Faculty of SET / School of Computer and Mathematical Sciences

COMP SCI 3007/7059/7659
Artificial Intelligence
Performance Assessment and Overfitting

Acknowledgement of Country

We acknowledge and pay our respects to the Kaurna people, the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to the country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Performance Assessment and Overfitting

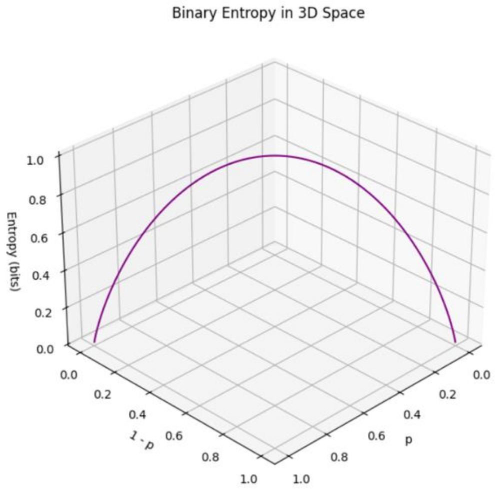
AIMA C18.3.5

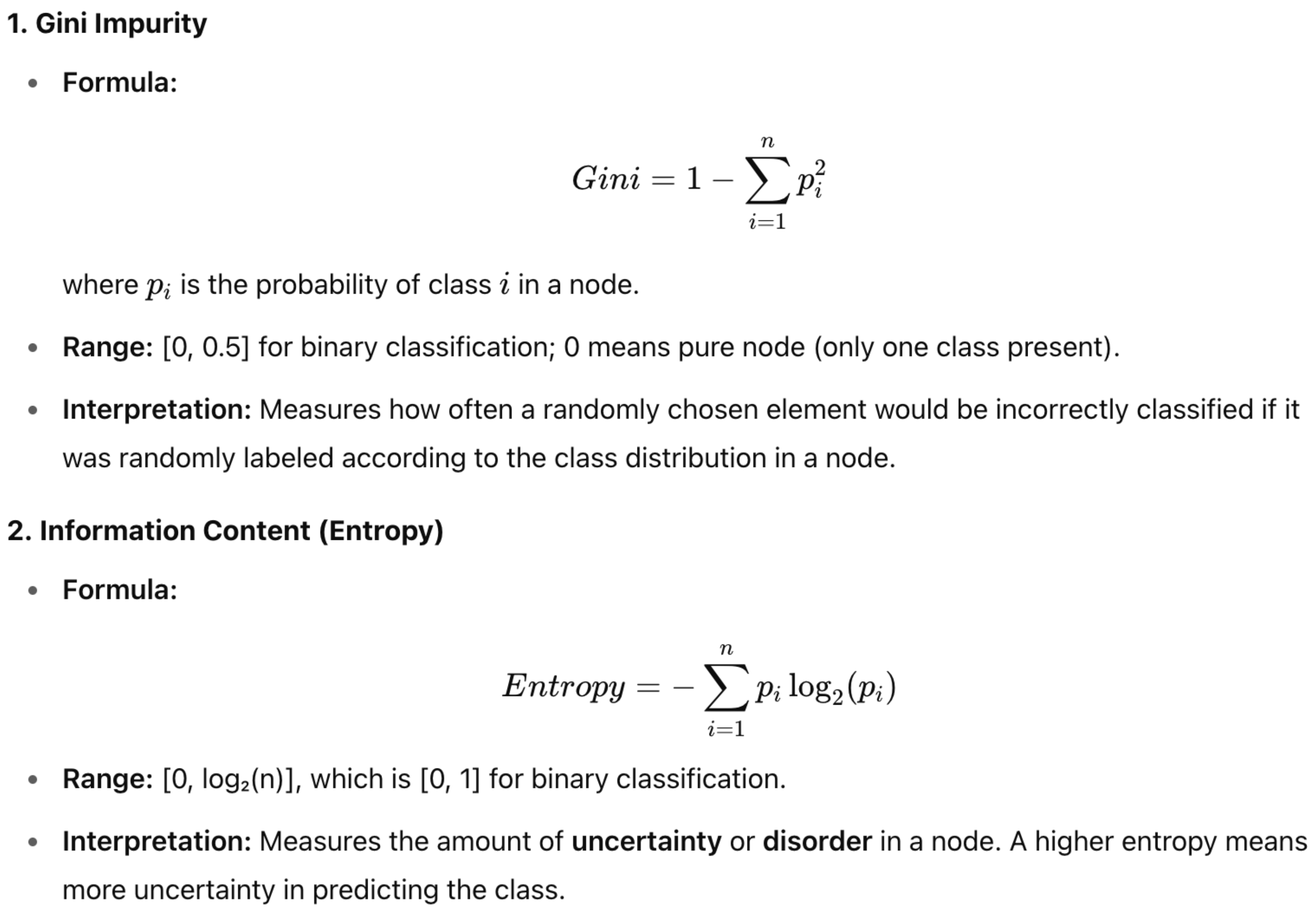
Impurity

- A different point of view is to regard ***I(node)*** as a measure of **impurity**. The more “mixed” the samples are at a node (i.e. has equal proportions of all class labels), the higher the impurity value. On the other hand, a homogeneous node (i.e. has samples of one class only) will have zero impurity.

- The Gain(A) value can thus be viewed as the amount of reduction in impurity if we split according to A.

- This affords the intuitive idea that we grow trees by **recursively trying to obtain leaf nodes which are as pure as possible.**





1. Gini Impurity

- **Formula:**

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

  where p_i is the probability of class i in a node.

- **Range:** [0, 0.5] for binary classification; 0 means pure node (only one class present).
- **Interpretation:** Measures how often a randomly chosen element would be incorrectly classified if it was randomly labeled according to the class distribution in a node.

2. Information Content (Entropy)

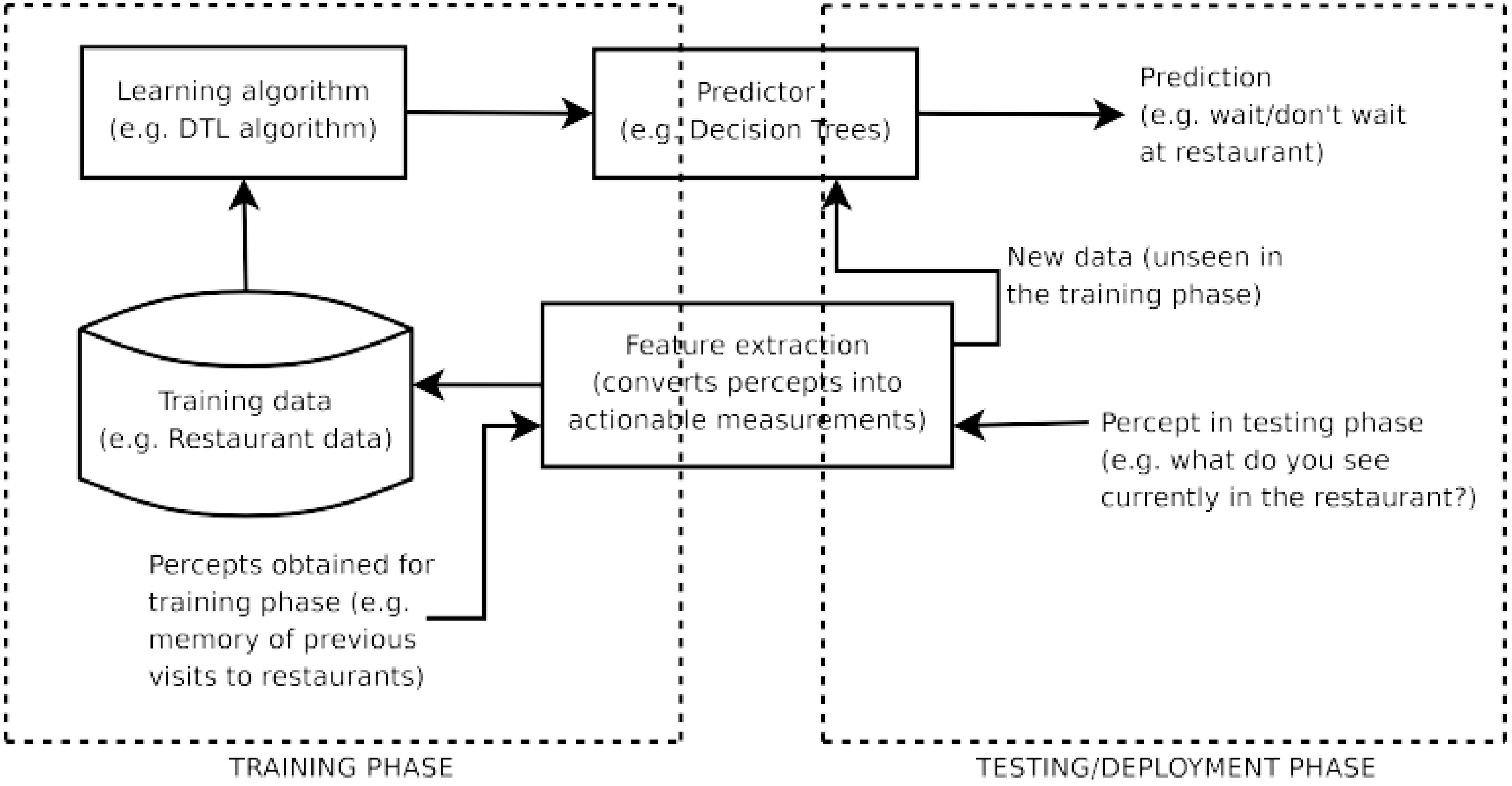
- **Formula:**

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

- **Range:** [0, $\log_2$(n)], which is [0, 1] for binary classification.
- **Interpretation:** Measures the amount of **uncertainty** or **disorder** in a node. A higher entropy means more uncertainty in predicting the class.

Basic pipeline of learning algorithms

There are 2 major stages in a learning algorithm:

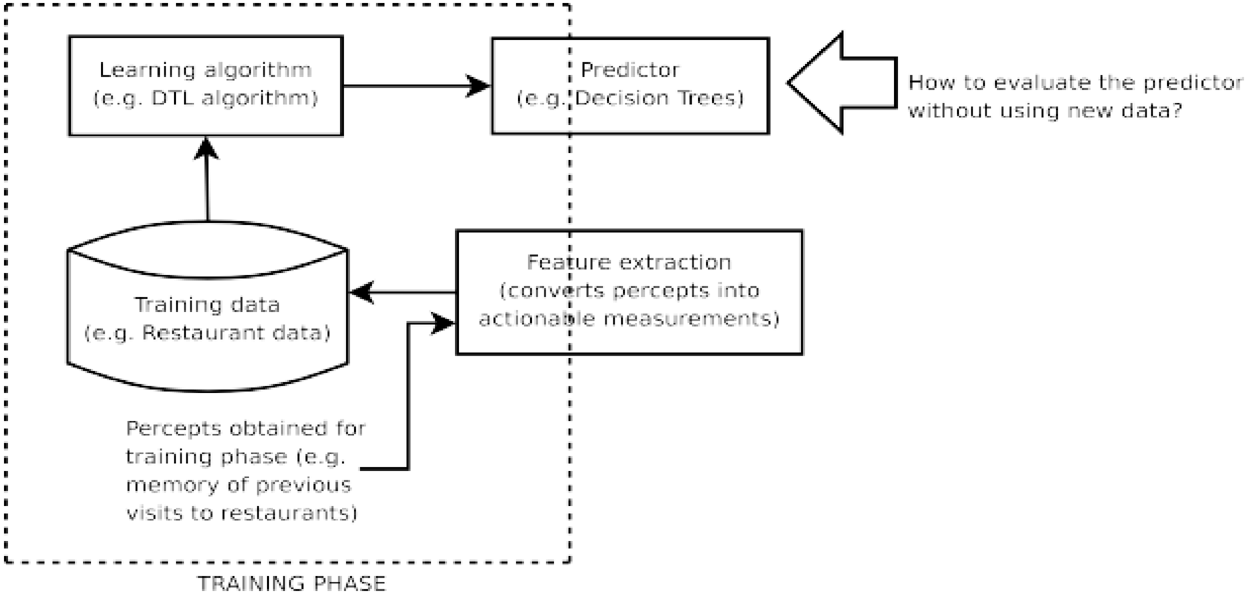


Note that the training and testing data are obtained at **different stages in time**.

A good learning system makes accurate predictions...

In most cases we would like to evaluate the predictor by **how well it predicts** the label for **new samples/data**, i.e., its performance in the **testing phase**. This implies, however, that we would have to **deploy** the system first before evaluating it — a very risky strategy!

Ideally, we want to evaluate before deployment, but how do we test prediction capability on **unseen** data before deployment?



Methodology for Assessing Performance

What we can do is as follows:

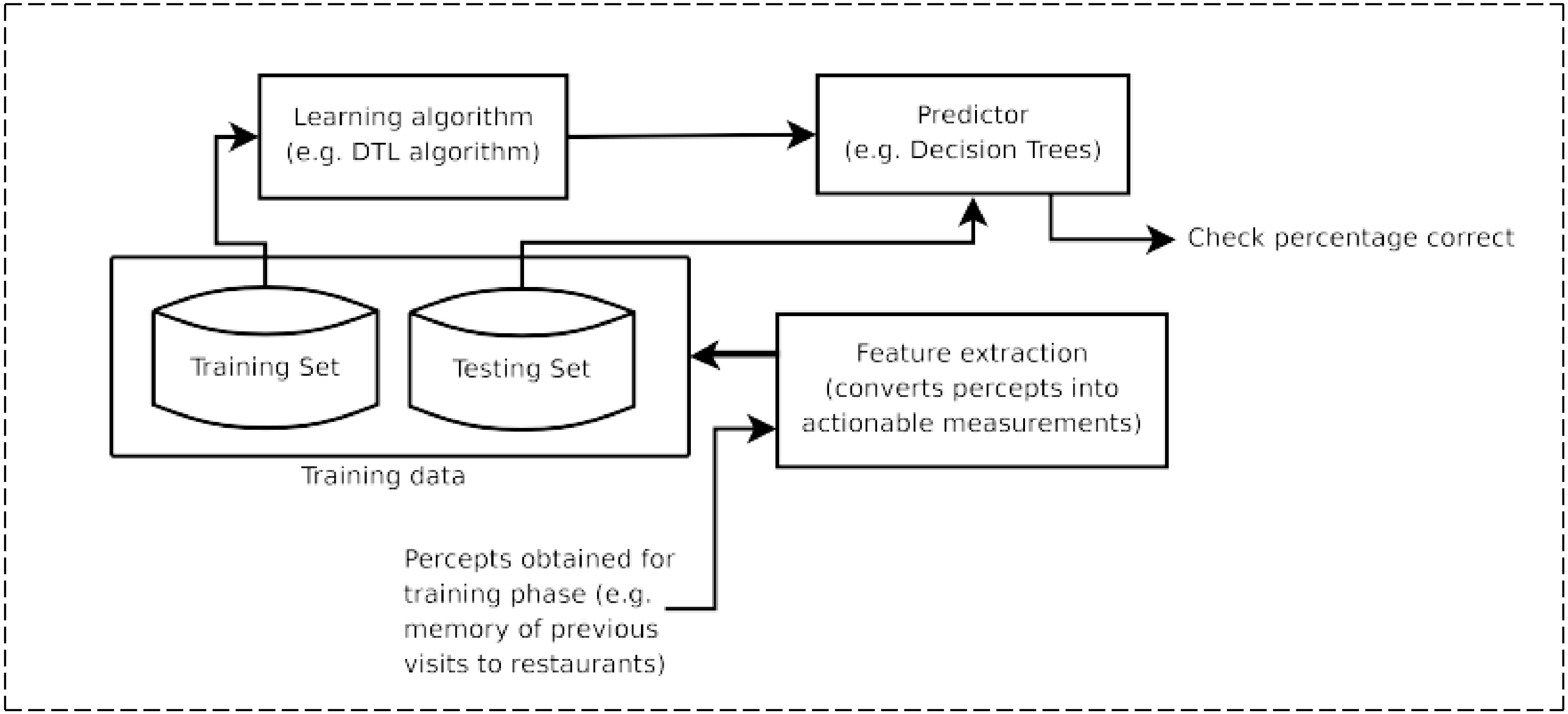
1. Collect a large set of training data, i.e. measurements and their desired labels.

2. **Randomly** divide it into two **disjoint (non-overlapping)** subsets: the **training set** and the **testing set**.

3. Apply the learning algorithm (e.g. decision trees) on the training set, producing a predictor/classifier.

4. Measure the percentage of samples in the testing set that are correctly labelled by the predictor/classifier.

Methodology for Assessing Performance (cont.)



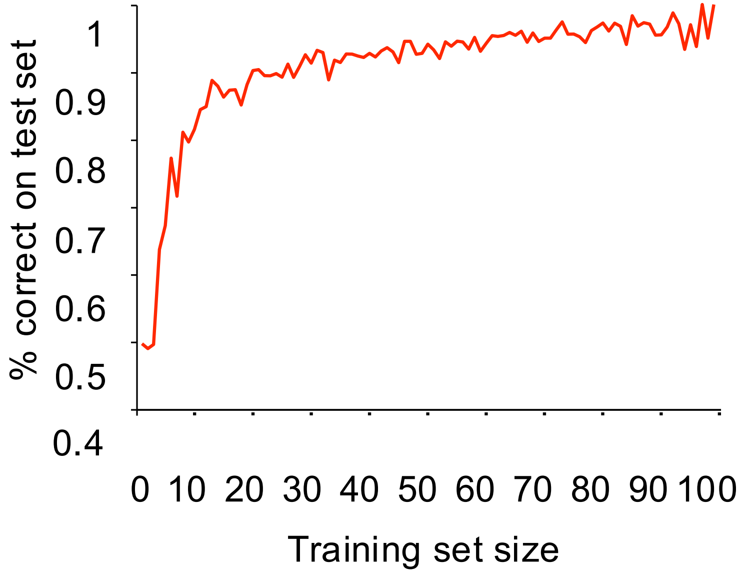
TRAINING PHASE

Training phase			Deployment phase
Training set	Testing/Validation set		Deployment set
Training	Validation	test	

Methodology for Assessing Performance (cont.)

The results can be plotted on a graph called the **learning curve**, i.e., a plot of the accuracy (% of correct prediction) versus size of training set.

Example: Applying the DTL algorithm on the "restaurant" problem with 100 randomly generated samples:



Notice that the correct prediction rate increases with the size of the training set, a sign that the learning algorithm is picking up the pattern in the data.

Peeking or Cheating

Peeking or Cheating: Allowing the learning algorithm to access or "see" the testing set before performance evaluation.

This is usually (inadvertently) done in 2 ways:

1. Allowing the training and testing sets to overlap. An extreme case is to report performance based on prediction results on the training set itself, i.e. training set = testing set!
2. A more subtle way is to tweak the learning algorithm based on its performance on the testing set.

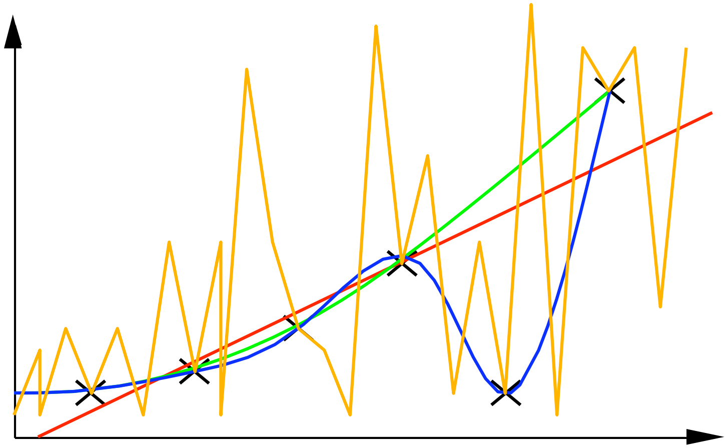
Always avoid peeking or cheating by ensuring absolute separation between the training and testing sets.

Overfitting

Overfitting: Learning meaningless regularity or patterns in the data.

Overfitting is detrimental to the **generalisation capability** (i.e. the prediction performance) of a predictor/classifier.

Recall the 1D regression problem: the line which captures best the pattern of the data will have the highest generalisation capability.

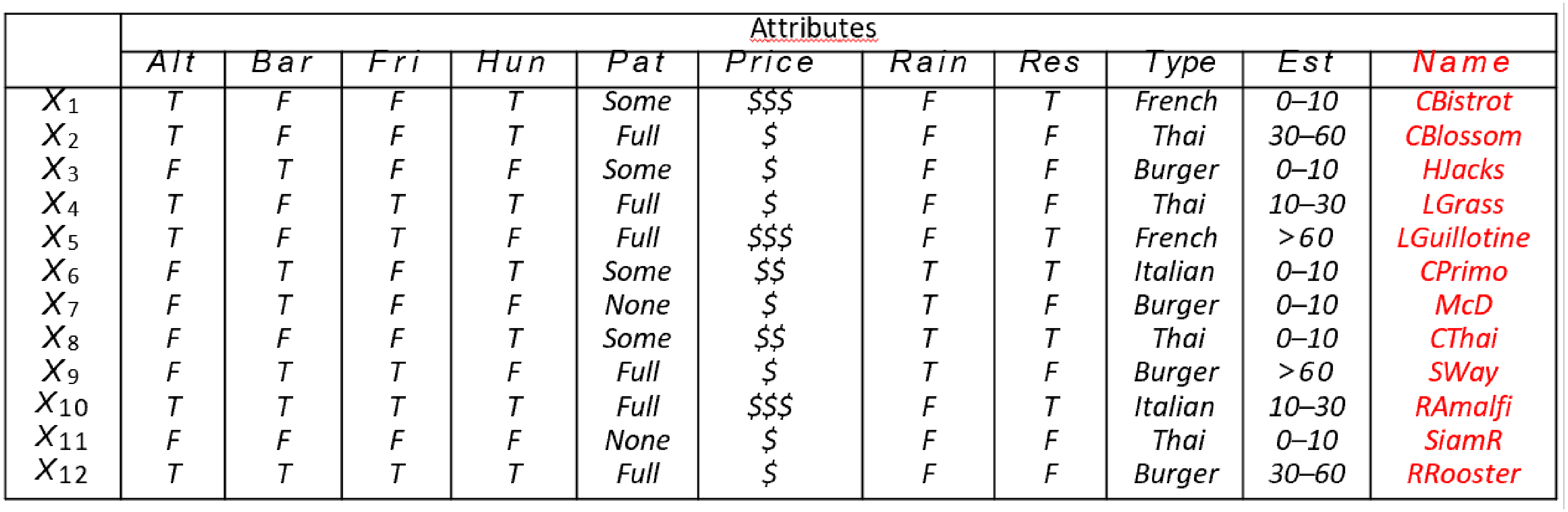


Recall Occam's Razor which states that we should prefer the simplest hypothesis that explains the data.

Example: Overfitting in the restaurant problem

Data can have irrelevant attributes/features.

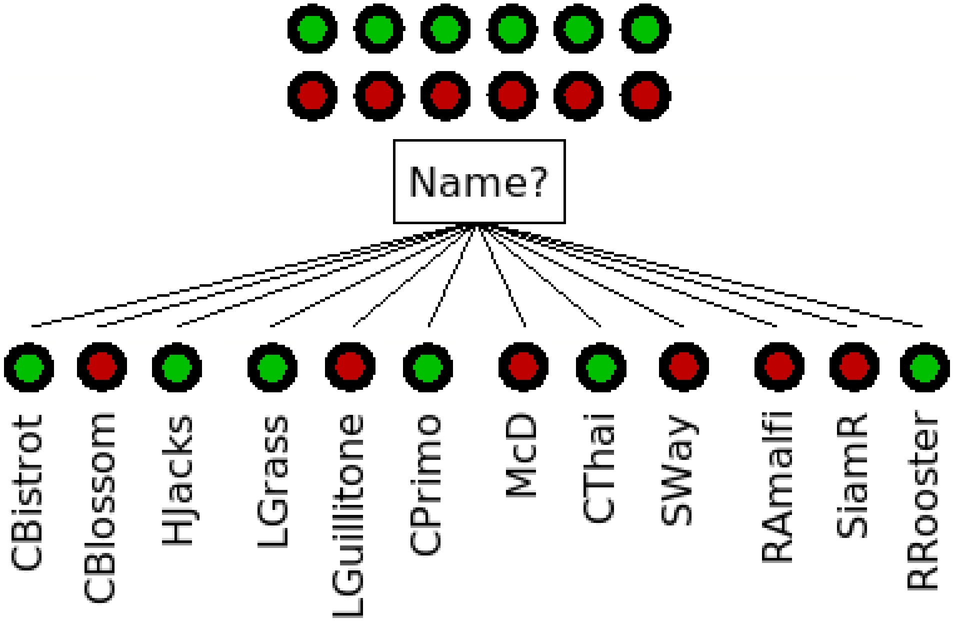
Let's add a new attribute (name of restaurant) to the restaurant data:



	Attributes										
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Name
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	CBistrot
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	CBlossom
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	HJacks
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	LGrass
X_5	T	F	T	F	Full	$$$	F	T	French	>60	LGuillotine
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	CPrimo
X_7	F	T	F	F	None	$	T	F	Burger	0–10	McD
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	CThai
X_9	F	T	T	F	Full	$	T	F	Burger	>60	SWay
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	RAmalfi
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	SiamR
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	RRooster

Example: Overfitting in the restaurant problem

As long as no two samples have the same value for some attribute, the DTL algorithm will **always** find a **consistent** hypothesis!

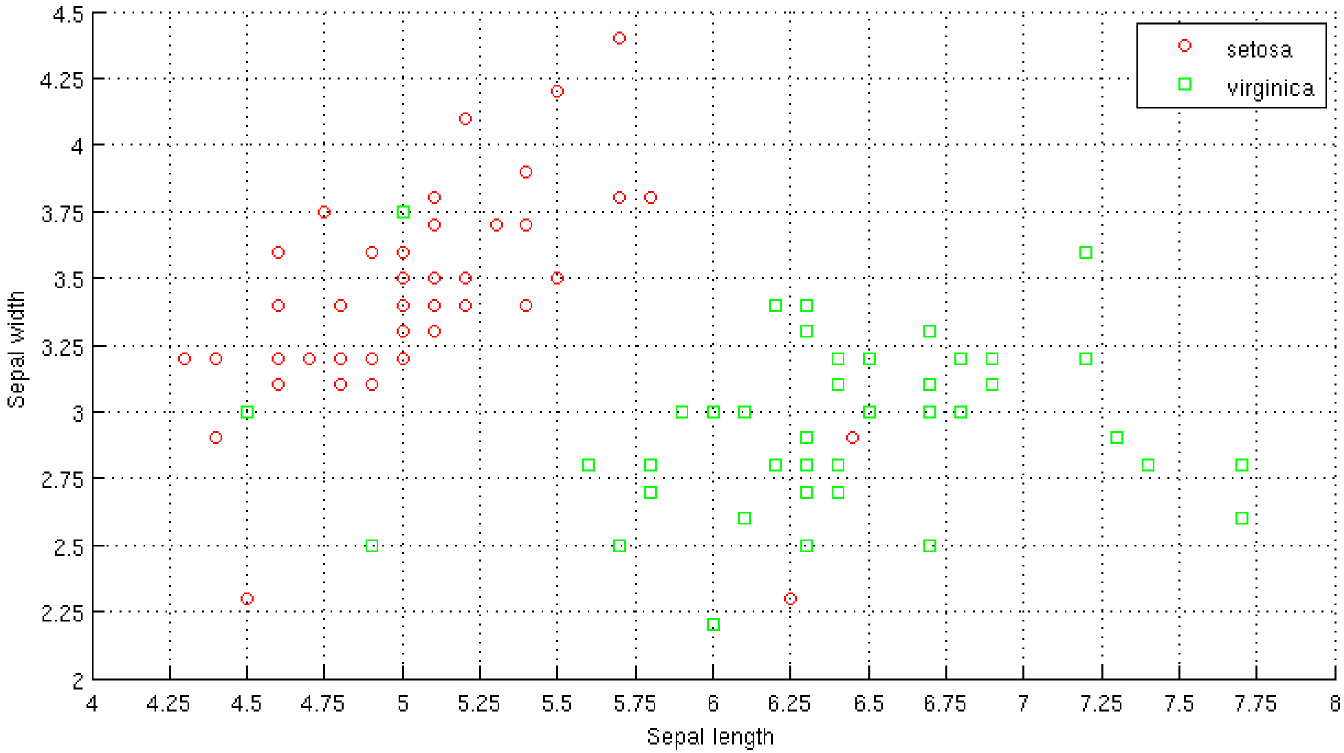


- A predictor or classifier that exhibits overfitting is unlikely to be useful.

DTL on Iris data

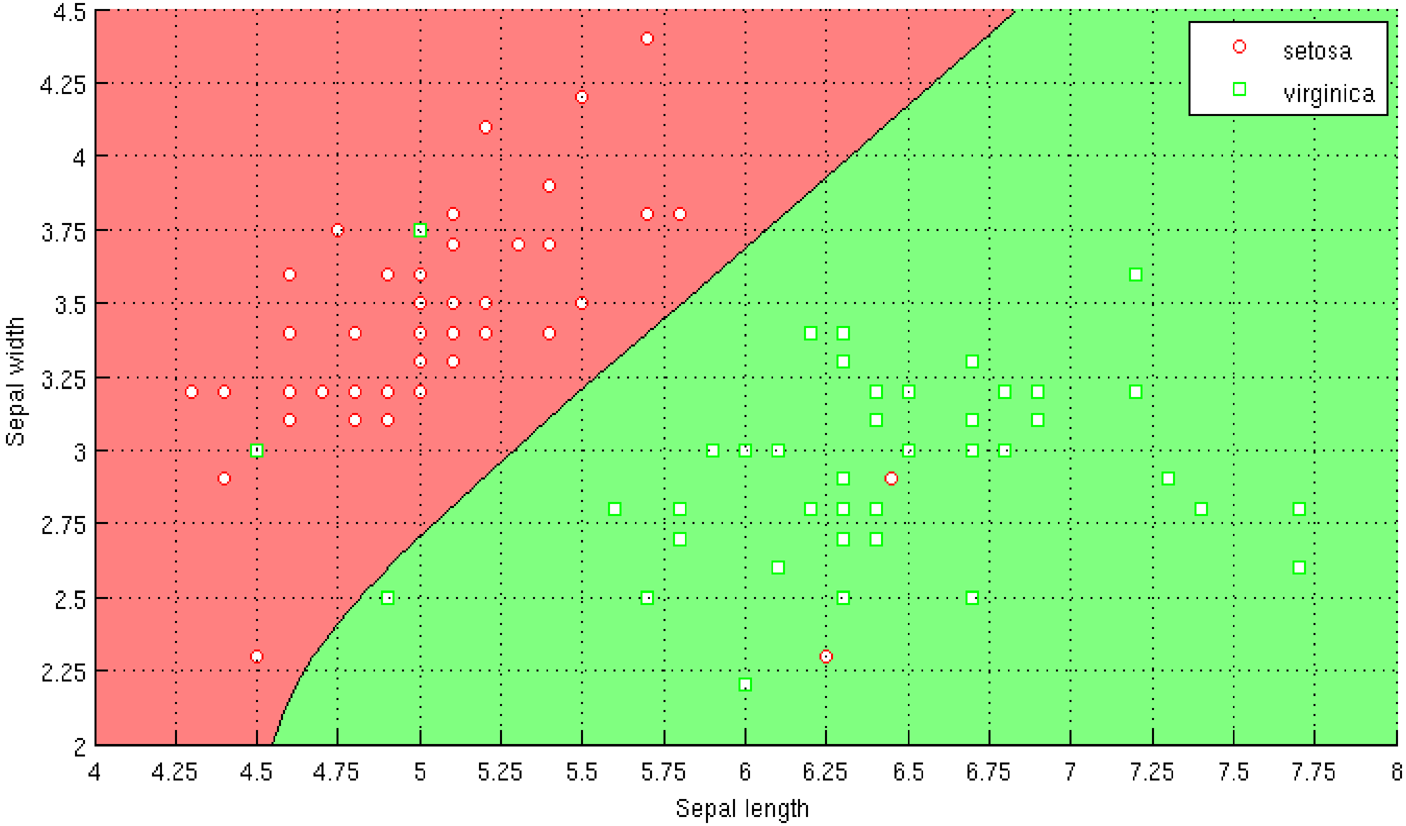
Trying to **predict** one of two types of irises based on measurements of the sepal length and width.

Each sample $\mathbf{x}_i$ contains two values (sepal length, sepal width) and a class label y_i (*setosa* or *virginica*). The training set can be plotted in 2D:



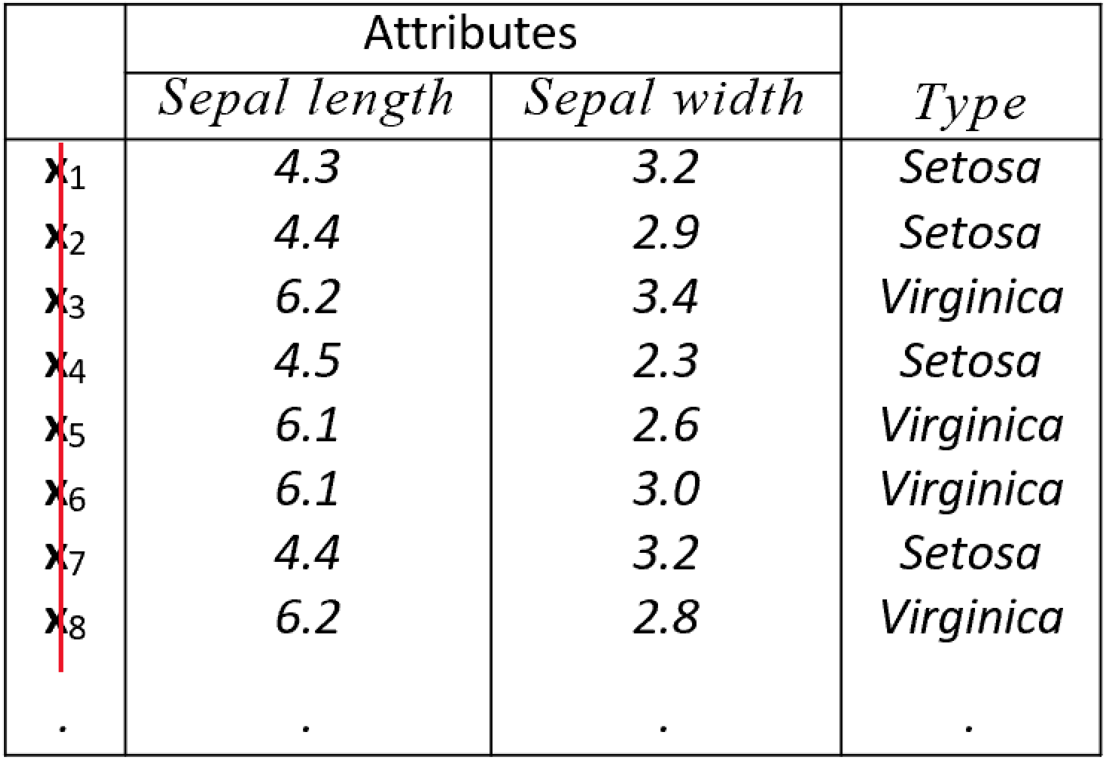
DTL on Iris data(cont.)

Intuitively the data is composed of two distinct areas with a smooth curve separating the two types of irises:



DTL on Iris data(cont.)

In contrast to the Restaurant data, the Iris data attributes is **continuous**, i.e. the value of each attribute (sepal length, sepal width) can vary continuously within a range. In tabular form:



		Attributes		Type
		Sepal length	*Sepal width*	
d_1	x_1	4.3	3.2	Setosa
d_2	x_2	4.4	2.9	Setosa
	x_3	6.2	3.4	Virginica
…	x_4	4.5	2.3	Setosa
	x_5	6.1	2.6	Virginica
	x_6	6.1	3.0	Virginica
	x_7	4.4	3.2	Setosa
d_8	x_8	6.2	2.8	Virginica
	.	.	.	.
		x_1	x_2	

Intuitively, each attribute corresponds to a **dimension** of the continuous valued data.

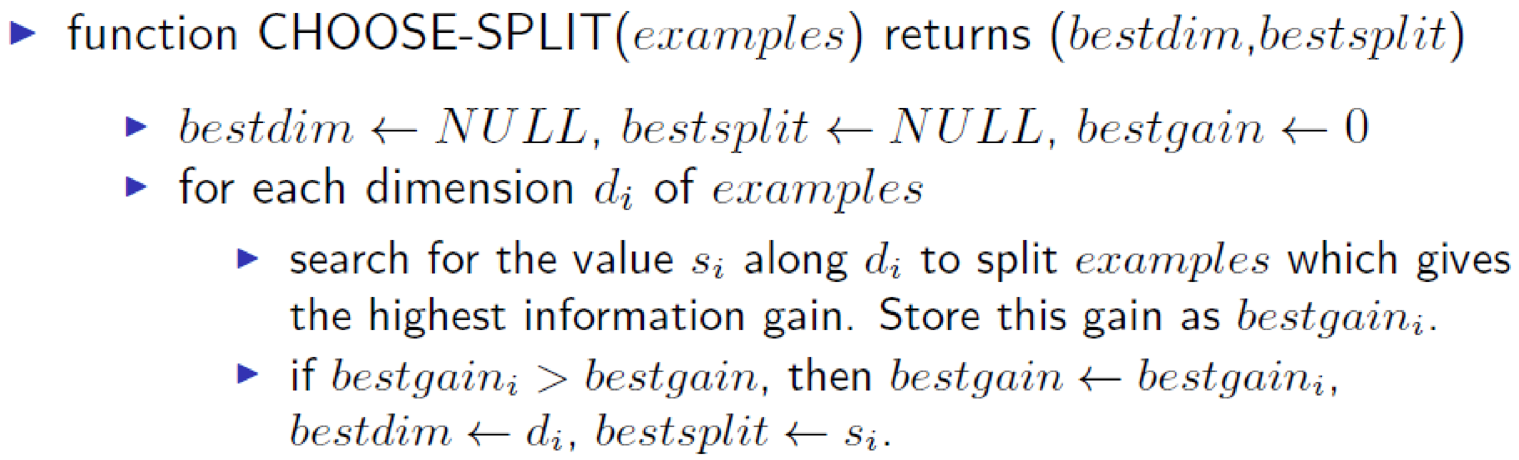
DTL for continuous data

In contrast to DTL for discrete data, in DTL for continuous data, we **never run-out of attributes** since we can always find a value to split along a particular dimension:

```
function DTL($examples$,$default$) returns a decision tree $dtree$
    if $examples$ is empty then return $default$
    else if all $examples$ have the same labels then return a single-node tree containing $examples$.
    else
        $(bestdim, bestsplit) \leftarrow$ CHOOSE-SPLIT($examples$)
        $dtree \leftarrow$ a new decision tree with root test $(bestdim, bestsplit)$
                                                   or equal to
        $left \leftarrow$ elements of $examples$ with value less than $bestsplit$ at dimension $bestdim$
        $leftchild \leftarrow$ DTL($left$,MODE($left$))
        Attach $leftchild$ as a subtree to $dtree$
        $right \leftarrow$ elements of $examples$ with value more than $bestsplit$ at dimension $bestdim$
        $rightchild \leftarrow$ DTL($right$,MODE($right$))
        Attach $rightchild$ as a subtree to $dtree$
```

DTL for continuous data(cont.)

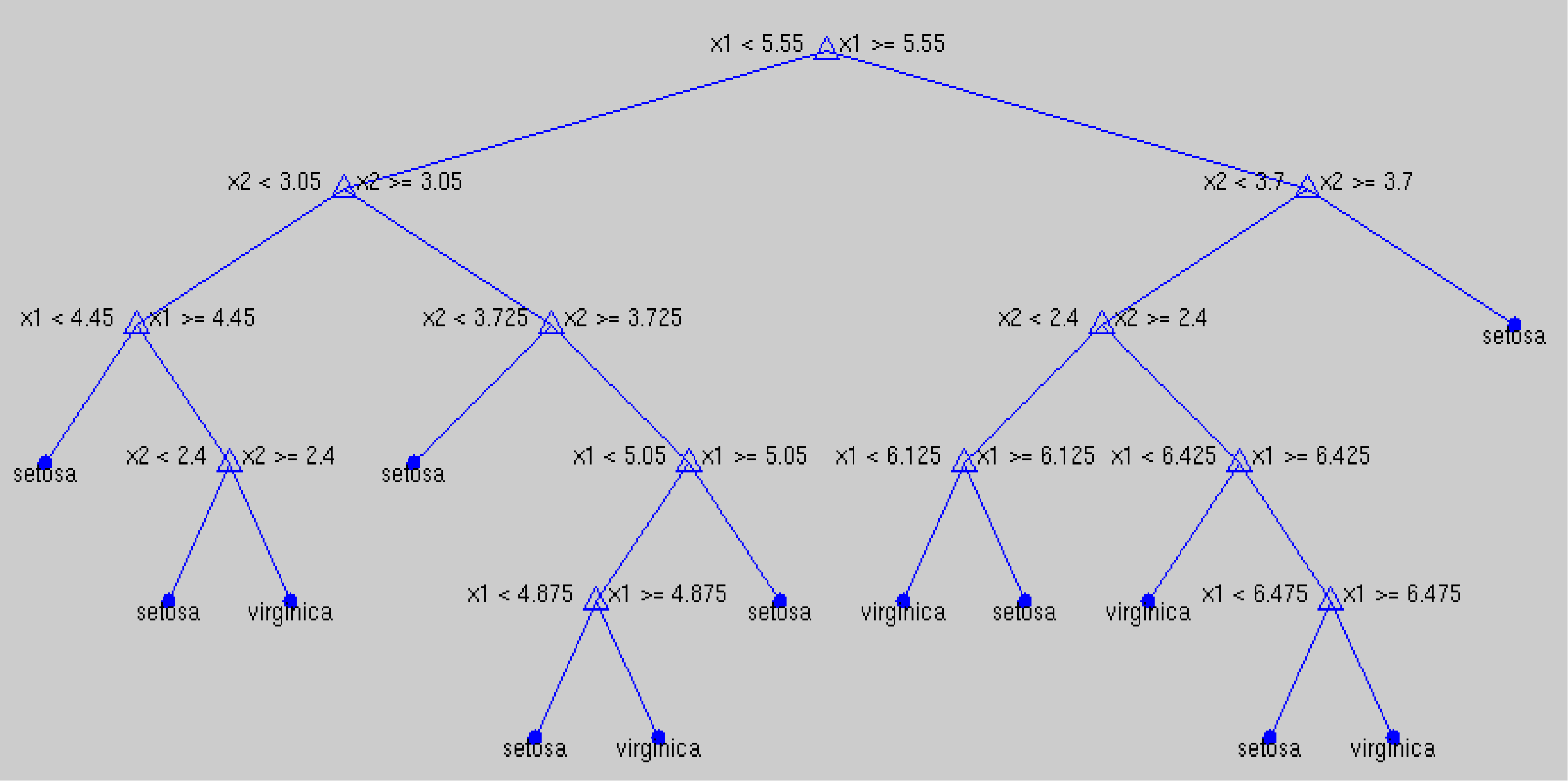
The subroutine CHOOSE-SPLIT basically finds the best dimension (i.e., attribute) and value along which to split the examples according to the principle of maximising information gain:



```
▸ function CHOOSE-SPLIT($examples$) returns ($bestdim$,$bestsplit$)
    ▸ $bestdim \leftarrow NULL$, $bestsplit \leftarrow NULL$, $bestgain \leftarrow 0$
    ▸ for each dimension $d_i$ of $examples$
        ▸ search for the value $s_i$ along $d_i$ to split $examples$ which gives
          the highest information gain. Store this gain as $bestgain_i$.
        ▸ if $bestgain_i > bestgain$, then $bestgain \leftarrow bestgain_i$,
          $bestdim \leftarrow d_i$, $bestsplit \leftarrow s_i$.
```

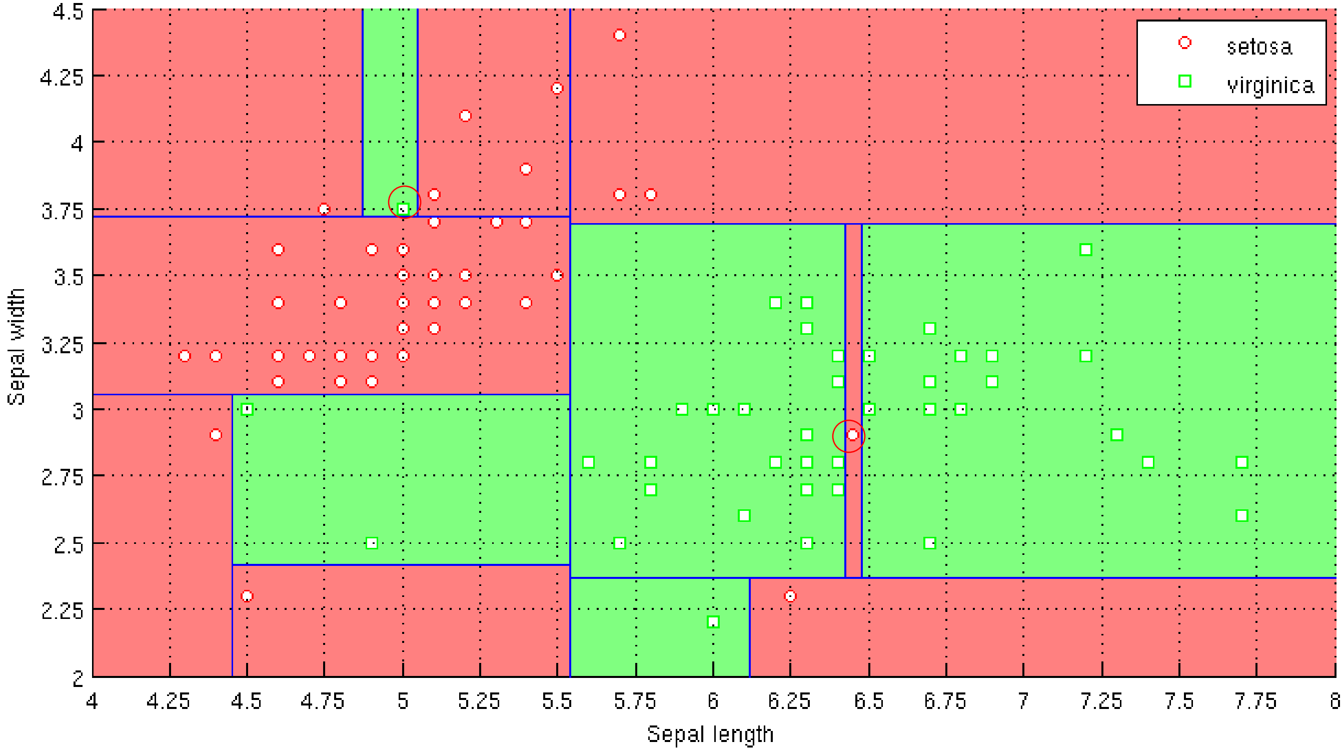
DTL for continuous data(cont.)

Running DTL on the Iris data produces the following Decision Tree:



Overfitting the Iris data

The **predictions** of the Decision Tree can be displayed as **decision areas** in the space of the data:



It can be seen that DTL is able to learn a Decision Tree that fits the data perfectly. However due to overfitting the Decision Tree will likely produce errors in the testing phase!

Decision Tree Pruning

A technique to prevent overfitting in DTL.

Two general strategies:

1. **Post-pruning**: This requires a separate testing set (i.e. validation).

First grow the tree fully and remove leaf nodes one-by-one if the prediction accuracy of the tree on the testing set improves. Stop as soon as the prediction accuracy starts to deteriorate.

2**. Pre-pruning**: Try to limit the growth of the tree by stopping prematurely. The following criteria can be used to decide when to stop splitting at a particular node:

- Stop splitting as soon as the **number of data remaining** at the node is less than a pre-determined number.
- Or, stop splitting as soon as the **information content** at the node is less than a pre-determined minimum value.

Pre-Pruning: Minimum number of data per node

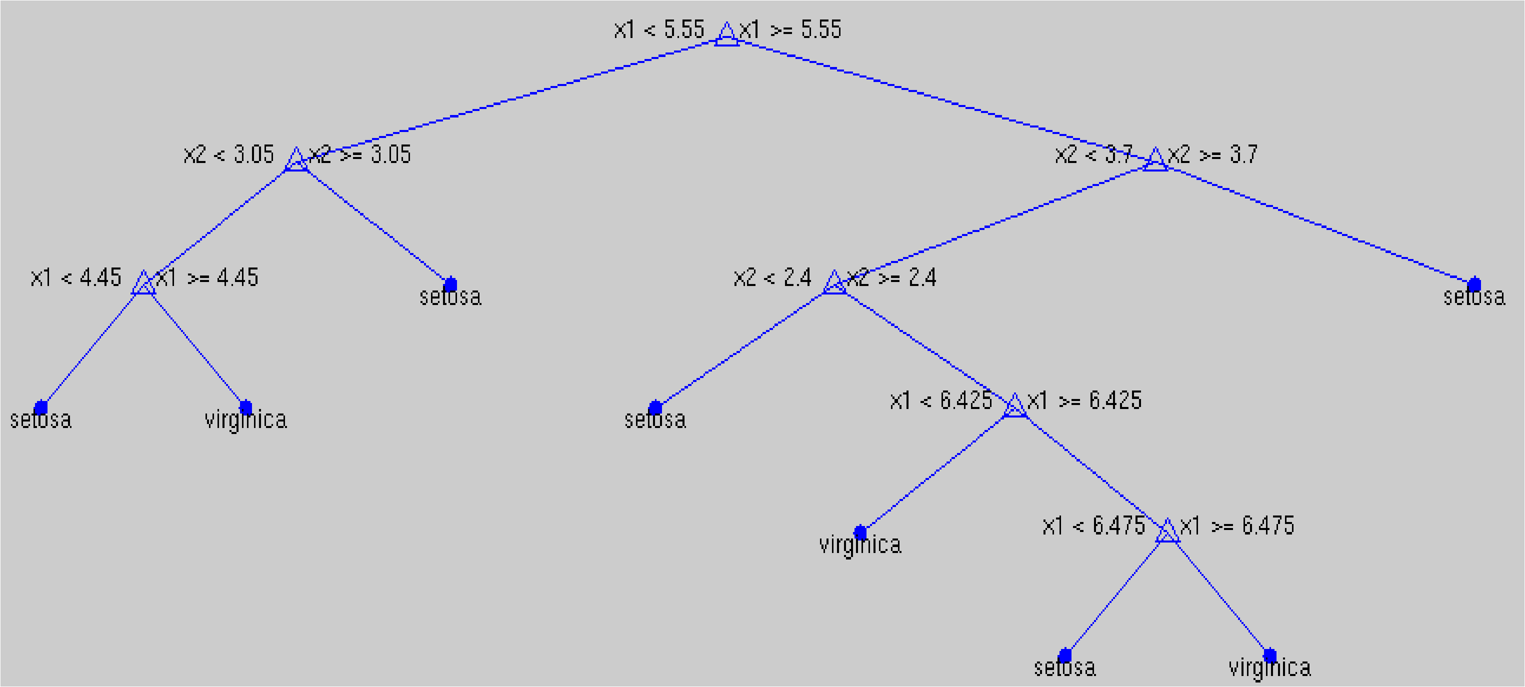
Essentially, we are just **changing the stopping criterion** of DTL such that it doesn't try to perfectly split the data:

```
function DTL($examples$,$default$,$minsplit$) returns a decision tree $dtree$
    if $examples$ is empty then return $default$
    else if $size(examples) < minsplit$ then return a single-node tree containing $examples$.
    else
        ($bestdim$, $bestsplit$) ← CHOOSE-SPLIT($examples$)
        $dtree$ ← a new decision tree with root test ($bestdim$, $bestsplit$)
                                                            or equal to
        $left$ ← elements of $examples$ with value less than $bestsplit$ at dimension $bestdim$
        $leftchild$ ← DTL($left$,MODE($left$),$minsplit$)
        Attach $leftchild$ as a subtree to $dtree$
        $right$ ← elements of $examples$ with value more than $bestsplit$ at dimension $bestdim$
        $rightchild$ ← DTL($right$,MODE($right$),$minsplit$)
        Attach $rightchild$ as a subtree to $dtree$
```

```
    else if all $examples$ have the same labels then return a single-node tree containing $examples$.
```

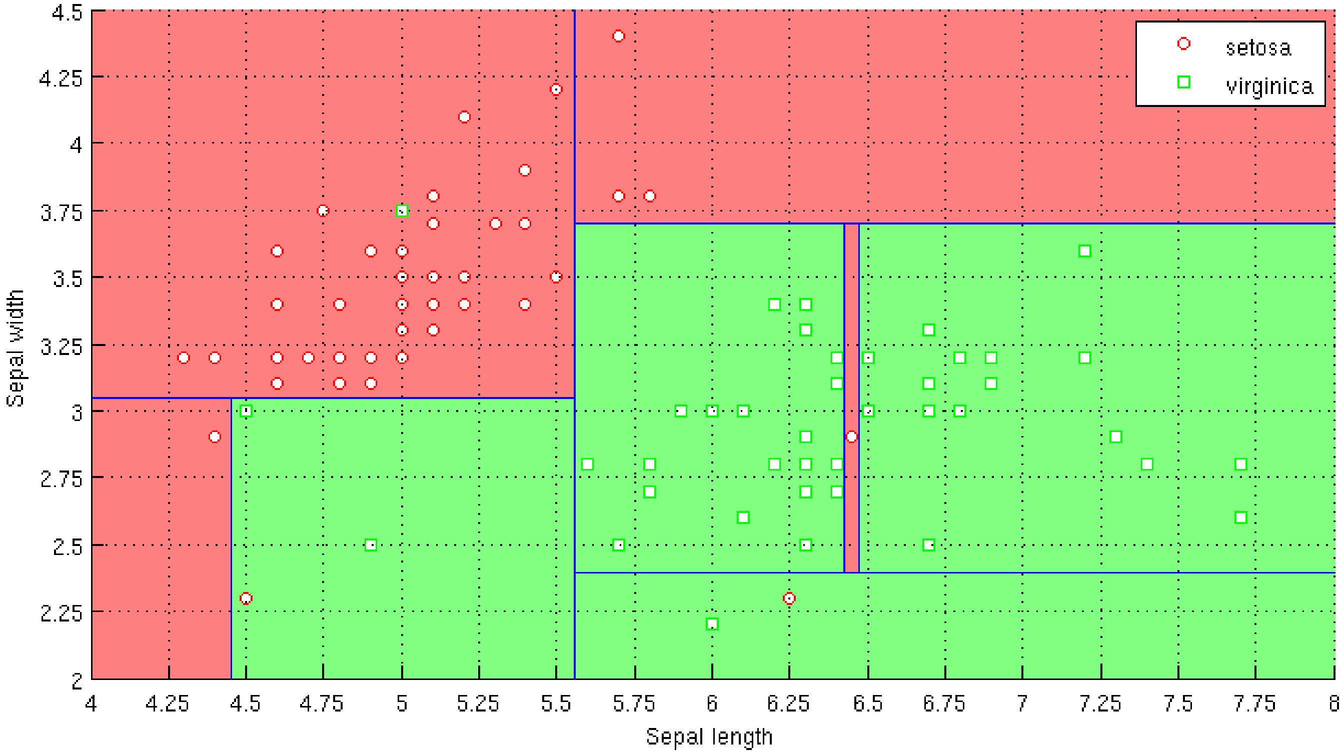
Pre-Pruning: Minimum number of data pernode(cont.)

For example, in the Iris data we impose $minsplit = 4$. The corresponding Decision Tree looks like:



Pre-Pruning: Minimum number of data per node(cont.)

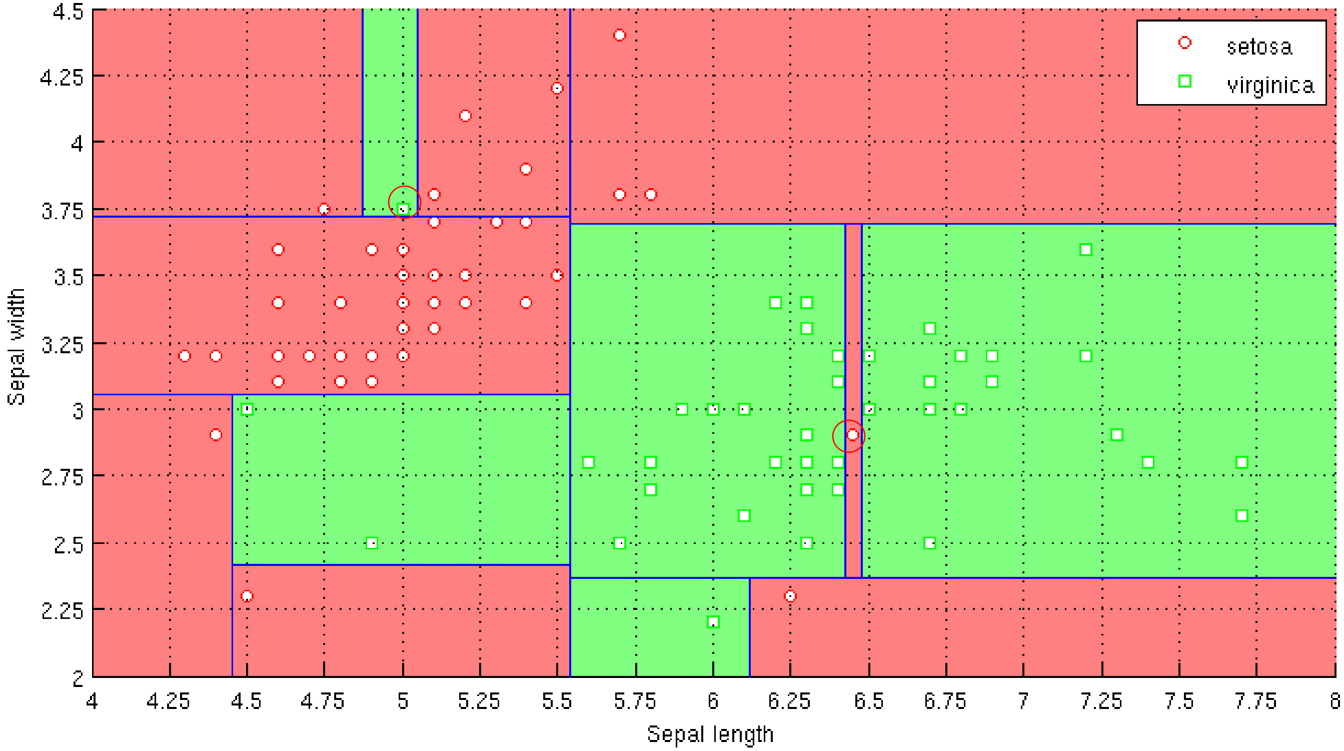
The decision areas now look like:



Notice that some training examples are classified incorrectly by the pruned tree. However this Decision Tree would **generalise** better to new unseen data.

Overfitting the Iris data

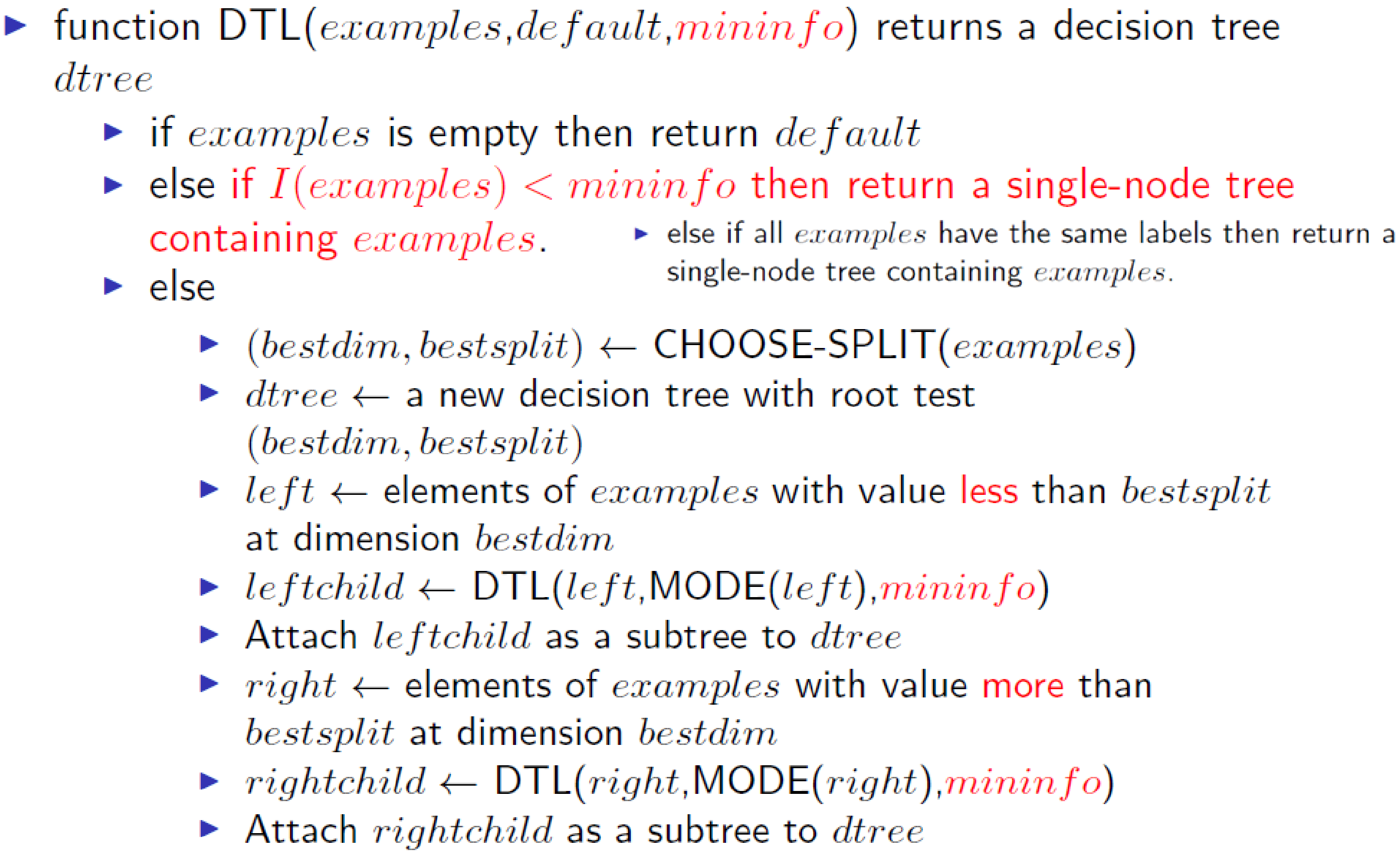
The **predictions** of the Decision Tree can be displayed as **decision areas** in the space of the data:



It can be seen that DTL is able to learn a Decision Tree that fits the data perfectly. However due to overfitting the Decision Tree will likely produce errors in the testing phase!

Pre-Pruning: Minimum information content per node

Another way is to impose a minimum information content at each node before allowing splitting:

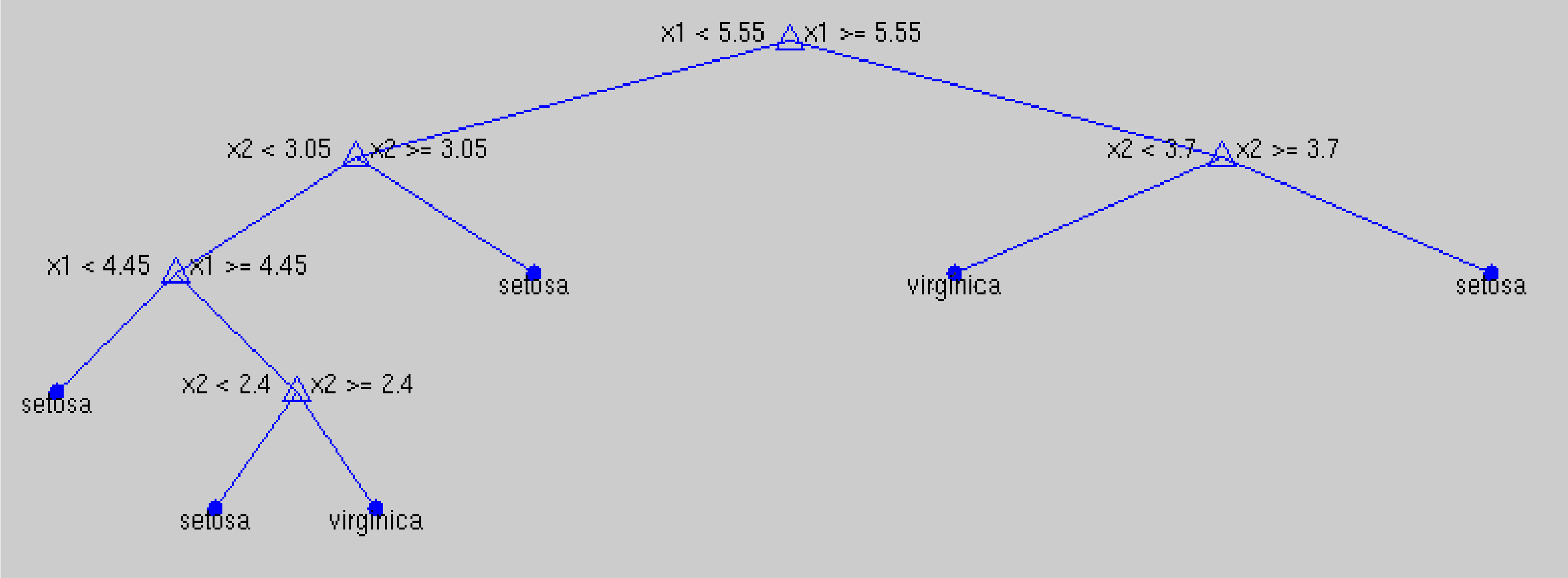


```
function DTL($examples$,$default$,$mininfo$) returns a decision tree $dtree$
    if $examples$ is empty then return $default$
    else if $I(examples) < mininfo$ then return a single-node tree containing $examples$.
    else
        $(bestdim, bestsplit) \leftarrow$ CHOOSE-SPLIT($examples$)
        $dtree \leftarrow$ a new decision tree with root test $(bestdim, bestsplit)$
        $left \leftarrow$ elements of $examples$ with value less than $bestsplit$ at dimension $bestdim$
        $leftchild \leftarrow$ DTL($left$,MODE($left$),$mininfo$)
        Attach $leftchild$ as a subtree to $dtree$
        $right \leftarrow$ elements of $examples$ with value more than $bestsplit$ at dimension $bestdim$
        $rightchild \leftarrow$ DTL($right$,MODE($right$),$mininfo$)
        Attach $rightchild$ as a subtree to $dtree$
```

- else if all $examples$ have the same labels then return a single-node tree containing $examples$.

Pre-Pruning: Minimum information content per node (cont.)

For example, in the Iris data, we impose the minimum information content of 0.3 bits. The corresponding Decision Tree looks like:



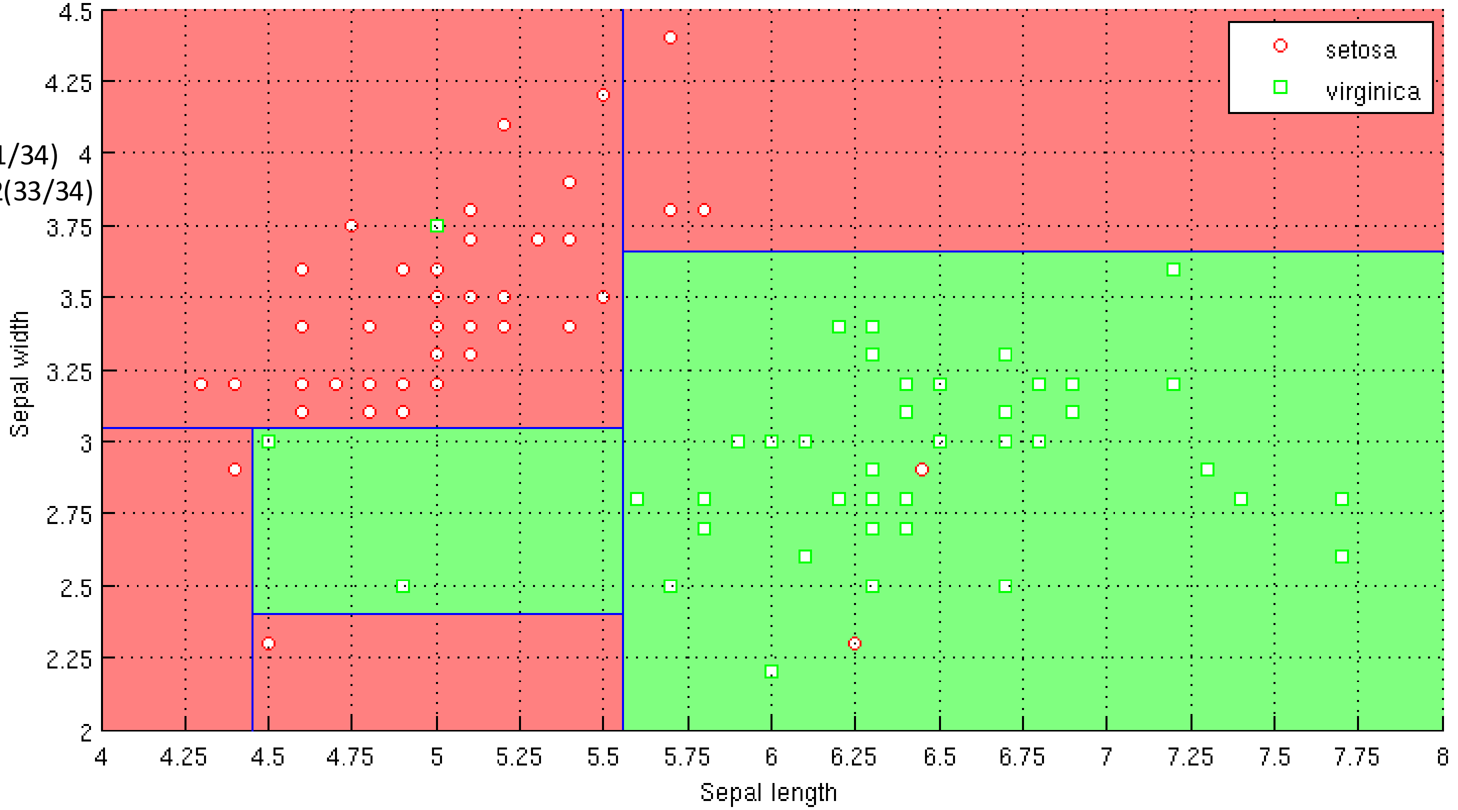
- You can verify that we stopped splitting as soon as the information content is less than 0.3 bits.

Pre-Pruning: Minimum information content per node (cont.)

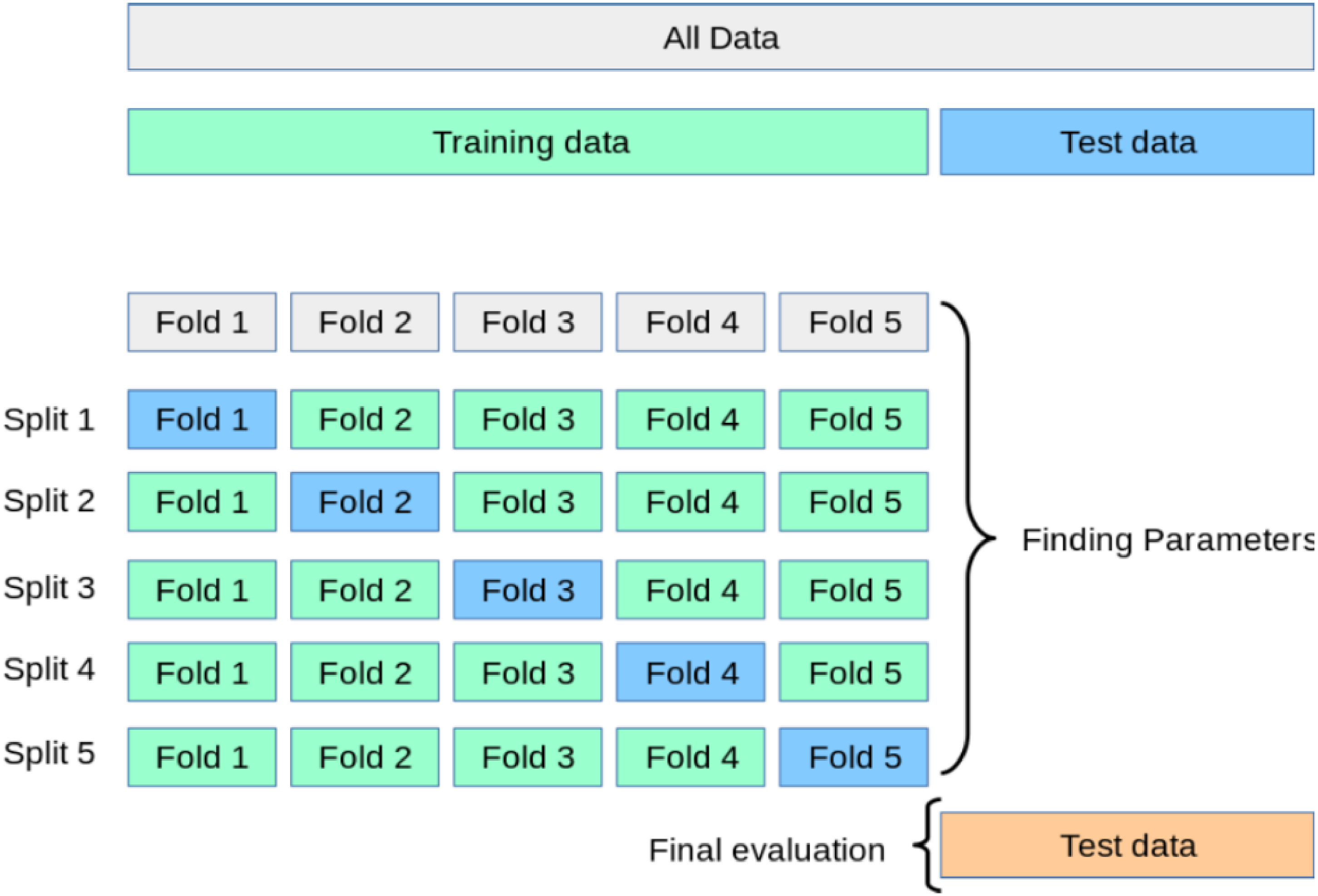
The decision areas now look like:

33 vs 1 – total 34

Entropy = -1/34*np.log2(1/34)
-33/34*np.log2(33/34)
=0.19



K-fold Cross-Validation



All Data

Training data | Test data

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Finding Parameters

Final evaluation | Test data

Ensemble Learning

Ensemble Learning: To combine the outputs of an ensemble of hypotheses (predictors/classifiers) with the aim of improving prediction accuracy.

Consider an ensemble of M hypotheses $h_1, h_2, \ldots h_M$ where we combine their predictions using a simple majority vote:

$$h_{overall}(x) = majority\{h_1(x), h_2(x), \ldots, h_M(x)\}$$

Ensemble Learning(cont.)

For example, we have 5 Decision Trees $h_1, h_2, \ldots, h_5$ trained with different pruning methods from the Iris data.

Given new data x each Decision Tree will make a prediction on which type of Iris it belongs to, e.g.

$h_1(x)$=*setosa*, $h_2(x)$=*virginica*, $h_3(x)$=*setosa*, $h_4(x)$=*setosa*, $h_5(x)$=*virginica*

The overall prediction on which type of Iris is simply

majority{setosa, virginica, setosa, setosa, virginica}

which amounts to *setosa*.

The question is, does this help in making more accurate predictions?

Ensemble Learning(cont.)

For the overall hypothesis to **misclassify**, it is clear that at least $(M-1)/2+1$ ensemble members must misclassify.

From the previous example, at least $(5-1)/2+1=3$ Decision Trees must misclassify before the overall prediction becomes wrong.

Assuming that each member hypothesis has a probability of error of p, and that the errors are independent, the overall probability of error is then:

$$error = \sum_{i=(M-1)/2+1}^{M} \binom{M}{i} p^i (1-p)^{M-i}$$

where $\binom{M}{i}$ is the binomial coefficient

$$\binom{M}{i} = \frac{M!}{i!(M-i)!}$$

i.e. the number of ways to choose i items out of M items.

Ensemble Learning(cont.)

For example, if $M = 5$ and $p = 0.1$, the probability of error for the classifier ensemble is

$$\sum_{i=3}^{5} \binom{5}{i} 0.1^i 0.9^{5-i} = \binom{5}{3} 0.1^3 0.9^2 + \binom{5}{4} 0.1^4 0.9^1 + \binom{5}{5} 0.1^5 0.9^0$$

which is less than 0.01!

Therefore, by combining a set of predictions from different hypotheses, we can actually reduce the overall probability of error.

Summary

- Always separate examples/data into a training set and a testing set to facilitate the assessment of predictors/classifiers before deployment. To prevent cheeking/peeking **ensure that the training and testing sets are disjoint.**
- Overfitting is a serious problem in learning algorithms that can negatively impact the prediction accuracy of predictors/classifiers.
- Decision Tree Pruning is a general method to minimise the effects of overfitting in Decition Tree learning.
- Ensemble learning (aggregating/combining the outputs of several different predictors) is also an effective method to improve the prediction accuracy of an AI system.