COMP47460

# Decision Trees

Aonghus Lawlor
Derek Greene

School of Computer Science
Autumn 2018

# Overview

- What is a Decision Tree?
- Feature Selection
  - Good v Bad features
  - Information Theory - Entropy
- How to build a Decision Tree?
  - ID3 top-down algorithm
  - Information Gain
- Decision Trees in Weka

# Decision Tree Learning

- Goal: Build a tree model that splits the training set into subsets in which all examples have the same class.
- A feature can be used to split the training set, one for each value or range of the feature…
  - e.g. insured = {true, false}
  - e.g. income = {low, average, high}
  - e.g. height < 6ft, height ≥ 6ft
- If necessary, each subset can be split again using another feature, and so on until all examples have the same class.
- Selecting a feature on which to split can be done using a measure based on uncertainty.
- Once the tree is built, we can use it to quickly classify new input examples (i.e. eager learning).

# Example: Apples v Pears

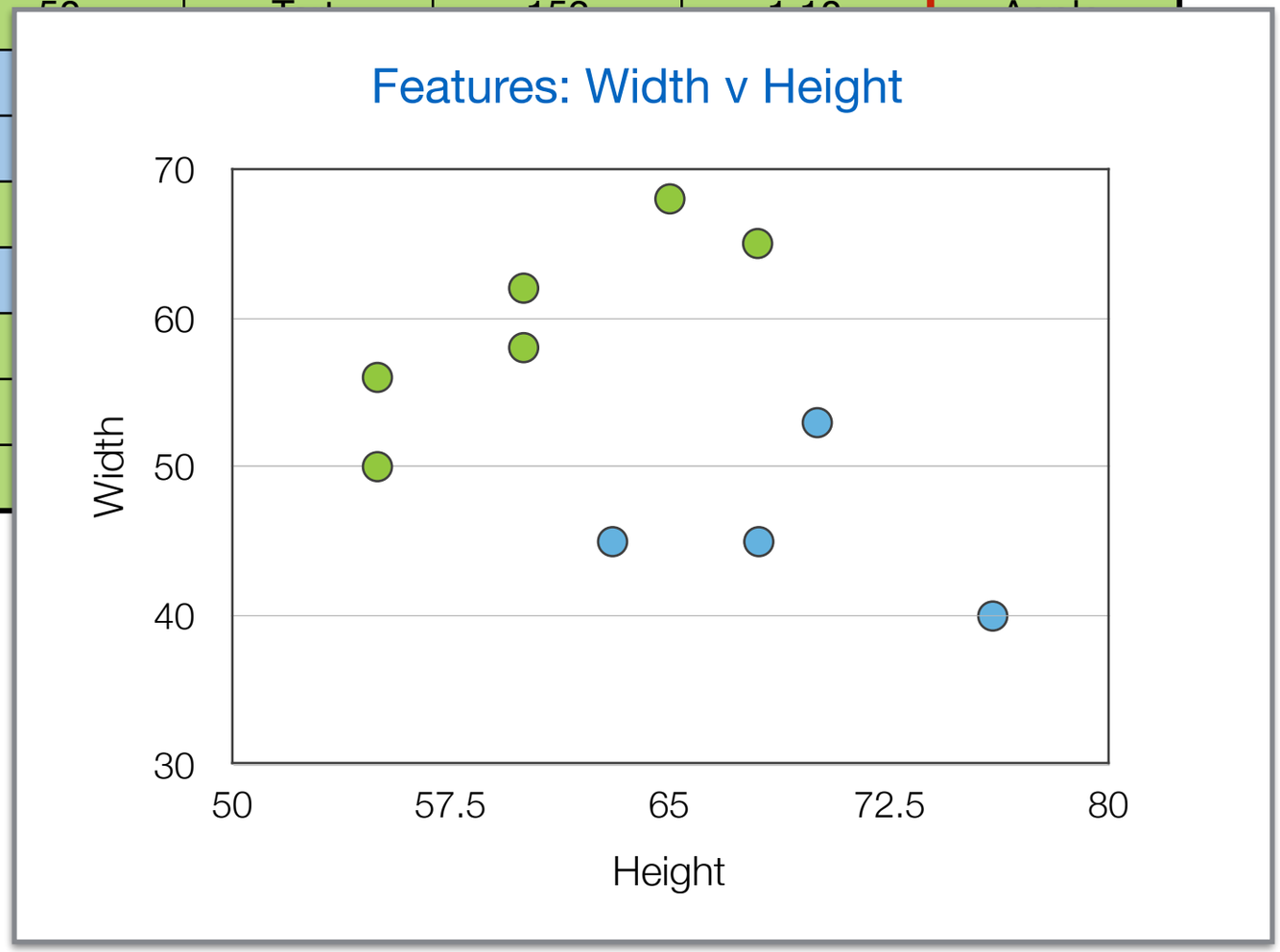
- 10 training examples such that: each has a class label (“apple” or “pear”), and each is described with 6 features.

| *Example* | *Colour* | *Height* | *Width* | *Taste* | *Weight* | *H/W* | *Class* |
|---|---|---|---|---|---|---|---|
| **1** | 210 | 60 | 62 | Sweet | 186 | 0.97 | Apple |
| **2** | 220 | 70 | 53 | Sweet | 180 | 1.32 | Pear |
| **3** | 215 | 55 | 50 | Tart | 152 | 1.10 | Apple |
| **4** | 180 | 76 | 40 | Sweet | 152 | 1.90 | Pear |
| **5** | 220 | 68 | 45 | Sweet | 153 | 1.51 | Pear |
| **6** | 160 | 65 | 68 | Sour | 221 | 0.96 | Apple |
| **7** | 215 | 63 | 45 | Sweet | 140 | 1.40 | Pear |
| **8** | 180 | 55 | 56 | Sweet | 154 | 0.98 | Apple |
| **9** | 220 | 68 | 65 | Tart | 221 | 1.05 | Apple |
| **10** | 190 | 60 | 58 | Sour | 175 | 1.03 | Apple |

# Example: Apples v Pears

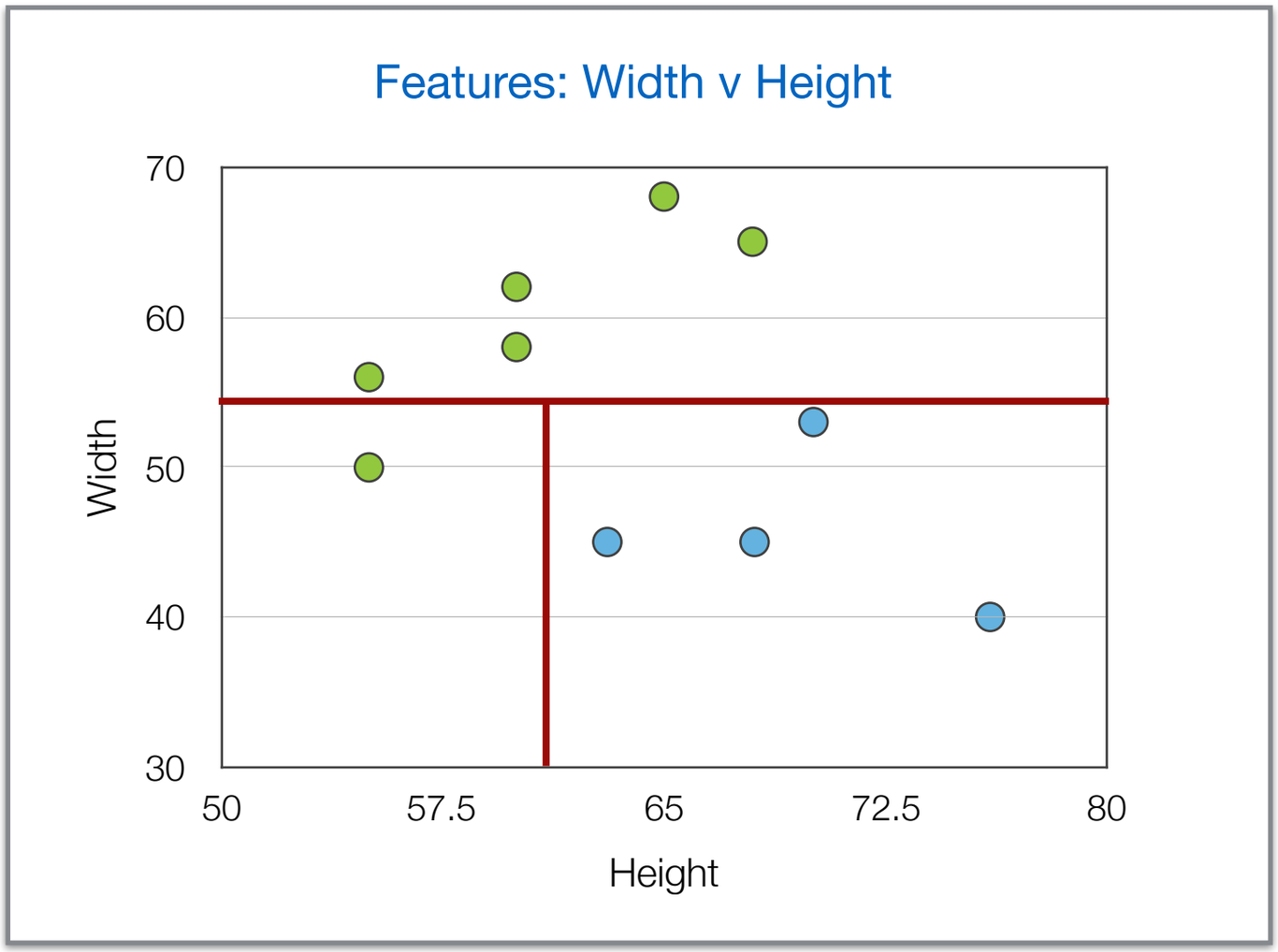
- 10 training examples such that: each has a class label (“apple” or “pear”), and each is described with 6 features.

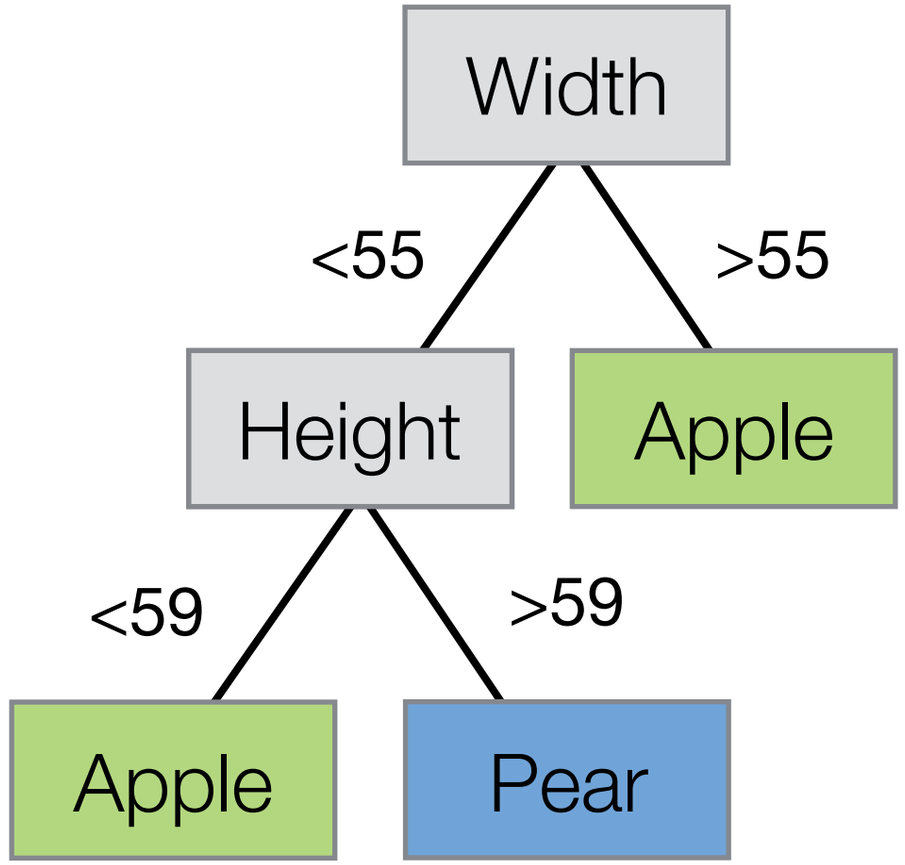
| *Example* | *Colour* | *Height* | *Width* | *Taste* | *Weight* | *H/W* | *Class* |
|---|---|---|---|---|---|---|---|
| **1** | 210 | 60 | 62 | Sweet | 186 | 0.97 | Apple |
| **2** | 220 | 70 | 53 | Sweet | 180 | 1.32 | Pear |
| **3** | 215 | 55 | | | | | |
| **4** | 180 | 76 | | | | | |
| **5** | 220 | 68 | | | | | |
| **6** | 160 | 65 | | | | | |
| **7** | 215 | 63 | | | | | |
| **8** | 180 | 55 | | | | | |
| **9** | 220 | 68 | | | | | |
| **10** | 190 | 60 | | | | | |



# Example: Apples v Pears

- Simple decision tree for classifying Apples v Pears using only 2 features: {Height, Weight}



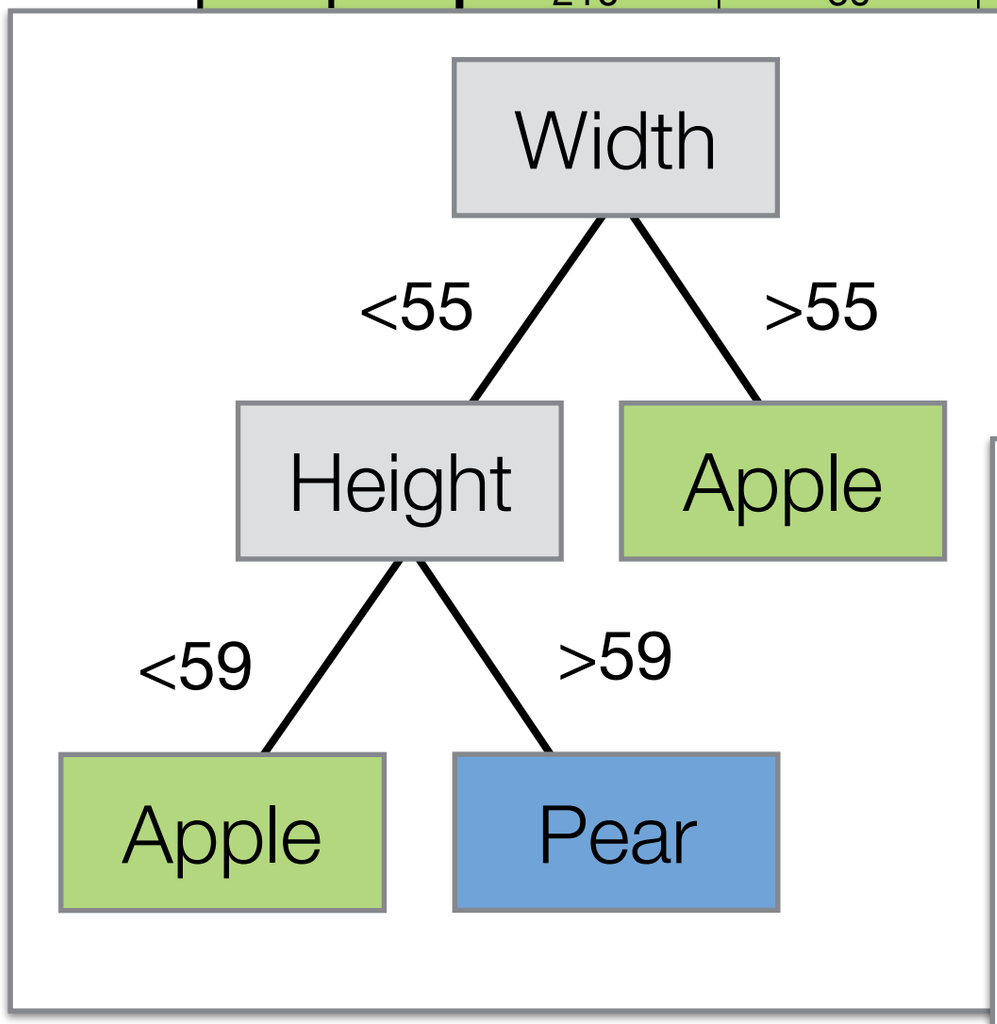



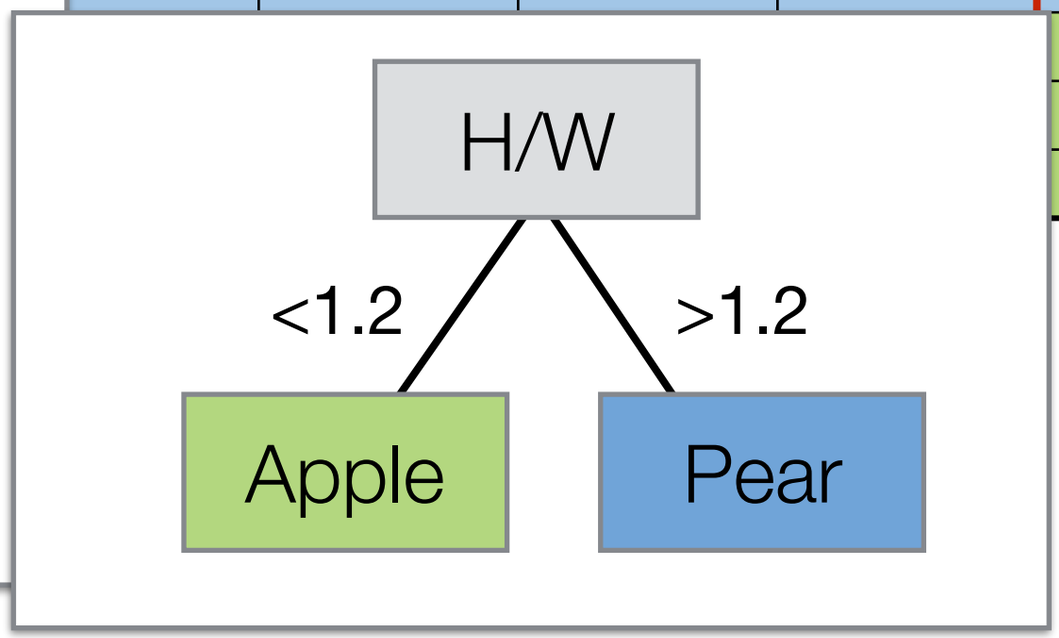

Just 2 features can split the data based on these decision rules.

# Example: Apples v Pears

- 10 training examples such that: each has a class label (“apple” or “pear”), and each is described with 6 features.

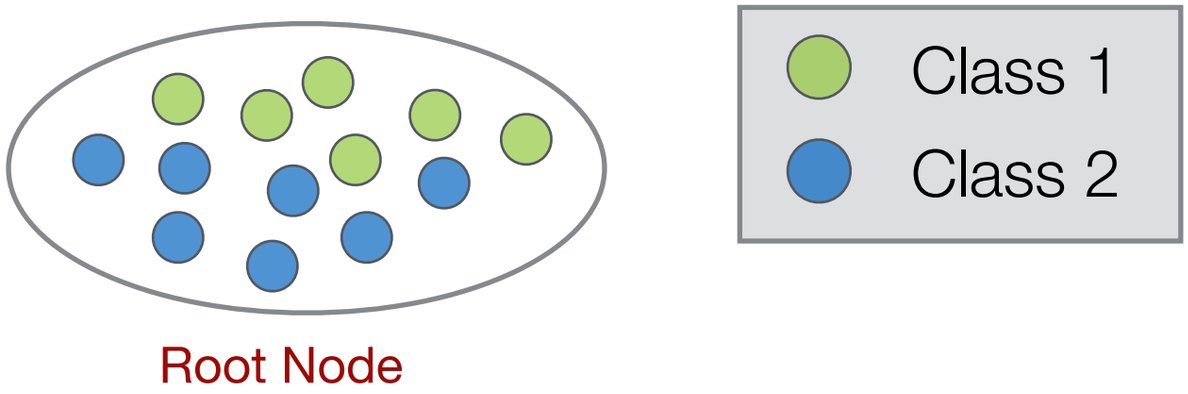
| *Example* | *Colour* | *Height* | *Width* | *Taste* | *Weight* | *H/W* | *Class* |
|---|---|---|---|---|---|---|---|
| 1 | 210 | 60 | 62 | Sweet | 186 | 0.97 | Apple |
| | | | 53 | Sweet | 180 | 1.32 | Pear |
| | | | 50 | Tart | 152 | 1.10 | Apple |
| | | | 40 | Sweet | 152 | 1.90 | Pear |
| | | | 45 | Sweet | 153 | 1.51 | Pear |
| | | | 68 | Sour | 221 | 0.96 | Apple |
| | | | 45 | Sweet | 140 | 1.40 | Pear |
| | | | | | | | Apple |
| | | | | | | | Apple |
| | | | | | | | Apple |



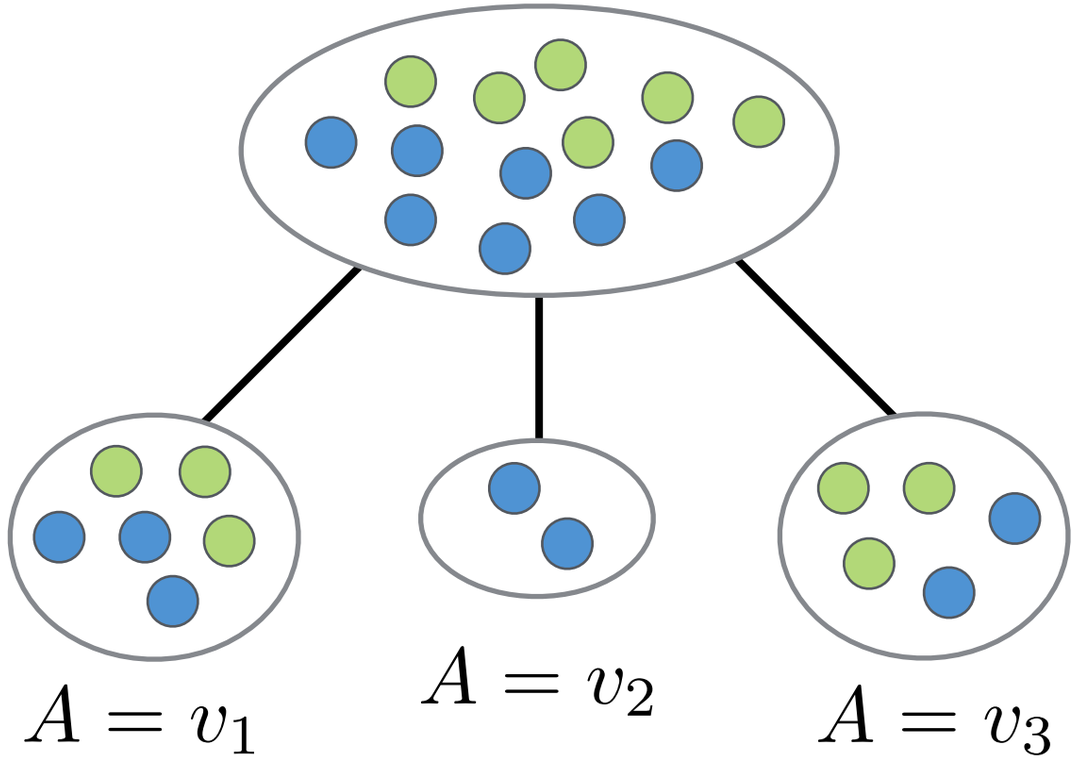
# Decision Tree Learning

1. Initially all examples in the training set are placed at the root node of the tree.




Root Node

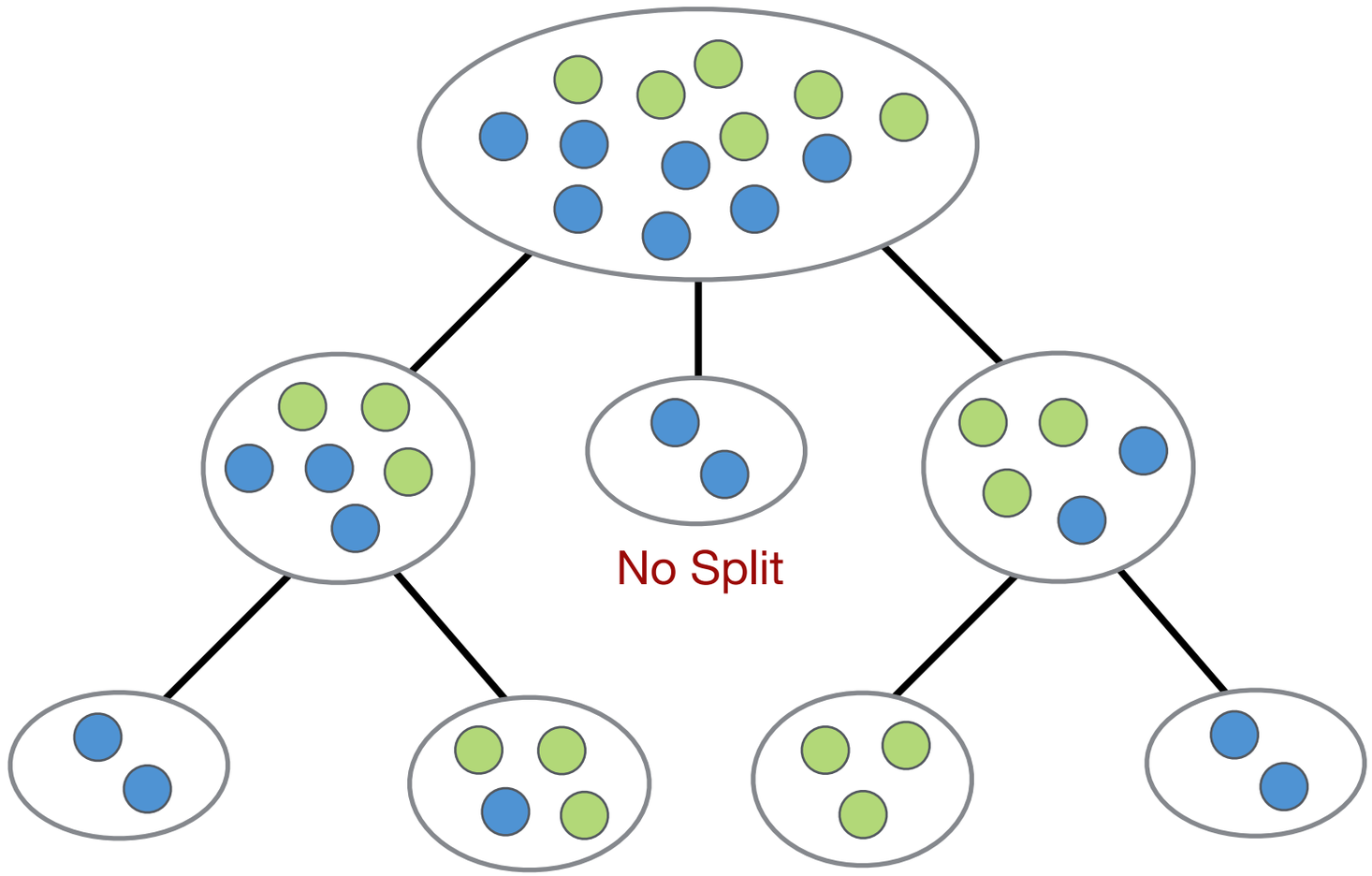
2. One of the available features (*A*) is now used to split the examples at the root node into two or more child nodes containing subsets of examples.




Decision Rules

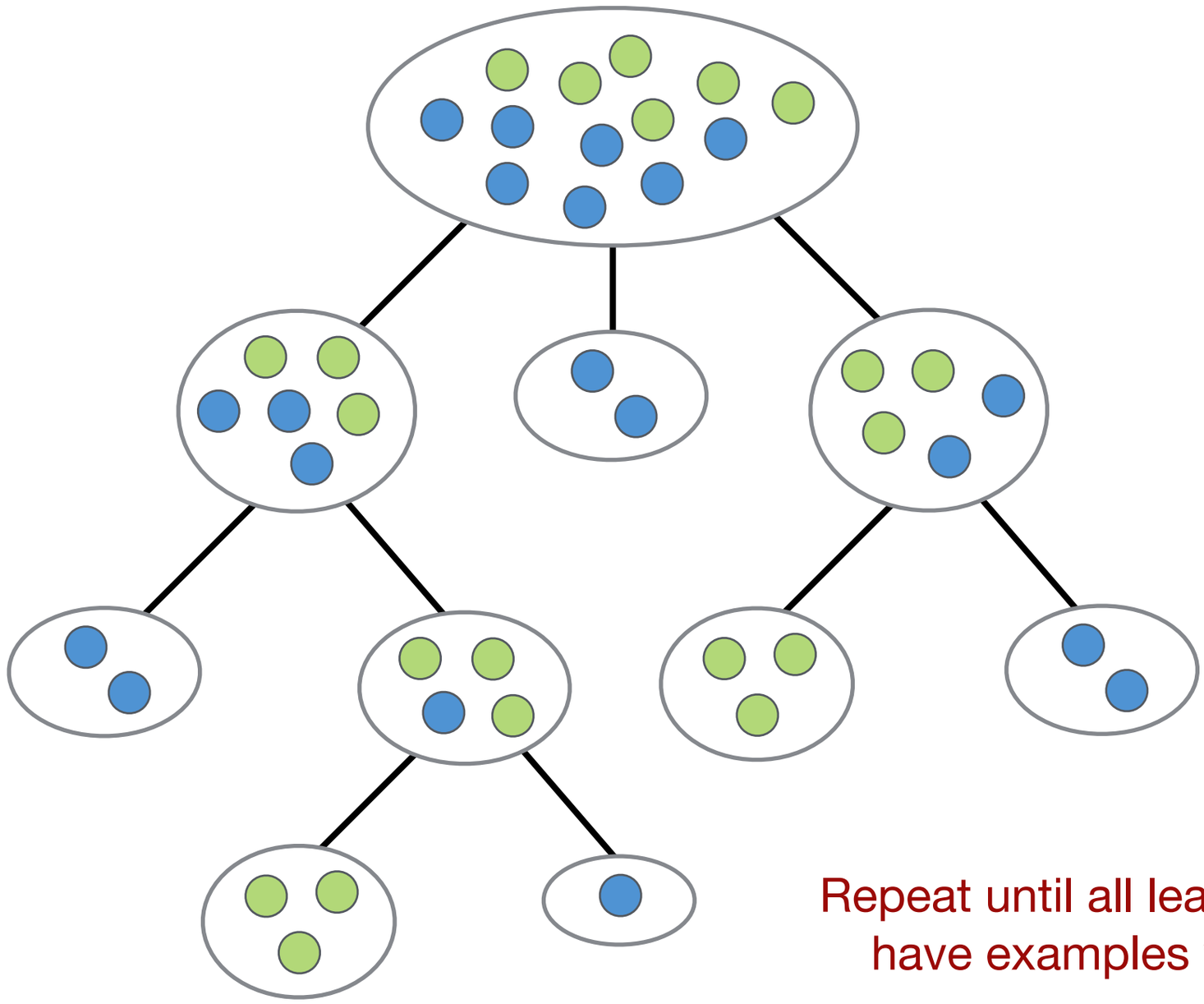
# Decision Tree Learning

3. The same process is now applied to each child node, except for any child node at which all examples have the same class.



# Decision Tree Learning

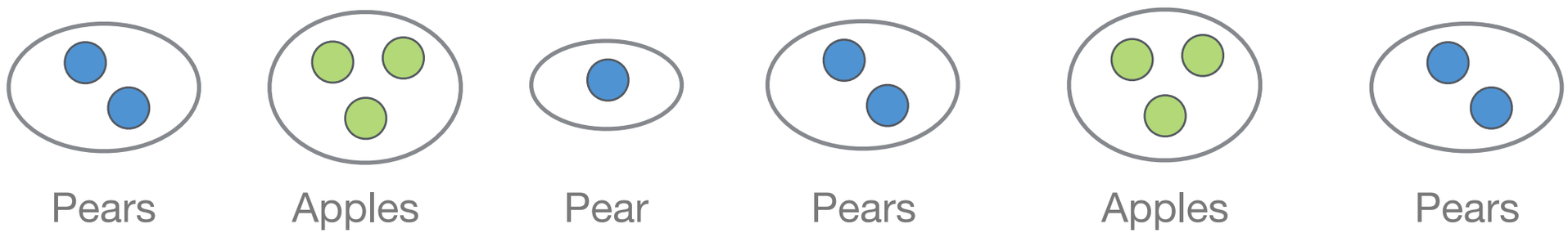
4. This continues until the training set has been divided into subsets in which all the examples have the same class.



Repeat until all leaf nodes in the tree have examples with same class

# Node Purity

- A tree node is pure if all examples at that node have the same class label.




- A decision tree in which all the leaf nodes are pure can always be constructed provided there are no clashes in the data.
  - i.e. examples having the same “description” in terms of features, but with different class labels.
- Most decision tree algorithms use some measure of node (im)purity to choose features to split when building the tree. This measure guides the learning process.

# Decision Trees Example

Q. "Will a customer wait for a restaurant table?"
*Russell & Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2009.*

Binary classification task (WillWait = {Yes,No}), with examples described by 10 different features:

| Feature | Description |
|---|---|
| *Alternate* | Is a suitable alternative restaurant nearby? (Yes/No) |
| *Bar* | Does the restaurant have a comfortable bar area to wait in? (Yes/No) |
| *Fri/Sat* | True on Fridays and Saturdays, False otherwise. |
| *Hungry* | Is the customer hungry? (Yes/No) |
| *Patrons* | How many people are in the restaurant: {None, Some, Full}? |
| *Price* | Restaurant's price range: {€, €€, €€€} |
| *Raining* | Is it raining outside? (Yes/No) |
| *Reservation* | Has the customer made a reservation? (Yes/No) |
| *Type* | Type of restaurant: {French, Italian, Thai, Burger} |
| *WaitEstimate* | Length of wait estimated by the host: {0-10, 10-30, 30-60, > 60 minutes} |

# Decision Trees Example

Q. "Will a customer wait for a restaurant table?"

*Russell & Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2009.*

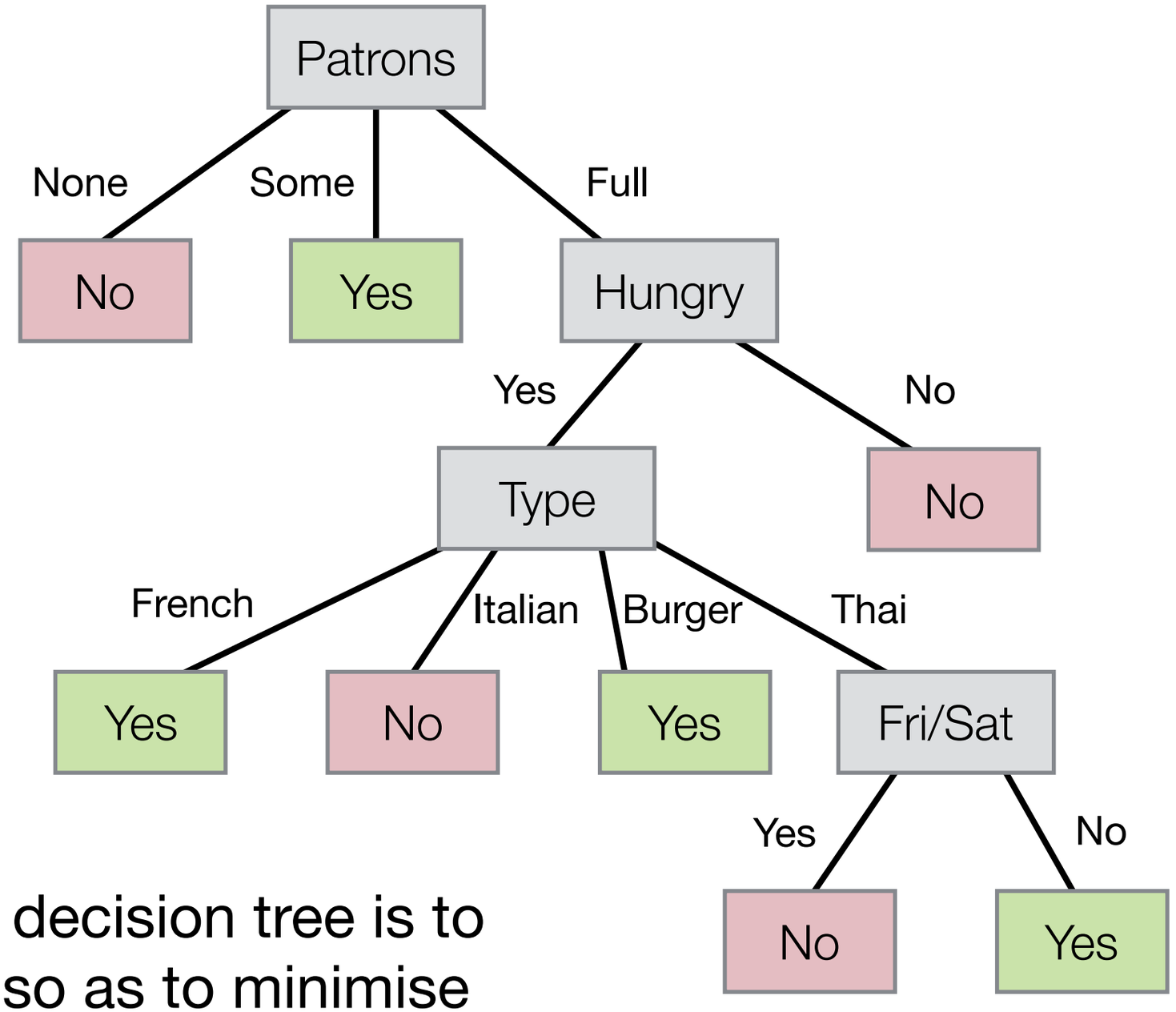
Binary classification task (WillWait = {Yes,No}), with examples described by 10 different features:

| Example | Alternate | Bar | Fri/Sat | Hungry | Patrons | Price | Raining | Reservation | Type | WaitEst | WillWait? |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *1* | Yes | No | No | Yes | Some | €€€ | No | Yes | French | 0-10 | Yes |
| *2* | Yes | No | No | Yes | Full | € | No | No | Thai | 30-60 | No |
| *3* | No | Yes | No | No | Some | € | No | No | Burger | 0-10 | Yes |
| *4* | Yes | No | Yes | Yes | Full | € | No | No | Thai | 10-30 | Yes |
| *5* | Yes | No | Yes | No | Full | €€€ | No | Yes | French | >60 | No |
| *6* | No | Yes | No | Yes | Some | €€ | Yes | Yes | Italian | 0-10 | Yes |
| *7* | No | Yes | No | No | None | € | Yes | No | Burger | 0-10 | No |
| *8* | No | No | No | Yes | Some | €€ | Yes | Yes | Thai | 0-10 | Yes |
| *9* | No | Yes | Yes | No | Full | € | Yes | No | Burger | >60 | No |
| *10* | Yes | Yes | Yes | Yes | Full | €€€ | No | Yes | Italian | 10-30 | No |
| *11* | No | No | No | No | None | € | No | No | Thai | 0-10 | No |
| *12* | Yes | Yes | Yes | Yes | Full | € | No | No | Burger | 30-60 | Yes |

➡ How do we build a "good" decision tree for this data set?

# Decision Trees - Objective

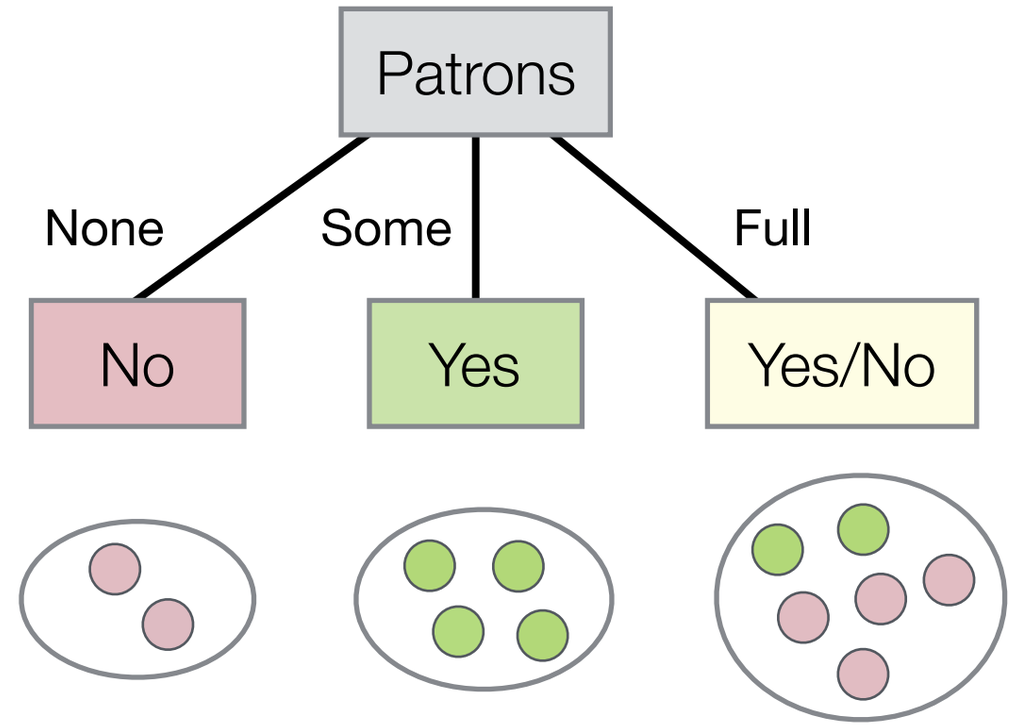
A “good” decision tree will classify all examples correctly using as few tree nodes as possible.



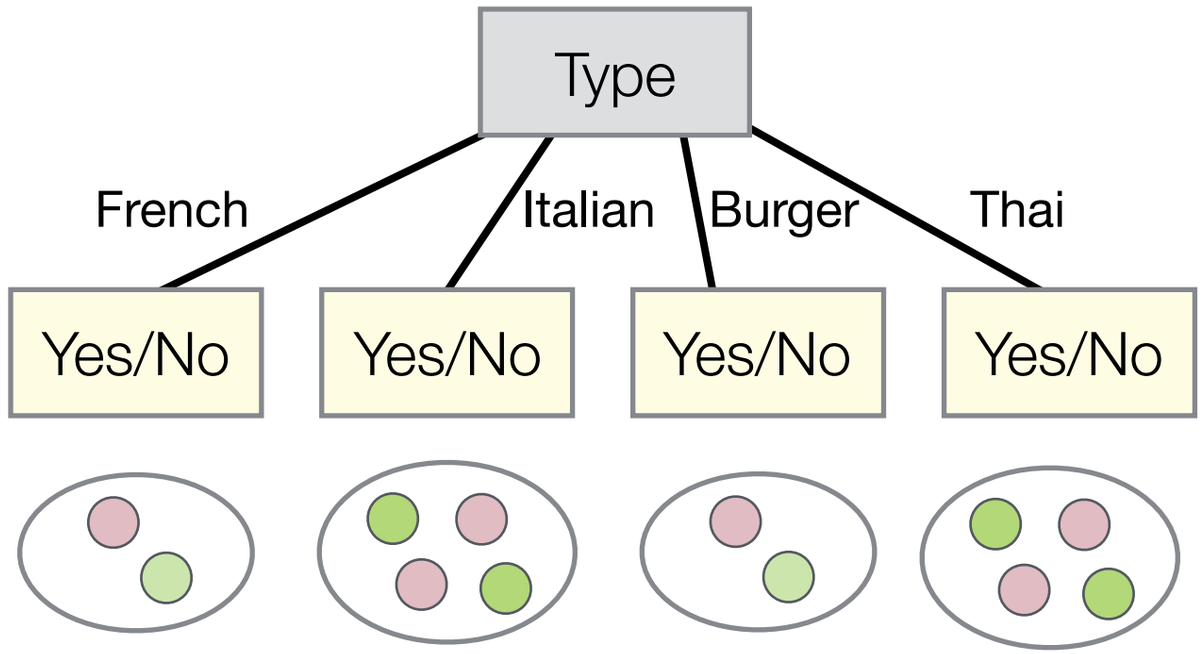

Objective in building a decision tree is to choose good features so as to minimise the depth of the tree.

# Good v Bad Features

- Good Features:
  A perfect feature divides examples into categories of one class
  ⇒ high purity




- Bad Features:
  A poor choice of feature produces categories of mixed classes
  ⇒ high impurity



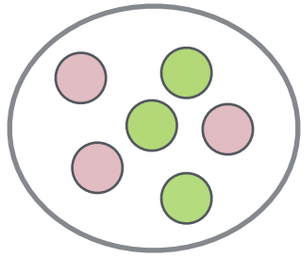
# Feature Selection

- Goal: Find good features which divide examples into categories of a single class.
- Feature selection algorithms have been developed which use impurity as an objective to guide feature selection (e.g. Entropy, Gini impurity).
- **Common selection strategy in decision trees:**
  - For each feature, some measure of impurity is applied to the current set of tree nodes.
  - The feature that maximises the reduction in impurity is selected as the next most useful feature.

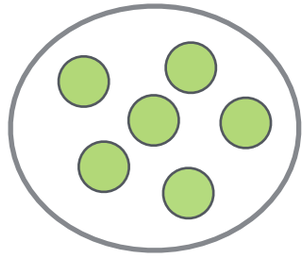
# Entropy

- Entropy: In information theory, a measure of uncertainty around a source of information. Low for predictable sources, higher for more random sources.

- In the context of decision trees, entropy provides a measure of impurity - how uncertain we are about the decision for a given set of examples.



High uncertainty → High entropy



Low uncertainty → Low entropy

- **Definition:**

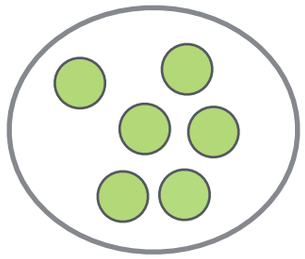
Entropy of a set of examples $S$ with class labels $\{C_1, \ldots, C_n\}$ :

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

where $p_i$ is the relative frequency (probability) of class $C_i$.

# Entropy Examples

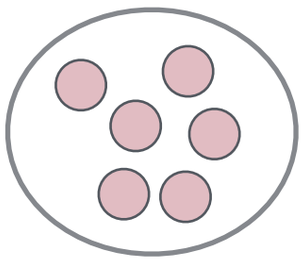
- The lowest possible entropy (i.e. zero) occurs when all examples have the same class label.
- The highest entropy occurs when we are most uncertain.



$$p_1 = 6/6 = 1.0 \qquad p_2 = 0/6 = 0.0$$

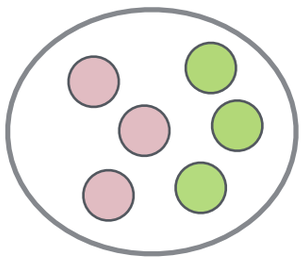
NB: Define $\log_2(0)=0$

$$H(S) = -((1 \times \log_2(1)) + (0 \times \log_2(0))) = -(0 + 0) = 0$$



$$p_1 = 0/6 = 0.0 \qquad p_2 = 6/6 = 1.0$$

$$H(S) = -((0 \times \log_2(0)) + (1 \times \log_2(1))) = -(0 + 0) = 0$$



$$p_1 = 3/6 = 0.5 \qquad p_2 = 3/6 = 0.5$$

$$H(S) = -((0.5 \times \log_2(0.5)) + (0.5 \times \log_2(0.5))) = -(-0.5 - 0.5) = 1$$

# Top-Down Induction of Decision Trees

- ID3 Algorithm: Popular algorithm which repeatedly builds a decision tree from the top down (Quinlan, 1986).
- Start with an empty tree and set of all training examples *S*.

ID3( $S$ ):

- IF all examples in $S$ belong to the same class $C$ THEN
  - Return new leaf node and label it with class $C$.
- ELSE
  - Select a feature $A$ based on some feature selection criterion.
  - Generate a new tree node with $A$ as the test feature.
  - FOR EACH value $v_i$ of $A$:
    - Let $S_i \subset S$ contain all examples with $A = v_i$.
    - Build subtree by applying ID3( $S_i$ )

# Criterion: Information Gain

- Information Gain (IG): Popular information theoretic approach for selecting features in decision trees, based on entropy.
- Measures a feature's overall impact on entropy when used to split a set of training examples into two or more subsets.
  - How much information do we learn by splitting on the feature?
  - How much is the reduction in entropy?

- **Definition:**

IG for feature $A$ that splits a set of examples $S$ into $\{S_1, \ldots, S_m\}$ :

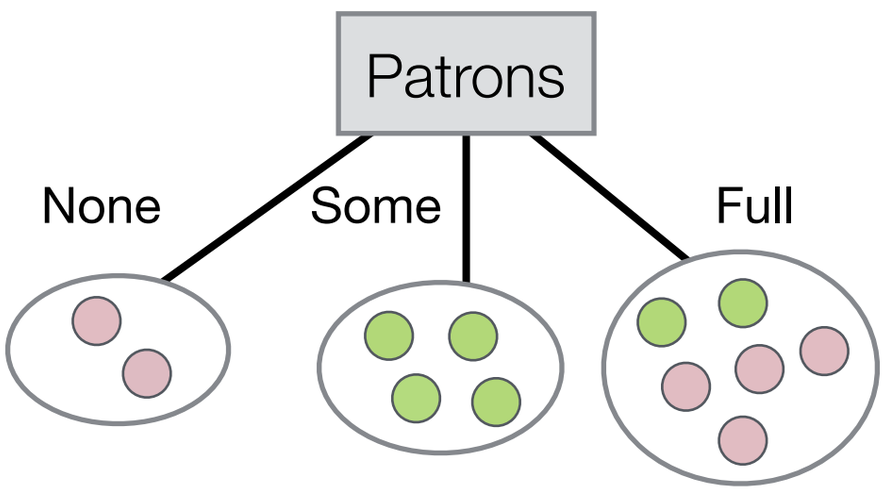
$$IG(S, A) = \text{(original entropy) - (entropy after split)}$$

$$IG(S, A) = H(S) - \sum_{i=1}^{m} \frac{|S_i|}{|S|} H(S_i)$$

Each subset is weighted in proportion to its size

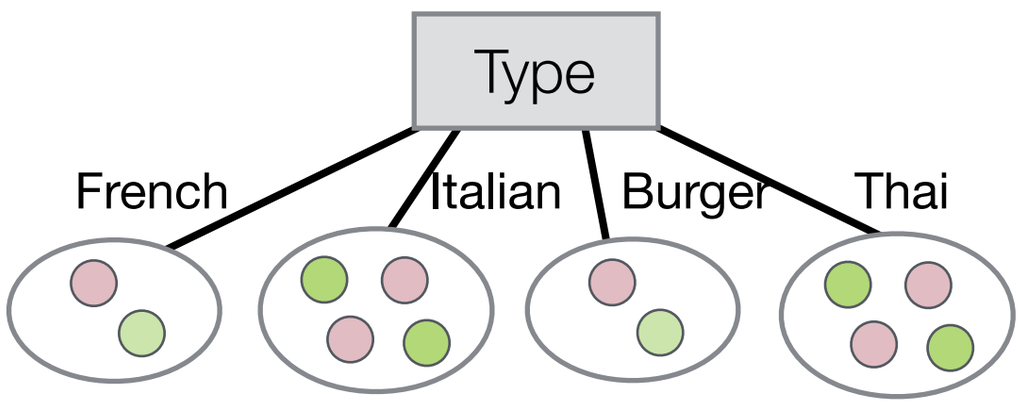
# Information Gain Example

- Previous example: Initial training set has 6 Yes, 6 No examples.
- Which feature should we select to split at the root node?




$$IG = H([\frac{6}{12}, \frac{6}{12}]) - \left( \frac{2}{12} H([0,1]) + \frac{4}{12} H([1,0]) + \frac{6}{12} H([\frac{2}{6}, \frac{4}{6}]) \right)$$

$$IG(\text{Patrons}) = 1 - 0.459 = 0.541$$



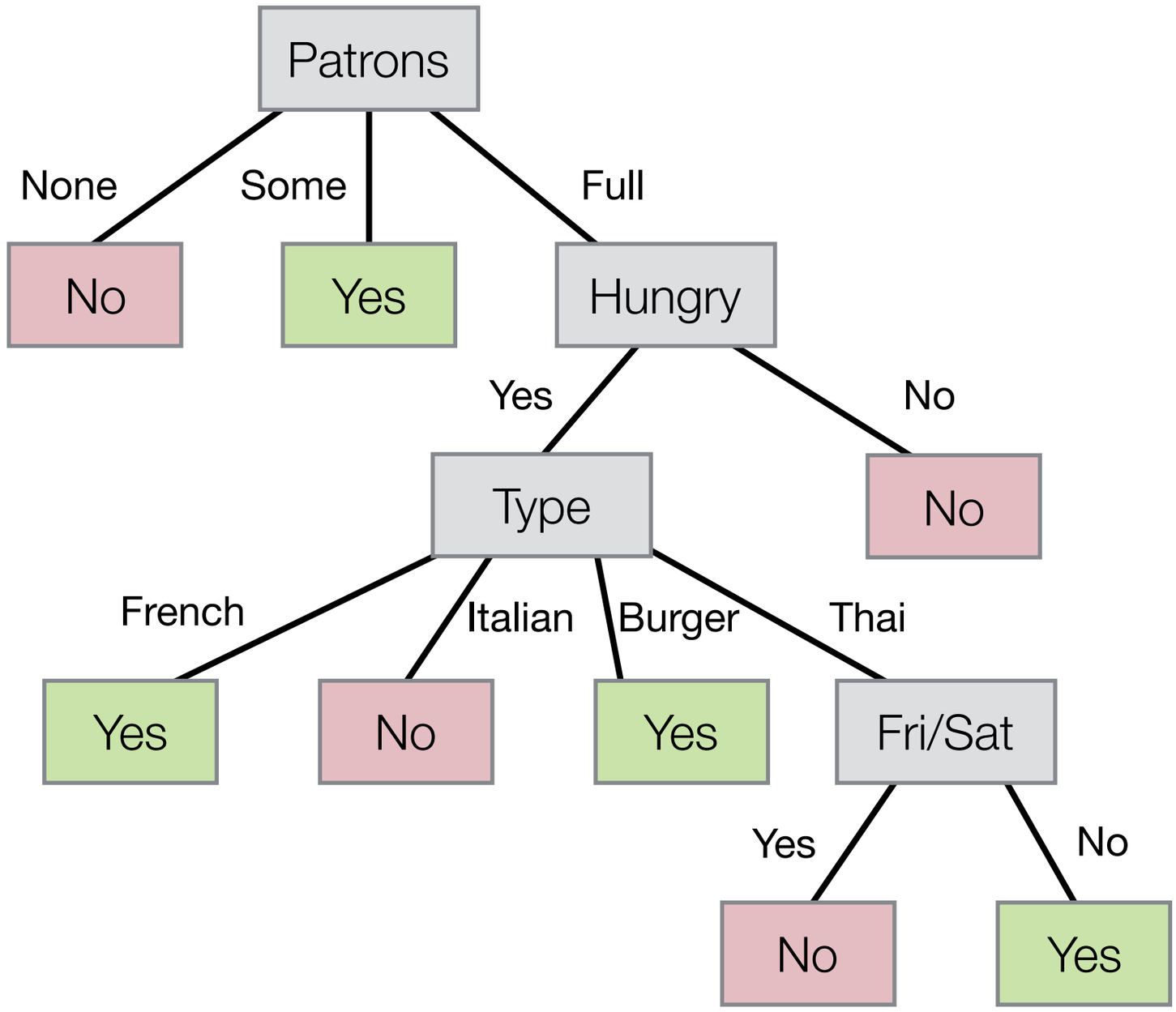

$$IG = H([\frac{6}{12}, \frac{6}{12}]) - \left( \frac{2}{12} H([\frac{1}{2}, \frac{1}{2}]) + \frac{4}{12} H([\frac{2}{4}, \frac{2}{4}]) + \frac{2}{12} H([\frac{1}{2}, \frac{1}{2}]) + \frac{4}{12} H([\frac{2}{4}, \frac{2}{4}]) \right)$$

$$IG(\text{Type}) = 1 - 1 = 0$$

➡ Feature “Patrons” has higher IG, so a better choice for splitting.

# Information Gain Example

- ID3 repeats the feature selection + splitting process until all examples have the same class, or no features are left to split.

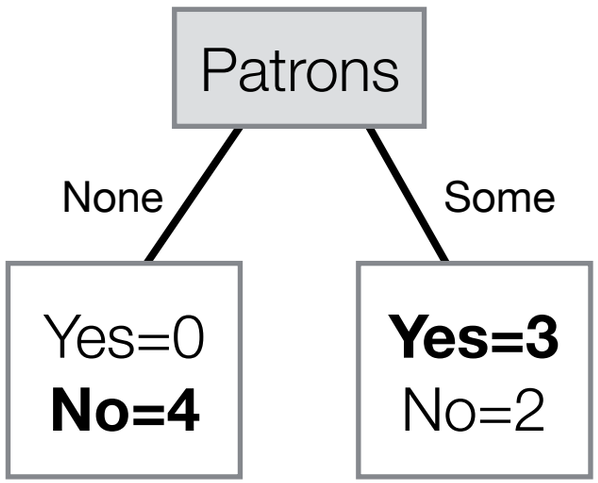


# Handling Inconsistent Data

- Inconsistent training data occurs when two identical examples from the training set have different labels:

```
true,female,town,false,yes,no,yes,no = YES
true,female,town,false,yes,no,yes,no = NO
```

- When building a tree, this can result in cases where leaf nodes are not “pure” and no features are left to split.



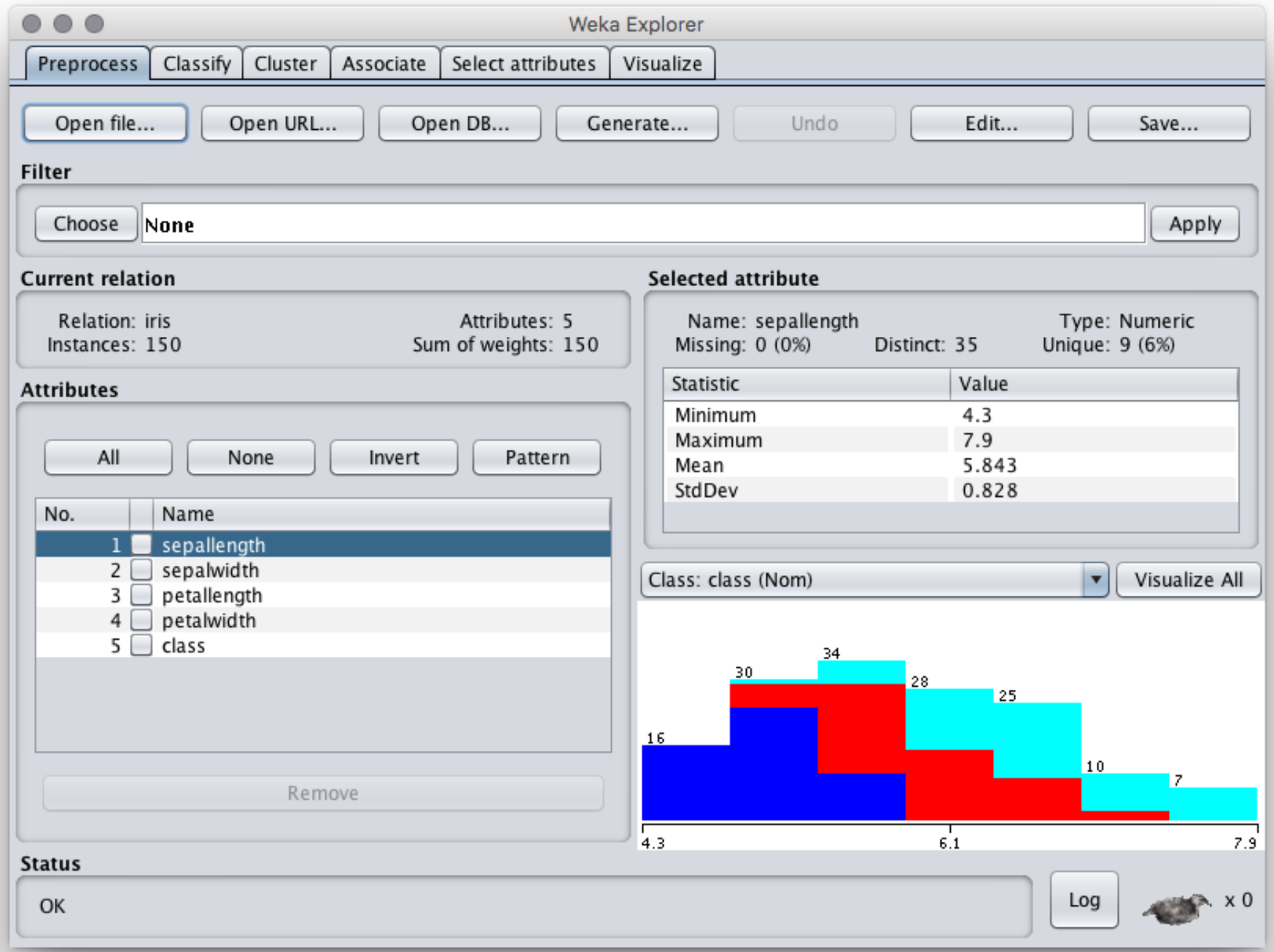

- Simple solution:
  - For the label, take the majority vote at the leaf node.
  - If there is a tie, randomly choose one of the class labels.

# C4.5 Algorithm

- C4.5 is an improved version of ID3 algorithm to over come some of its disadvantages (Quinlan, 1993).
- It contains several improvements to make it "an industrial strength" decision tree learner, including:
  - Handling continuous numeric features.
  - Handling training data with missing values.
  - Choosing an appropriate feature selection measure.
  - Providing an option for pruning trees after creation to reduce likelihood of overfitting.
- J48 is a widely-used open source implementation of C4.5, which is included in the Weka toolkit.

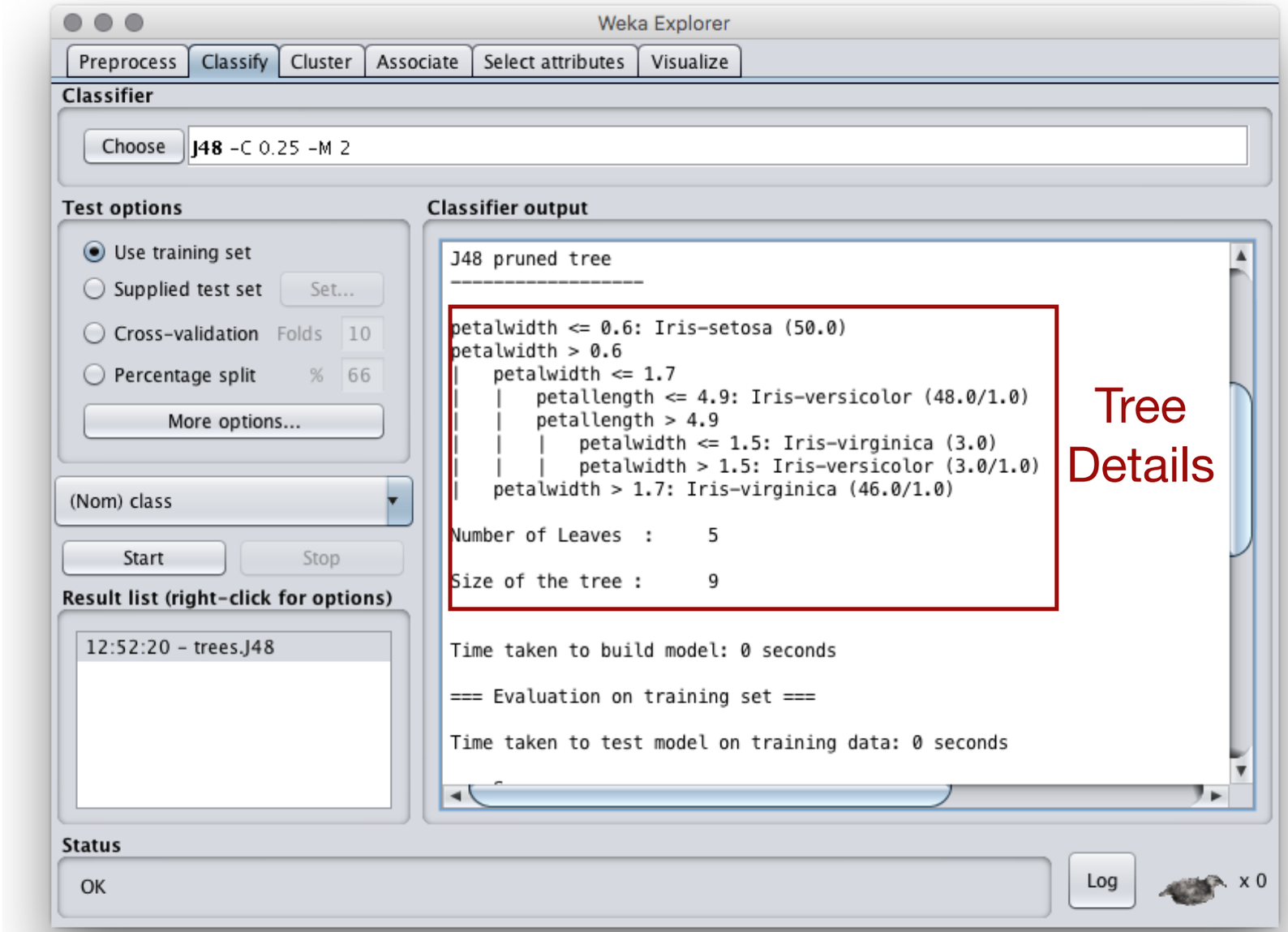
# J48 Decision Trees in Weka

1. Launch the WEKA application and click on the *Explorer* button.
2. *Open File* - iris.arff



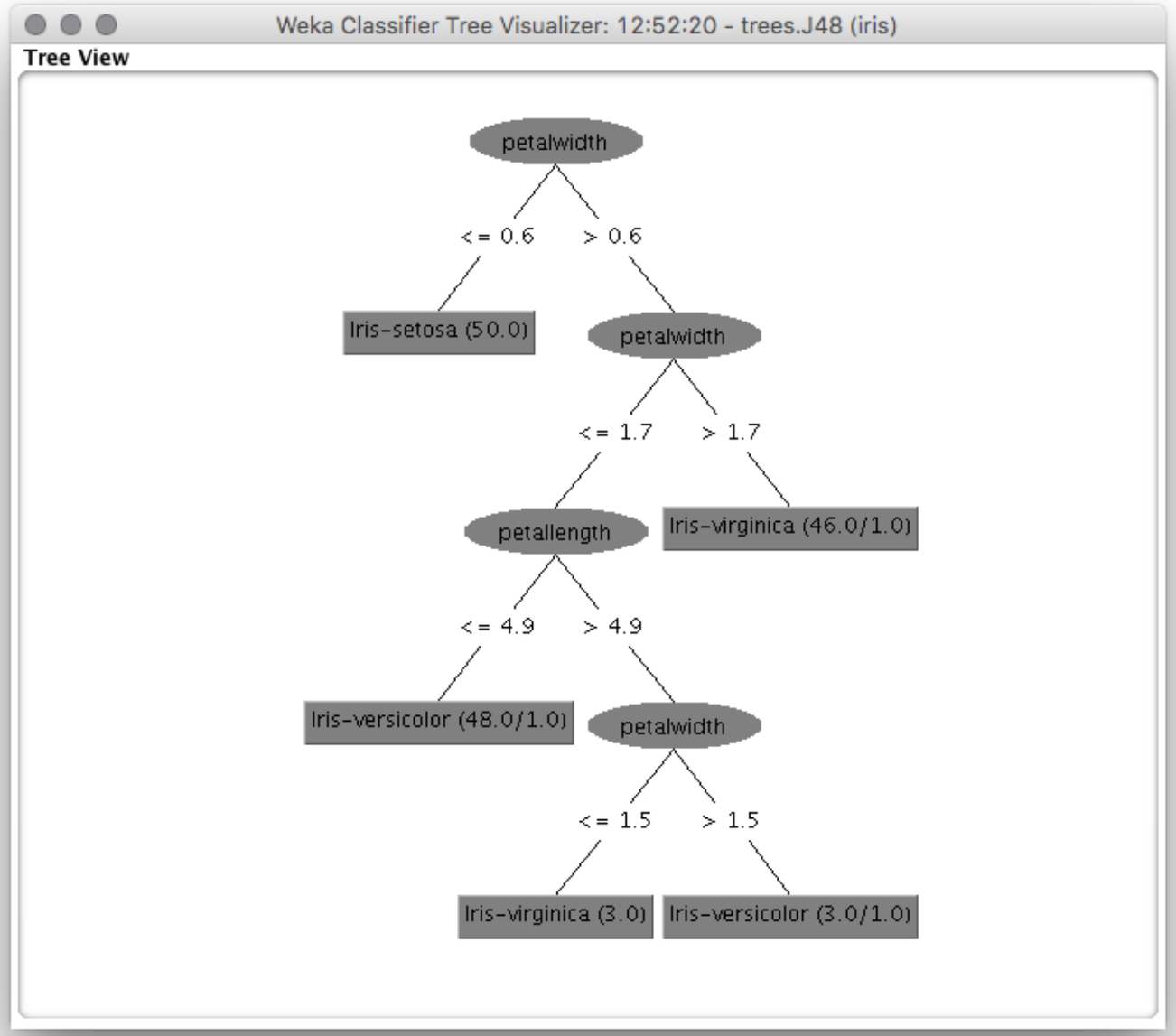
# J48 Decision Trees in Weka

3. In *Classify* tab, click *Choose* and find *Trees*→*J48* on the list.
4. Select *Test options: Use Training Set* (for this example).
5. Click *Start*.



# J48 Decision Trees in Weka

To view the decision tree, right click on the result in the result list and select *Visualize Tree*. A new window will display the pruned tree that was created by J48 on the training data.



# Summary

- A decision tree is an eager learning algorithm where the model is induced from the data in the form of decision rules.
- Many different trees can correctly model same training data.
- We want the simplest tree possible that can generalise to new unseen examples.
- Information Gain helps us select features to use when building simple decision trees.
- The ID3 algorithm for decision trees does not handle numeric data. The extended C4.5 algorithm can handle this type of feature.
- An implementation of the C4.5 algorithm (J48) is available in the Weka Explorer.

# References


- Russell & Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2009.
- Quinlan, J. R. 1986. “Induction of Decision Trees”. Machine Learning 1, 1 (Mar. 1986), 81-106.
- Mitchell, Tom M. Machine Learning. McGraw-Hill, 1997. pp. 55–58.
- Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.