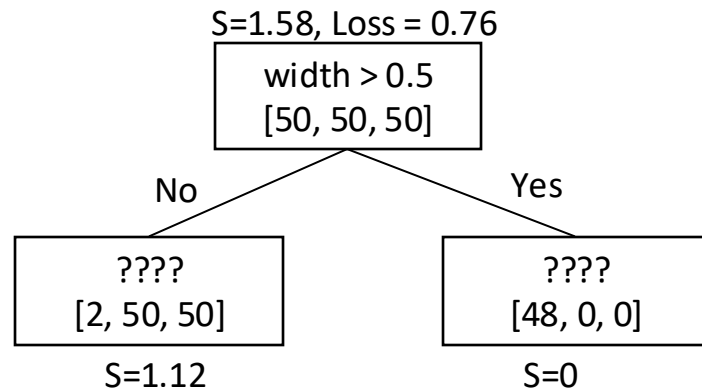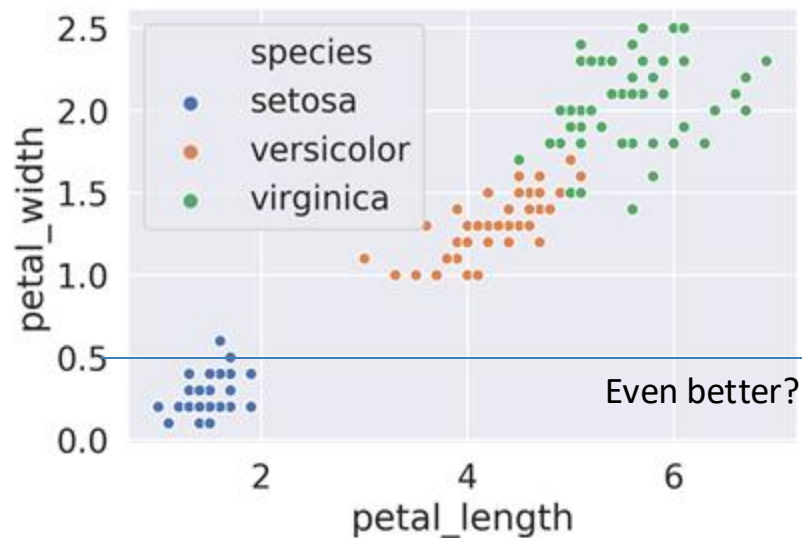
Defining a Best Feature

Split choice #4: width > 0.9. Compute entropy of child nodes:

- entropy([50, 50]) = 1
- entropy([50]) = 0
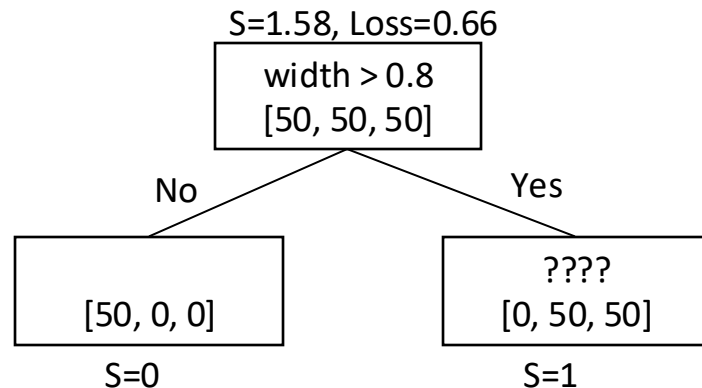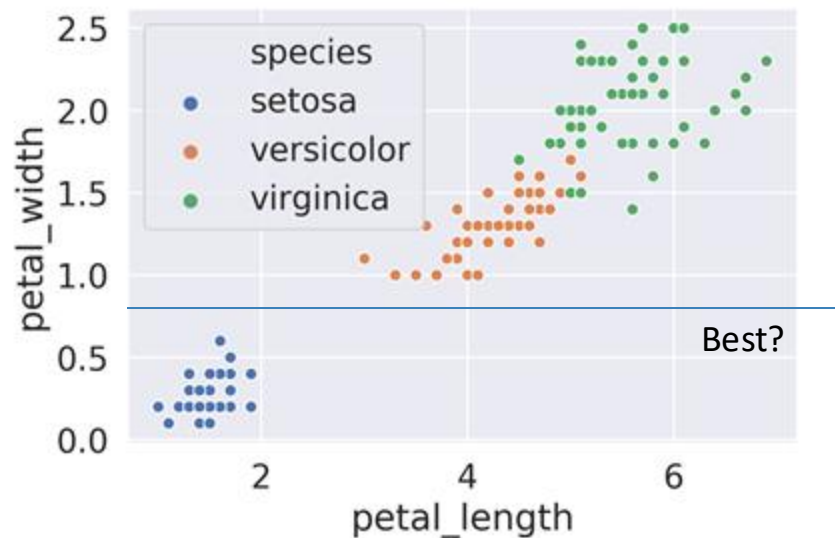- Weighted average: **0.66**: Lower than split choice #3!

S=1.58, Loss=0.66

width > 0.8
[50, 50, 50]

No   Yes

[50, 0, 0]
S=0

????
[0, 50, 50]
S=1

Decision Tree Generation

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 <u>**loss of the resulting split**</u> is **minimized**, e.g. x = petal_width, β = 0.8 has loss 0.66
    ○ **Split data into two nodes**, one where x < β, and one where x ≥ β

Notes: A node that has only one type is called a "**pure**" node. A node that has duplicate data that cannot be split is called "**unsplittable**"

Let's now turn our attention to avoiding overfitting

Restricting Decision Tree Complexity

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
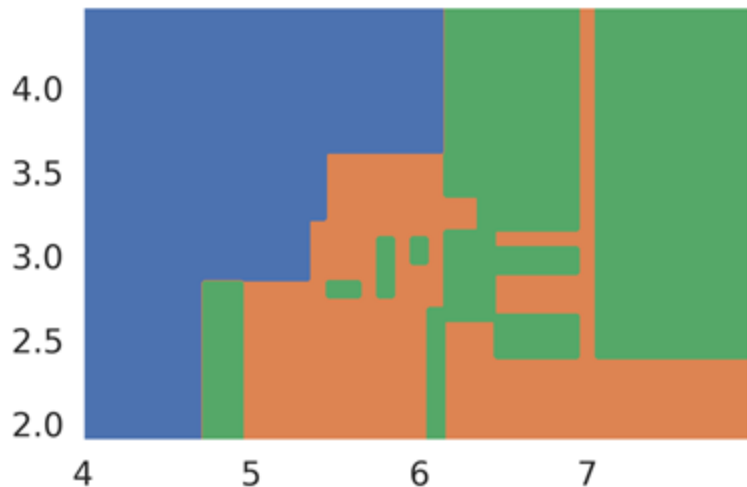
Regularization Terms and Decision Trees?

We can't just use the regularization term idea from linear models

- There is no global function being minimized
- Instead, the decision tree is built up node by node in a "greedy" fashion
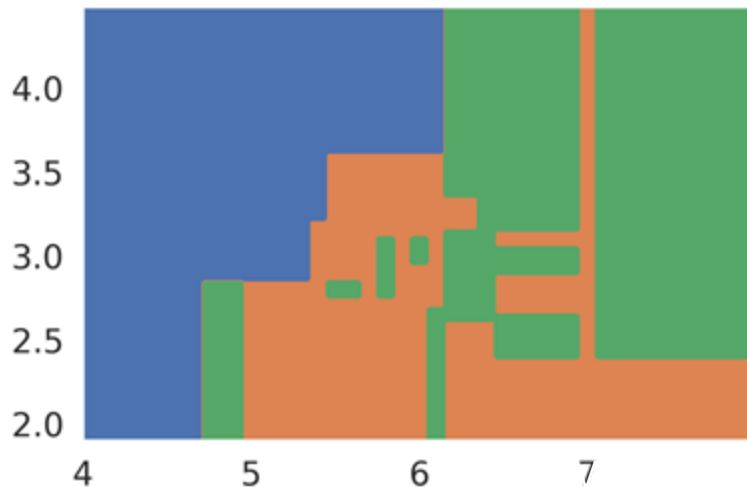
Approach 1: Preventing Growth

Approach 1: Set one or more special rules to prevent growth

Examples:

- 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: Pruning

Approach 2: Let tree fully grow, then cut off less useful branches of the tree.
For example, consider the highlighted branch below:

- Many rules that affect few points
- How can we avoid this?

Specific Pruning Example

One way to prune:

- Before creating the tree, set aside a validation set
- If replacing a node by its most common prediction has no impact on the validation error, then don't split that node
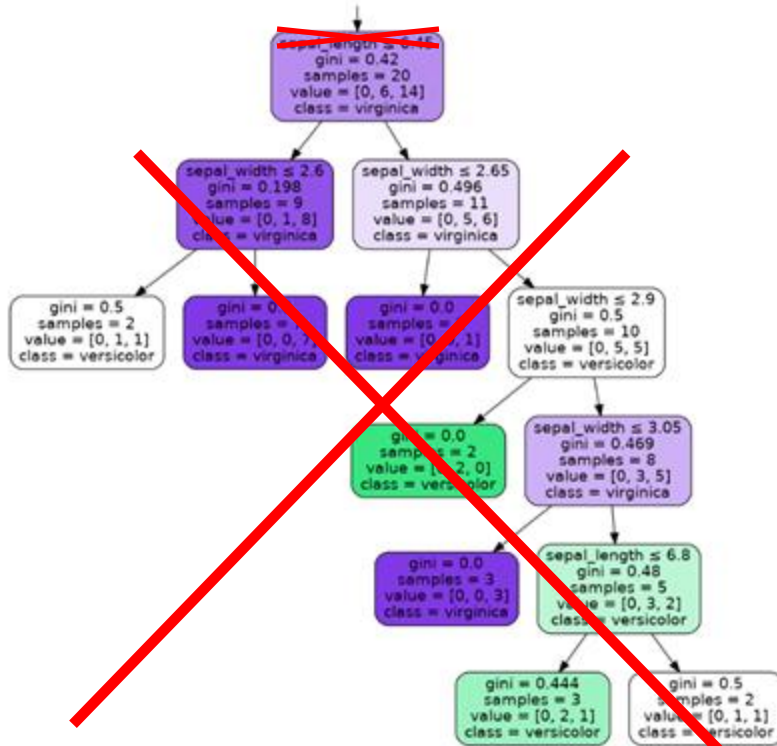
Overfitting and Our Algorithm

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
- Approach 2: Allow full growth then prune branches afterwards

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

Random Forests

Random Forests: Harnessing Variance

As we've seen, fully-grown decision trees will almost always overfit data

- Low model bias, high model variance
- In other words, small changes in dataset will result in very different decision tree
- Example: Two models on the right trained on different subsets of the same data

Random Forest Idea: Build many decision trees and have them vote
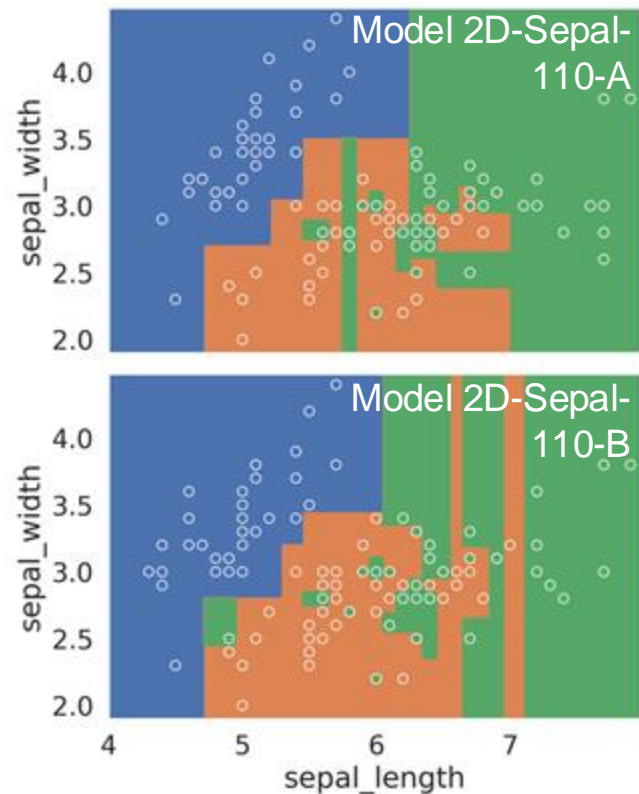
Random Forests: Harnessing Variance

What do we mean by vote?

- For a given x/y, use whichever prediction is most popular

Consider example at right with 9 models

- 6 votes orange, 3 votes green
- Random forest prediction is orange
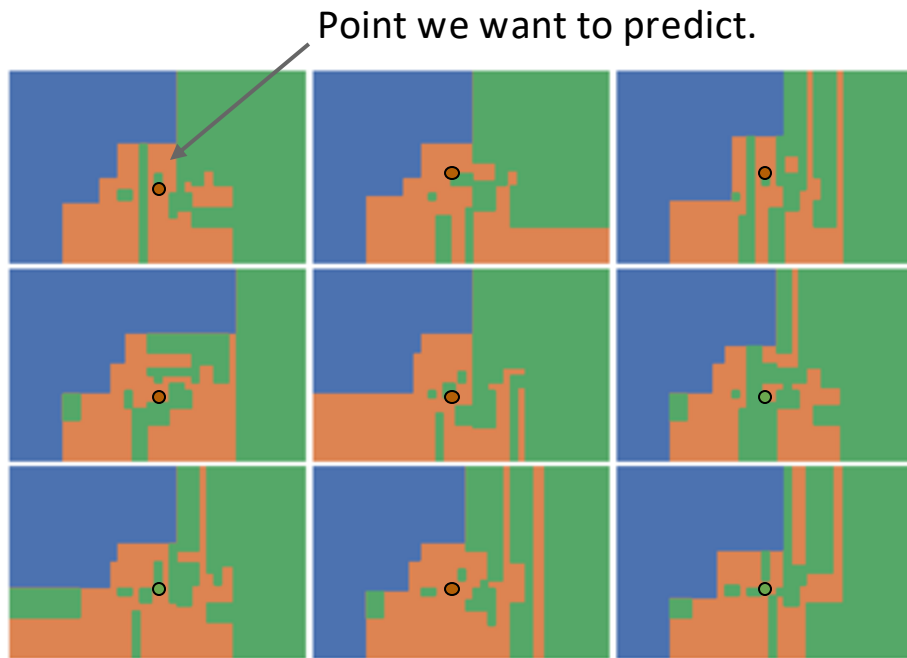
Point we want to predict.

Building Many Trees

Big fundamental problem: We only have one training set

How can we build many trees using one training set?

Building Many Trees

Big fundamental problem: We only have one training set.

How can we build many trees using one training set?

Bagging

Bagging: Short for Bootstrap AGGregatING

- Generate bootstrap resamples of training data (draw samples with replacements)
- Fit one model for each resample
- Final model = average predictions of each small model
- Invented by Leo Breiman in 1994 (Berkeley Statistics!)

Random Forests

- Bagging often isn't enough to reduce model variance!
  - Decision trees often look very similar to each other
  - E.g., one strong feature always used for first split
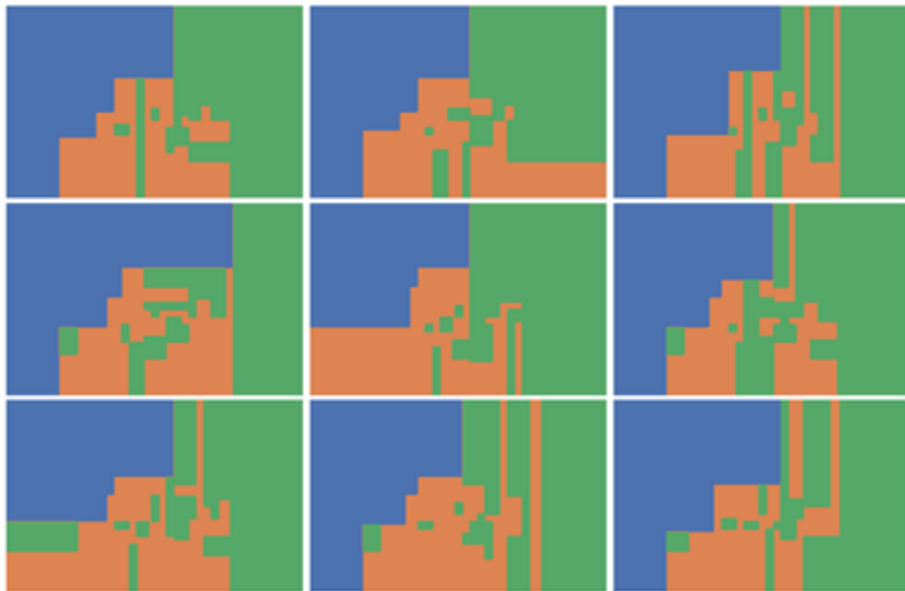
Random Forests

- Bagging often isn't enough to reduce model variance!
  - Decision trees often look very similar to each other and thus make similar predictions
  - Ensemble will still have low bias and high model variance
  - E.g. one strong feature always used for first split
- Idea: Only use a sample of m features at each split
  - Usually m = sqrt(p) for decision trees used for classification
  - Here p is the number of features
- Algorithm creates individual trees, each overfit in a different way
  - The hope is that the the overall forest has low variance

Random Forests Algorithm

- *Bootstrap training* data **T** times. For each resample, fit a decision tree
- To predict, ask the T decision trees for their predictions and take majority vote

This approach has two hyperparameters T and m, where m is the number of features selected randomly for each split

Avoiding Overfitting with Heuristics

We've seen many approaches to avoid overfitting decision trees

- Preventing growth
- Pruning
- Random forests

These ideas are generally "heuristic"

- Not provably best or mathematically optimal
- Instead, they are just ideas that somebody thought sounded good, implemented, then found to work in practice acceptably well

Why Random Forests?

- Versatile: does both regression and classification
- Nonlinear decision boundaries without complicated feature engineering
- Doesn't overfit as often as other nonlinear models (e.g. polynomial features)

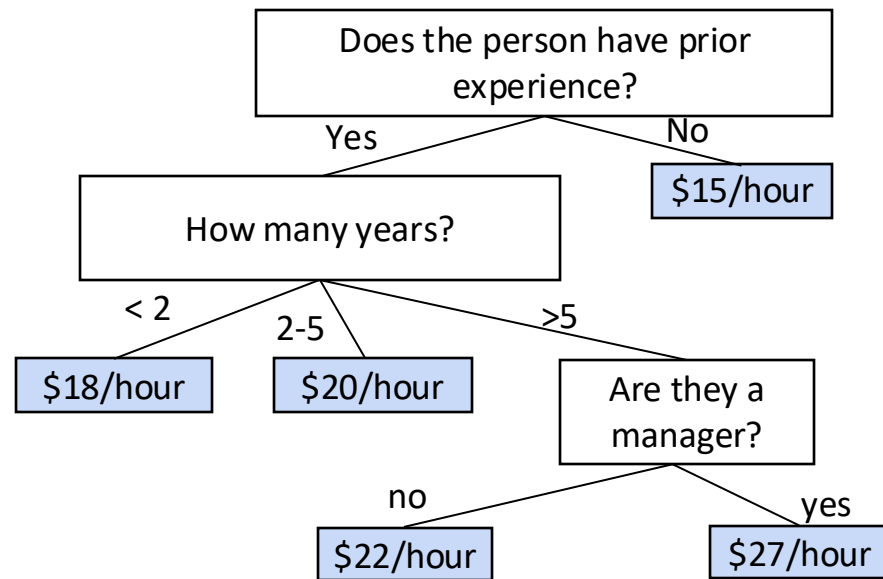# Beyond Decision Trees for Classification

Trees for Regression

In earlier lectures we saw how we could use a logistic regression model for classification

- This lecture: We saw decision trees as an alternative technique for classification

We could do the same exercise for regression

- Rather than using a linear model, we could build a regression tree

Does the person have prior experience?

Yes → How many years?
No → $15/hour

How many years?
< 2 → $18/hour
2-5 → $20/hour
>5 → Are they a manager?

Are they a manager?
no → $22/hour
yes → $27/hour

Summary

Decision trees provide an alternate non-linear framework for classification and regression

- The underlying principle is fundamentally different
- Decision boundaries can be more complex
- Danger of overfitting is high

Keeping complexity under control is not nearly as mathematically elegant and relies on heuristic rules

- Hard constraints
- Pruning rules
- Random forests
  - Very interesting application of bootstrapping

Summary

In practice, you will see that there are actually even more conceptual frameworks for regression and classification:

- Linear models
- Decision trees
- Nearest neighbors
- Support vector machines.
- Bayes Nets
- Perceptrons
  - Neural networks / deep learning
- And many many many many more

Machine learning classes will go over these ideas in more detail

- How they work (and the tradeoffs for using them)
  are beyond our class