## Plan

In this lecture we review several Classical Machine Learning models.

These also allow us to introduce some concepts that are useful for many other models.

- Decision Trees
    - Simple Trees
    - Random Forests
        - introduce
            - Ensembles
            - Bootstrapping, Bagging
- Boosting
- Support Vector Machines
    - Margin Loss

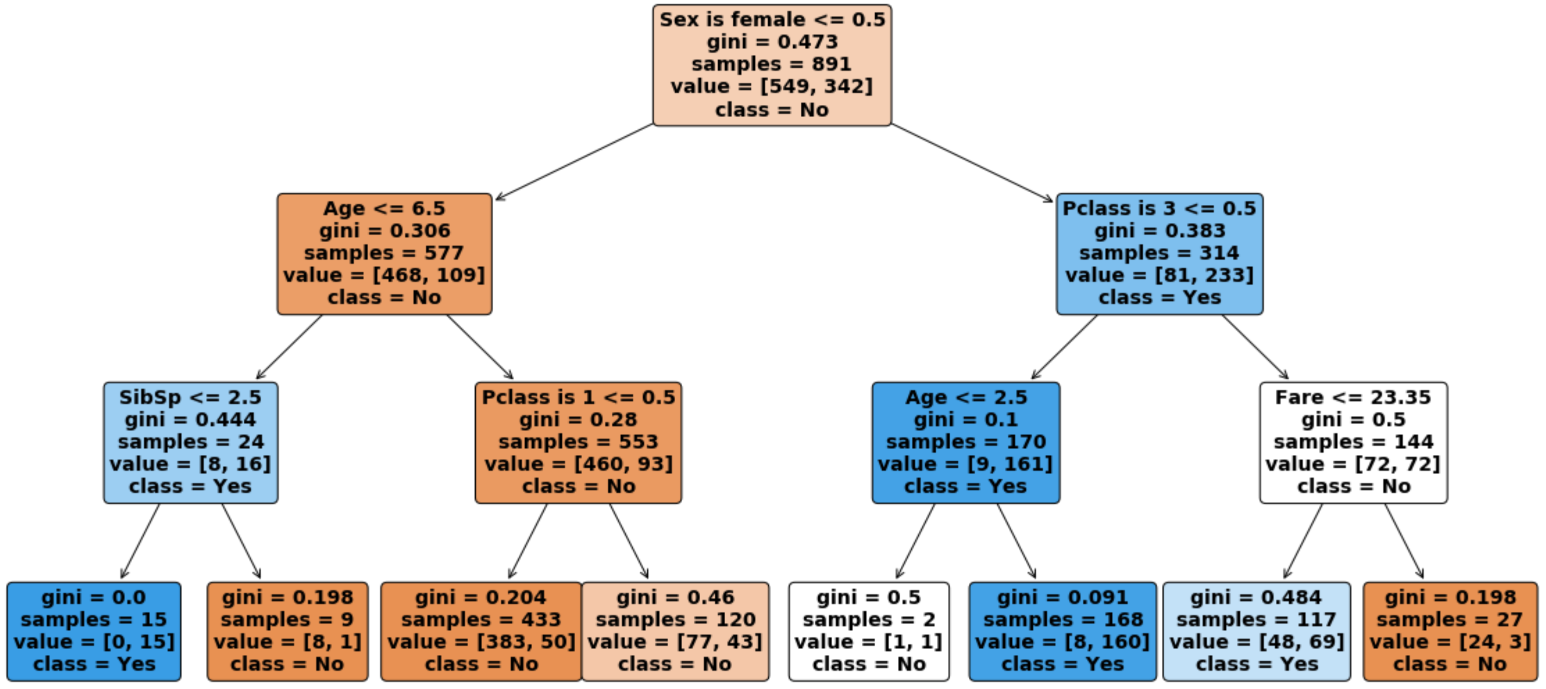
## Decision Tree Terminology

In contrast to other models (e.g., Logistic Regression), which had some mathematical basis, Decision Trees will feel very operational.

Let's dive in with an example: a Decision Tree to solve the Titanic Surival Classification task.

As usual we let $C$ denote the set of distinct categories/classes (possible targets) for our Classification task.

```
In [4]: th = dthelp.TitanicHelper()
        ret = th.make_titanic_png(max_depth=3)
        if hasattr(ret, "fname"):
            Image(filename=ret["fname"] + ".png")
```



## Nodes

- Each box is called a *node*
- There are two types of nodes
    - Those with no arrows exiting (called a *leaf* or *terminal* node)
    - Those with arrows exiting (called an *interior* or *non-terminal* node)
- The single node with no arrow entering is called the *root* node

## Edges

- An arrow, which is directed, is called an *edge*
    - The node from which an edge exits is called a *parent* node
    - The node to which an edge is directed is called a *child* node

Edges connect only a parent to a child.

The edges thus define an *acyclic* graph

## Labels

A non-leaf node is labelled with a True/False question/test that is applied to an example

- The test is evaluated on an example
- The left child of the node is associated with a True evaluation of the test
- The right child of the node is associated with a False evaluation of the test

A leaf node is labelled with one category/class in $C$

Consider the $n$-dimensional space of feature vectors $\mathbf{x}$

$$\mathrm{domain}(\mathbf{x}) = \mathrm{domain}(\mathbf{x}_1) \times \mathrm{domain}(\mathbf{x}_2) \times \ldots \times \ \mathrm{domain}(\mathbf{x}_n)$$

With each possible feature vector $\mathbf{x}$ in the space, we can associate a target $f(\mathbf{x})$.

We will take the liberty to have $S$ denote the infinite set of *all possible examples*

$$S = \{(\mathbf{x}, \mathbf{y}) \,|\, \mathbf{x} \in \mathrm{domain}(\mathbf{x}), \mathbf{y} = f(\mathbf{x})\}$$

- There is a subset of $S$ associated with each node: $S_{\mathrm{n}}$
- The test at node $\mathrm{n}$ partitions $S_{\mathrm{n}}$ into disjoint subsets $L_{\mathrm{n}}, R_{\mathrm{n}}$

$$\begin{aligned}
S_{\mathrm{n}} &= L_{\mathrm{n}} \cup R_{\mathrm{n}} \\
\phi &= L_{\mathrm{n}} \cap R_{\mathrm{n}} \\
L_{\mathrm{n}} &= \{(\mathbf{x}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})}) | (\mathbf{x}^{(\mathbf{i})}, y^{(\mathbf{i})}) \in S_{\mathrm{n}},\ \text{"True" answer to question}\} \\
R_{\mathrm{n}} &= \{(\mathbf{x}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})}) | (\mathbf{x}^{(\mathbf{i})}, y^{(\mathbf{i})}) \in S_{\mathrm{n}},\ \text{"False" answer to question}\}
\end{aligned}$$

That is, the question partitions $S_{\mathrm{n}}$ into "left" and right subsets $L_{\mathrm{n}}, R_{\mathrm{n}}$ depending on the answer.

## Example: Decision Tree for Titanic Survival

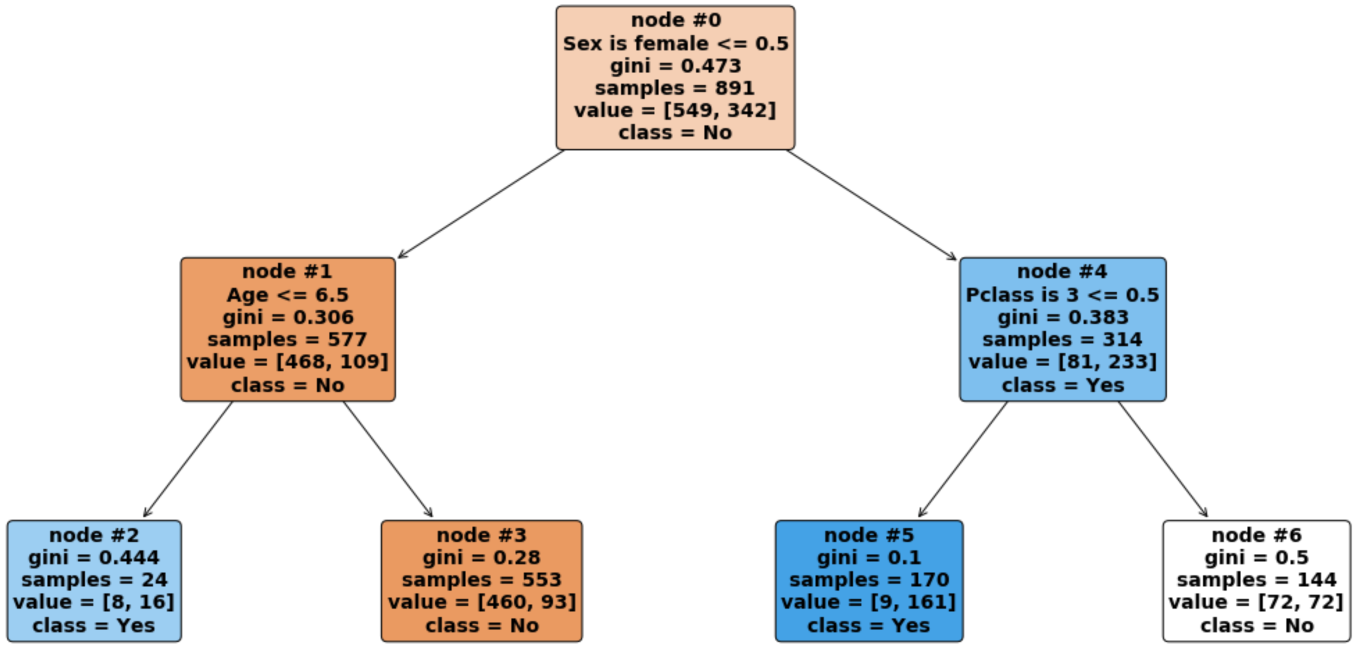
Let's illustrate using a Decision Tree for the Titanic Survival Classification task.

- High level
- We will subsequently explain the exact details for the labels and colors

In [5]:
```
th = dthelp.TitanicHelper()

ret = th.make_titanic_png(max_depth=2, node_ids=True)
if hasattr(ret, "fname"):
    Image(filename=ret["fname"] + ".png")

fig_titanic2 = ret["plt"]["fig"]
```



- Root node $\boxed{\text{\#0}}$
  - $S_{\#0} = S$ is the entire universe of examples
  - Labelled with question: "Is $\mathbf{x}$ *non* Female ?"
- Left child $\boxed{\text{\#1}}$
  - $S_{\#1} =$ subset of $S_{\#0}$ that are `Male'
- Right child $\boxed{\text{\#4}}$
  - $S_{\#4} =$ subset of $S_{\#0}$ that are `Female'

- Left child $\boxed{\#2}$
  - $S_{\#2}$ subset of $S_{\#1}$ with Age $\leq 6.5$
  - Is a *leaf*
  - Labeled with class `Yes` (i.e., Survive)
    - Corresponding to the subset of entire universe $S$ that are Males aged no more than 6.5 years
- Right child $\boxed{\#3}$
  - $S_{\#3}$ subset of $S_{\#1}$ with Age $> 6.5$
  - Is a *leaf*
  - Labeled with class `No` (i.e., did not Survive)

## Prediction

Given a test example (with features $\mathbf{x}$)

- Apply the sequence of questions to $\mathbf{x}$
  - Evaluate the test of the current node on $\mathbf{x}$
  - Depending on the evaluation
  - Evaluate the test of the Left/Right child of the current node
- This defines a path to a leaf node
- Prediction $\hat{y}$ is the class label of the leaf

This makes prediction in Decision Trees very fast.

Return to parent notebook

## Training: a first look at the algorithm

### Feature encoding

Before describing the algorithm, we enumerate the features we will use

- Numeric: Age, SibSp, Parch, Fare
- Categorical: Sex, Pclass

The categorical features will be One Hot Encoded

- The Sex feature is replaced by two binary features: Is Female, Is Male
- The Pclass feature is replaced by three binary indicator features $\text{Is}_\text{Class 1}, \text{Is}_\text{Is Class 2}, \text{Is}_\text{Class 3}$

So don't expect to see a test like `Sex == Male` ?

- Instead: "`Is Male == 1 ?`"
- Testing "$\text{Is}_\text{Male} == \text{True}$ ?"

## The training algorithm

We use the training examples as a proxy for $S$, the universe of examples

$$S_{\text{train}} = \langle \mathbf{X}, \mathbf{y} \rangle = [\mathbf{x}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})} | 1 \le i \le m]$$

to build the tree *recursively*.

Let's start with the root node

- Let $\mathrm{n}_{\text{root}}$ denote the root node
- Associate the entire set of training examples $S_{\text{train}}$ with $S_{\text{root}} = S_{\text{train}}$

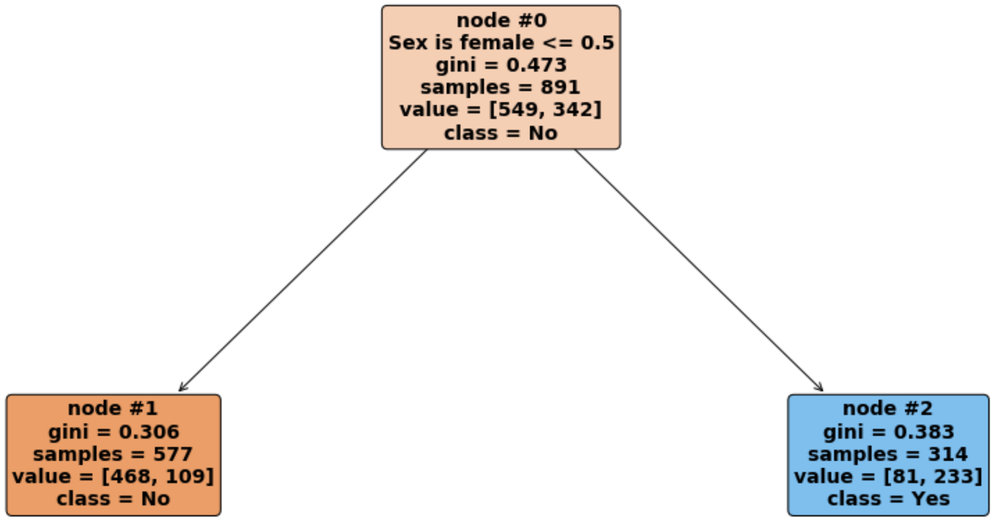
We construct a node $\mathrm{n}$ by the procedure `Construct(` $\mathrm{n}$ `)` :

- Use $S_{\mathrm{n}}$ to label $\mathrm{n}$ with a test/question
- The test splits $S_{\mathrm{n}}$ into
    - $L_{\mathrm{n}}$: the subset of $S_{\mathrm{n}}$ consisting of examples where the test is True
    - $R_{\mathrm{n}}$: the subset of $S_{\mathrm{n}}$ consisting of examples where the test is False
- if $L_{\mathrm{n}}$ is not empty:
    - create a new node $\mathrm{n}_L$ as the left child or $\mathrm{n}$
    - `Construct(`$\mathrm{n}_L$ `)`
- if $R_{\mathrm{n}}$ is not empty:
    - create a new node $\mathrm{n}_R$ as the right child of $\mathrm{n}$
    - `Construct(`$\mathrm{n}_R$ `)`

To illustrate, let's apply `Construct(` $n_{\text{root}}$ `)`:

```
In [6]: th = dthelp.TitanicHelper()

        ret = th.make_titanic_png(max_depth=1, node_ids=True)
```



We will describe the notation for each node.

- Root node: $\#0$
    - Labelled with question: "Is $\mathbf{x}$ *non* Female ?"
    - $S_{\#0}$ is the entire training set
        - `samples = 891`: This is number of examples in the training set
        - `values = [549, 342]`: The examples of $S_{\#0}$ can be divided into
            - 549 with `Survived` == No
            - 342 with `Survived` ==Yes

- Left child of root: $\#1$
  - `samples = 577`: This is the subset of $S_{\#0}$ (training set) consisting of the 577 non-`Female` (i.e.,`Male`) examples
  - `values = [468, 109]`
  - The examples of $S_{\#1}$ can be divided into
    - 468 with `Survived` == No
    - 109 with `Survived` == Yes

- Right child of root $\#2$:
  - `samples = 314`: This is the subset of $S_{\#0}$ (training set) consisting of the 577 `Female` examples
  - `values = [81, 233]`
  - The examples of $S_{\#2}$ can be divided into
    - 81 with `Survived` == No
    - 233 with `Survived` == Yes

At this point, the left and right children are both leaf nodes

- Left child of root: $\#1$
  - Labelled with prediction "No": `class = No`
- Right child of root: $\#2$
  - Labelled with prediction "Yes": `class = Yes`

"No" is colored orange "Yes" is colored blue

**Digression**

The question

> *"Is $\boldsymbol{x}$ non Female ?" -- encoded via the test:* `Sex is female <= 0.5`

is a bit contorted.

This is an artifact of the categorical feature `Sex` being replaced by binary features $\text{Is}_{\text{Female}}, \text{Is}_{\text{Male}}$.

> `Sex is female <= 0.5`

represents

$$\text{Is}_{\text{Female}} == 0$$

hence my translation to "non Female".

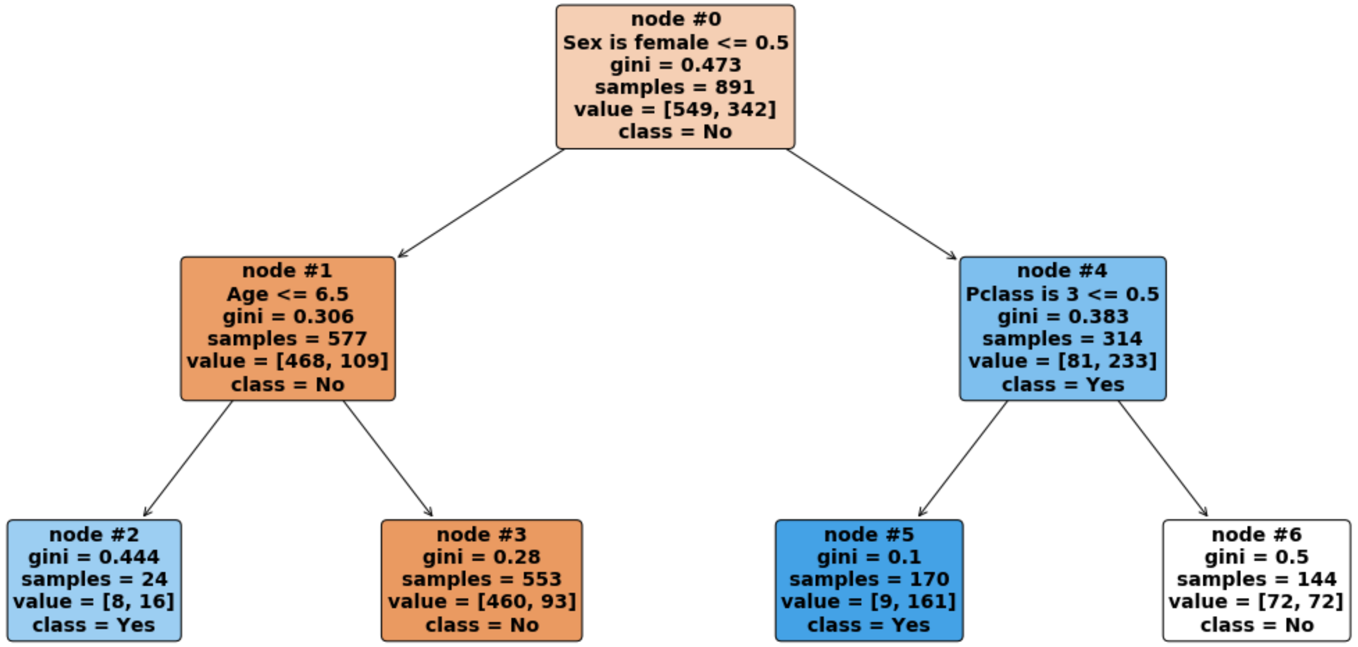
If the categorical variable had more than 2 classes, this test would appear less contorted.

We can now recursively apply `Construct(` $\#1$ `)` and `Consruct(` $\#2$ `)`

- Note that the numbering of the nodes changes

In [7]:
```
th = dthelp.TitanicHelper()

ret = th.make_titanic_png(max_depth=2, node_ids=True)
```



With the tree now depth 2, we have 4 leaf nodes.

If we were to continue this procedure indefinitely

- We would eventually have "pure" leaf nodes
    - All examples in the node are in the same class
    - Further splitting would not change the predicted class
    - The algorithm would stop

Return to parent notebook

## Training: a deeper look at the algorithm

### Encoding the test

The test evaluated at node $\mathrm{n}$ is a comparison

- Of a feature $\mathbf{x}_j$
- With a threshold value $t_{\mathrm{n},j}$

Thus we can represent the test at $\mathrm{n}$ as the pairs $(\mathbf{x}_j, t_{\mathrm{n},j})$.

Deciding the $j$ and $t_{n,j}$ at a node $\mathrm{n}$ will be at the heart of the algorithm.

## The threshold

Consider feature $\mathbf{x}_j$.

We restrict the possible threshold values $V_j$ for comparison with $\mathbf{x}_j$ to

- The distinct values of $\mathbf{x}_j$ in the training set

$$V_j = \{\mathbf{x}_j^{(\mathbf{i})} | 1 \leq i \leq m\}$$

This is true for both numeric and categorical features $\mathbf{x}_j$.

## Note

- A variant uses the *mid-point* between distinct values
- The labelling of questions in our diagram always uses the comparison
$$\mathbf{x}_j \le \text{midpoint value}$$
- So categorical tests look like
$$\text{Is Female} \le 0.5$$
which is equivalent to
$$\text{Is Female} == 0$$

## Choosing the test

There are

- A finite number ($n$) of features
- A finite number $||V_j||$ of distinct values for the threshold

So there are only a countable set of possible choices for the test.

How do we choose the test $(\mathbf{x}_j, t_{\text{n},j})$ with which to label a non-leaf node ?

sklearn manual (https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation)

We will describe the algorithm for choosing the test.

Let's re-write our initial algorithm to give us a little more flexibility:

Initialization:

- Let $\text{n}_\text{root}$ denote the root node
- Associate the entire set of training examples $S_\text{train}$ with $S_\text{root} = S_\text{train}$

Here is pseudo-code for a procedure `split(` $\mathrm{n}, S_\mathrm{n}$ `)` to construct a sub-tree rooted at node $\mathrm{n}$:

```
• if we *can split* $S_\mathrm{n}$
    ▪ Determine the question that "best" splits $S_\mathrm{n}$ into $L_\mathrm{n}, R_\mathrm{n}$
    ▪ Create a child node $\mathrm{n}_L$ with corresponding examples $L_\mathrm{n}$
    ▪ Create a child node $\mathrm{n}_R$ with corresponding examples $R_\mathrm{n}$
    ▪ split$(\mathrm{n}_L, L_\mathrm{n})$
    ▪ split$(\mathrm{n}_R, R_\mathrm{n})$
```

This pseudo-code was vague on

- How do we define the "best" split of the examples at node $\mathbf{n}$ ?
- What does "if we can split"a node mean ?

We answer each in turn.

## Measuring the quality of a split

In order to determine the "best" split, we need a metric of the quality of the split.

We start with measuring the "randomness" of a node $\mathrm{n}$

- The examples $S_\mathrm{n}$ may correspond to different classes, each with its own frequency.
- So $S_\mathrm{n}$ induces a probability distribution on the class labels.
- We need a metric that measures the randomness of this distribution at node $\mathrm{n}$

If we had a metric of randomness, we can define our quality metric of a split as the difference between

- The randomness of $S_{\mathrm{n}}$
- The weighted (by size) randomness of nodes $L_{\mathrm{n}}, R_{\mathrm{n}}$

We call this metric the *information gain* achieved by the split.

The *best split* would be the one which maximizes the information gain.

Any ideas for a metric of randomness of a distribution ?

Entropy !

This a a very good measure to use.

In the interest of showing alternatives, we continue with a different choice.

Gini.

## Gini score

For node $\mathrm{n}$:

- Let $p_{\mathrm{n},c}$ be the fraction of $S_{\mathrm{n}}$ examples with class $c \in C$

$$p_{\mathrm{n},c} = \frac{count_{\mathrm{n},c}}{|S_{\mathrm{n}}|}$$

Then the Gini score (metric of randomness) of node $\mathrm{n}$ is defined as

$$G_{\mathrm{n}} = 1 - \sum_{c \in C} {p_{\mathrm{n},c}}^2$$

$G_{\mathrm{n}}$ is called the **impurity** of node $\mathrm{n}$

**Note**

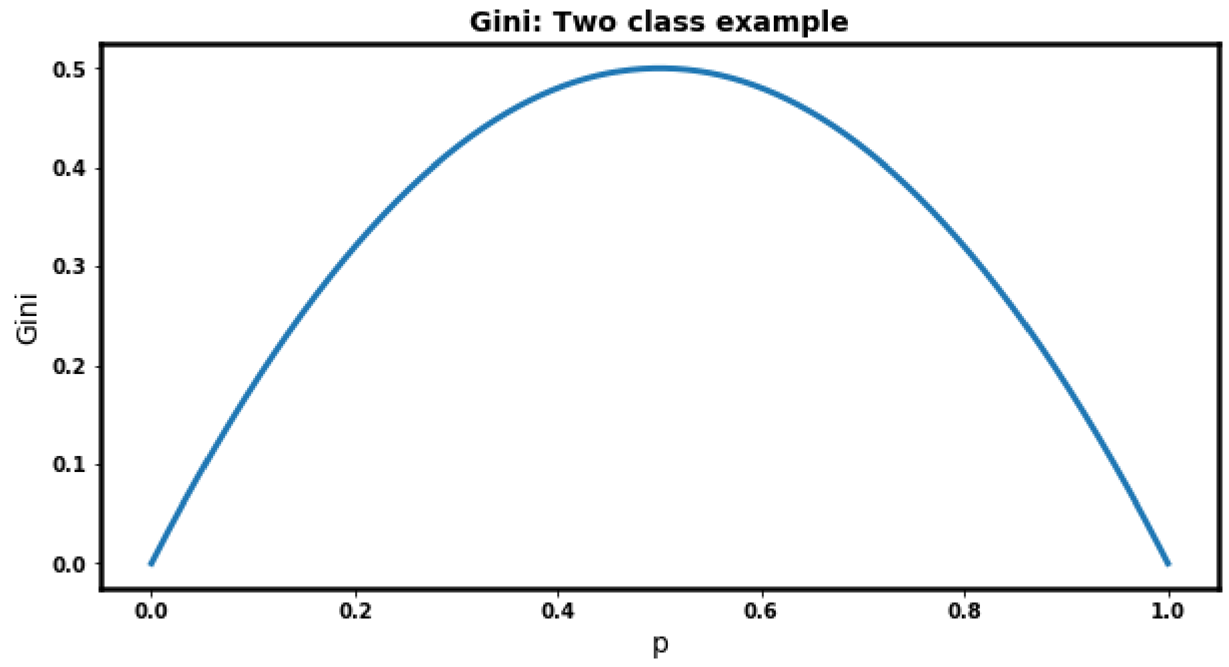
We will try to minimize *impurity* (just as we would minimize entropy)

Goal is to have pure nodes, i.e., all examples in node $\mathrm{n}$ are in same target class.

Here's what Gini looks like as a function of $p_{\mathrm{n},c}$ for binary $C$

In [8]:

```
gh = dthelp.GiniHelper()
_ =gh.plot_Gini()
```



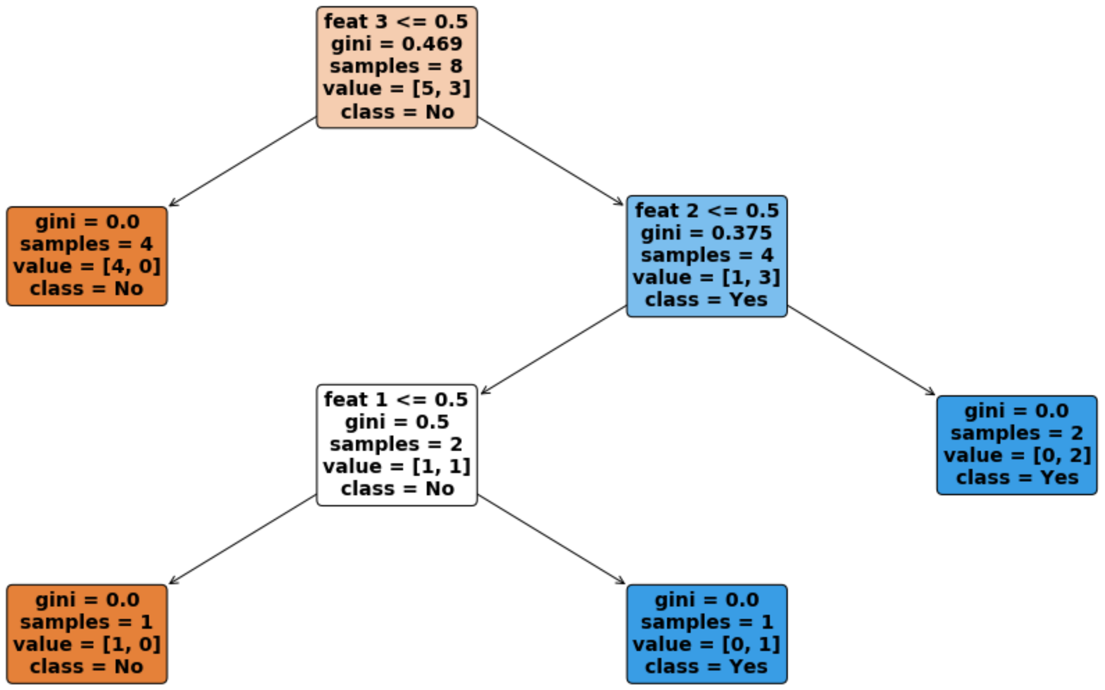
Just like Entropy, Gini impurity

- Is minimized by a pure distribution (of either class)
- Is maximized by an equally balanced distribution

## Gini illustration

It will be easiest to illustrate with a toy example having only categorical features.

In [9]:
```
gh = dthelp.GiniHelper()
_ = gh.make_logicTree_png()
```

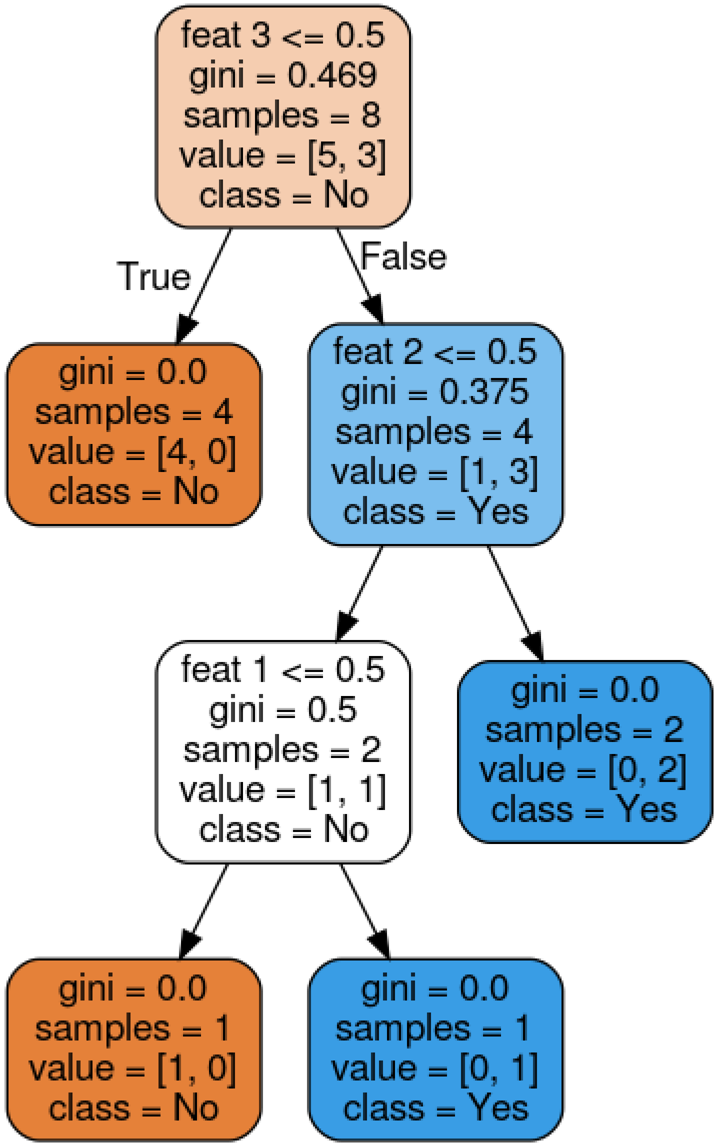


Here are the examples

```
In [10]: df_lt = gh.df_lt
         df_lt
```

Out[10]:

| | feat 1 | feat 2 | feat 3 | target |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 |


feat 3 <= 0.5
gini = 0.469
samples = 8
value = [5, 3]
class = No
True
False
gini = 0.0
samples = 4
value = [4, 0]
class = No
feat 2 <= 0.5
gini = 0.375
samples = 4
value = [1, 3]
class = Yes
feat 1 <= 0.5
gini = 0.5
samples = 2
value = [1, 1]
class = No
gini = 0.0
samples = 2
value = [0, 2]
class = Yes
gini = 0.0
samples = 1
value = [1, 0]
class = No
gini = 0.0
samples = 1
value = [0, 1]
class = Yes

Let's compute Gini of the root

```
In [11]: gini_lt = gh.gini(df_lt, gh.target_name_lt, gh.feature_names_lt, noisy=True)

         print("\n\nMethod returns ", gini_lt)
```

```
Gini, by hand:
Count by target:

0    5
1    3
Name: target, dtype: int64
Frequency by target:

0    0.625
1    0.375
Name: target, dtype: float64

1 - sum(freq**2) = 0.469


Method returns  0.46875
```

And Gini of the right sub-tree

In [12]:
```
df_right = df_lt[ df_lt["feat 3"] > 0.5 ]

gh.gini( df_right, gh.target_name_lt, gh.feature_names_lt, noisy=True)
```

```
Gini, by hand:
Count by target:

1    3
0    1
Name: target, dtype: int64
Frequency by target:

1    0.75
0    0.25
Name: target, dtype: float64

1 - sum(freq**2) = 0.375
```

Out[12]:
```
0.375
```

## The "best" $(j, k)$ split: From Gini of children, to Cost for split at parent

Let node $\mathrm{n}$

- Have child nodes $\mathrm{n}_L, \mathrm{n}_R$
- $S_\mathrm{n}$ denote the set of examples corresponding to node $\mathrm{n}$
- Have Gini impurity score $G_\mathrm{n}$

We can associate a Cost with the choice of splitting node $\text{n}$ with question $(j, t_{\text{n},j})$:

$$\text{Cost}_{j,t_{\text{n},j}}(S_\text{n}) = \frac{m_L}{(m_L + m_R)} G_L + \frac{m_R}{(n_L + m_R)} G_R$$

- where $m_L$
  $=$
  $|L_\text{n}$
  $|, m_R$
  $=$
  $|R_\text{n}|$

That is, the Cost of splitting $S_\text{n}$ on $X_j \leq t_{\text{n},j}$ is

- The weighted sum of the Gini's of the partitions created.

Finally:

- The best split $(j, t_{\mathrm{n},j})$ for node $\mathrm{n}$ is the one that minimizes the Cost
$$j, t_{\mathrm{n},j} = \operatorname*{argmin}_{j, t_{\mathrm{n},j}} \mathrm{Cost}_{j, t_{\mathrm{n},j}}$$
- $1 \le j \le n$
- $t_{\mathrm{n},j} \in V_j$

This is the split that maximizes Information Gain (since $G_{\mathrm{n}}$ is constant, relative to the choices)

Let's examine the cost of split at the root for each (binary) feature

```
In [13]: gh.cost(df_lt, gh.target_name_lt, gh.feature_names_lt, noisy=True)

Split feature feat 1 on 0.00
        G_left (# = 4) = 0.375, G_right (# = 4) = 0.500
        weighted (G_left, G_right) = 0.438
Split feature feat 2 on 0.00
        G_left (# = 4) = 0.375, G_right (# = 4) = 0.500
        weighted (G_left, G_right) = 0.438
Split feature feat 3 on 0.00
        G_left (# = 4) = 0.000, G_right (# = 4) = 0.375
        weighted (G_left, G_right) = 0.188

Out[13]: 0.1875
```

So split on $(3, 0)$ (feature "feat 3", threshold 0) gives the minimum cost.

That explains the split at the root.

## What does ”if we can split”a node mean

Time to answer our second vague statement: is there a time when we can't/shouldn't split node $\mathrm{n}$

**When can't we split $S_{\mathrm{n}}$ ?**

- $|S_{\mathrm{n}}| = 0$
    - an empty child, which we ignore (n.b., whose sibling is pure)
- $S_{\mathrm{n}}$ is pure

**When shouldn't we split $S_\mathrm{n}$ ?**

One obvious case

- When the Information Gain of **all** possible splits is negative

There are some less obvious cases related to the Performance Measure of our out of sample test set.

If we don't restrict the answer to "if we can split"

- Then we will eventually have leaf nodes that are all pure.
- That's good, but it's also possible to have a single example corresponding to a leaf node
- Overfitting !
    - Memorize training: each leaf memorizes an example

## Prediction $\hat{\mathbf{y}}_\mathbf{n}$ for node $\mathbf{n}$

We need to label a *leaf* node $\mathrm{n}$ with a category $c \in C$.

This will be the prediction $\hat{\mathbf{y}}^{(\mathbf{i})}$ that will be made for test example $\mathbf{x}^{(\mathbf{i})}$.

We do this by choosing

- The class $c$
- That occurs most frequently in $S_\mathrm{n}$, the set of training examples associated with node $\mathrm{n}$

$$
\begin{aligned}
count_{\mathrm{n},c} &= |\{\, i \mid (\mathbf{x}^{(\mathbf{i})}, \mathbf{y}^{(\mathbf{i})}) \in S_\mathrm{n}, \mathbf{y}^{(\mathbf{i})} = c \,\}| \\
\hat{\mathbf{y}}_\mathrm{n} &= \underset{c \in C}{\operatorname{argmax}}\, count_{\mathrm{n},c}
\end{aligned}
$$

**Note**

Our diagrams use the same logic for labelling non-leaf nodes with a class.

Return to parent notebook

## Decision Tree Regression

A Classifier (like the Decision Tree) that partitions examples can be modified to solve a Regression task.

We simply need to modify

- The category label assigned to a leaf node
- The measure used for the quality of a split

A category label is associated with each leaf node $\mathrm{n}$

- Classification: label is target category occuring with highest frequency in the examples in $S_{\mathrm{n}}$
$$\hat{\mathbf{y}}_{\mathrm{n}} = \operatorname*{argmax}_{c \in C} count_{\mathrm{n},c}$$
- Regression: label is *average* of the targets of the examples in $S_{\mathrm{n}}$ $$ \hat{y}*node{n}* = *dfrac{1}*{| *S*\node{n} |}\sum\limits*{\scriptstyle* (*x*^*ip*, *y*^*ip*) *in*
*S*\node{n}} \y^{(i)}

**$$**

The quality of the split of $S_{\text{n}}$ into $L_{\text{n}}$ (size $m_L$) and $R_{\text{n}}$ (size $m_R$)

- Classification: minimize weighted impurity (or entropy) of the subsets created by split
$$\begin{array}{lll} \text{Cost}_{j,t_{\text{n},j}}(S_{\text{n}}) & = & \frac{m_L}{(m_L+m_R)}G_L + \frac{m_R}{(n_L+m_R)}G_R \\ \text{where} \\ G_s & = & \text{impurity/entropy of set s}, s \in L, R \end{array}$$

- Regression: minimize the MSE of the subsets created by split $$ \begin{array}[lll]\ \text{Cost}{j, t{\node{n},j}}(S_\node{n}) & = & \dfrac{m_L}{(m_L + m_R)}\text{MSE}_L + \dfrac{m_R}{(m_L + m_R)}\text{MSE}_R \ \text{where } \ \text{MSE}_s & = & \text{MSE of set s}, s \in {L,R}\

```
       & = & \frac{1}{|s|} \sum_{(\x^\ip, y^\ip) \in s} { (\hat{\y}_s -
 \y^\ip)^2 }& \text{where } \hat{\y}_s \text{ is the predicted value for
all examples in set } s \\
```

  \end{array} $$

Return to parent notebook

## Overfitting example

Here is a Regression task that illustrates the tendency of "deep" trees to overfit.

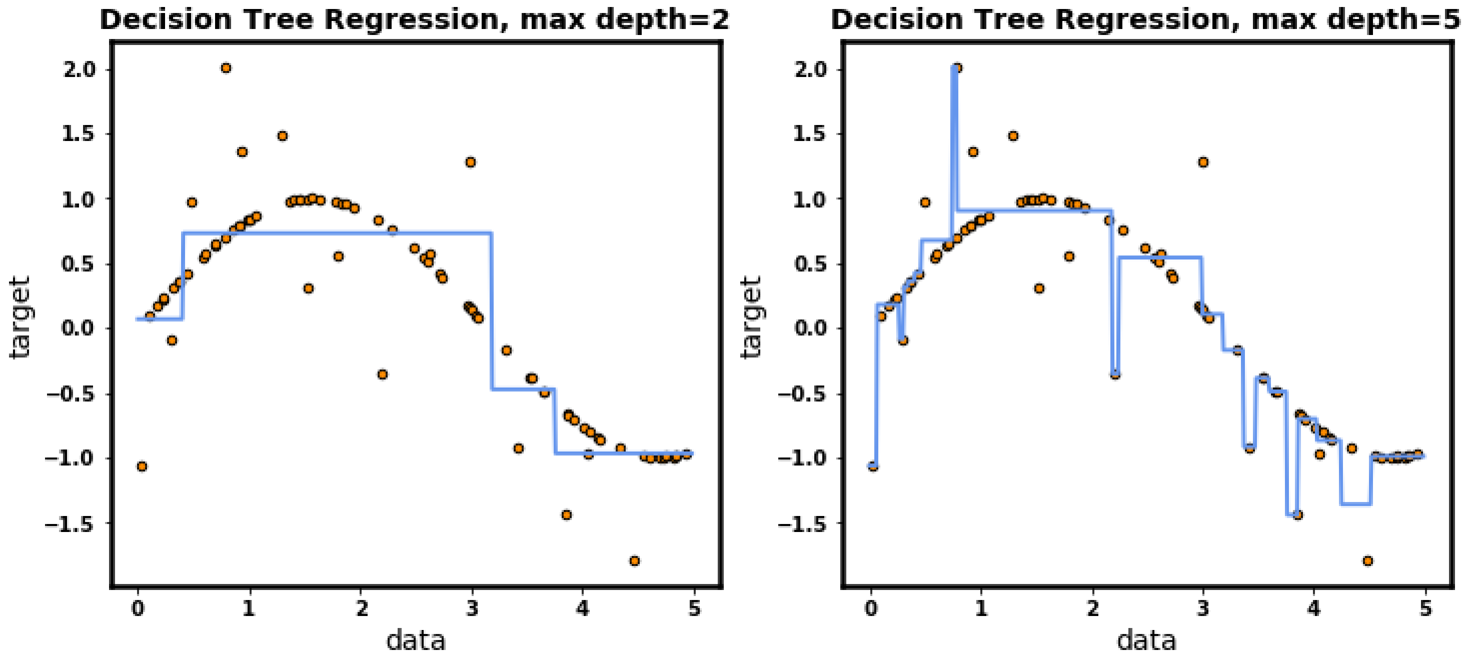
The light blue line is the "line of best fit".

In [14]:

```
rh = dthelp.RegressionHelper()

ret =rh.make_plot()
overfit_fig, trees_fig = ret["fig1"], ret["fig2"]
```

In [15]:
```
overfit_fig
```
Out[15]:



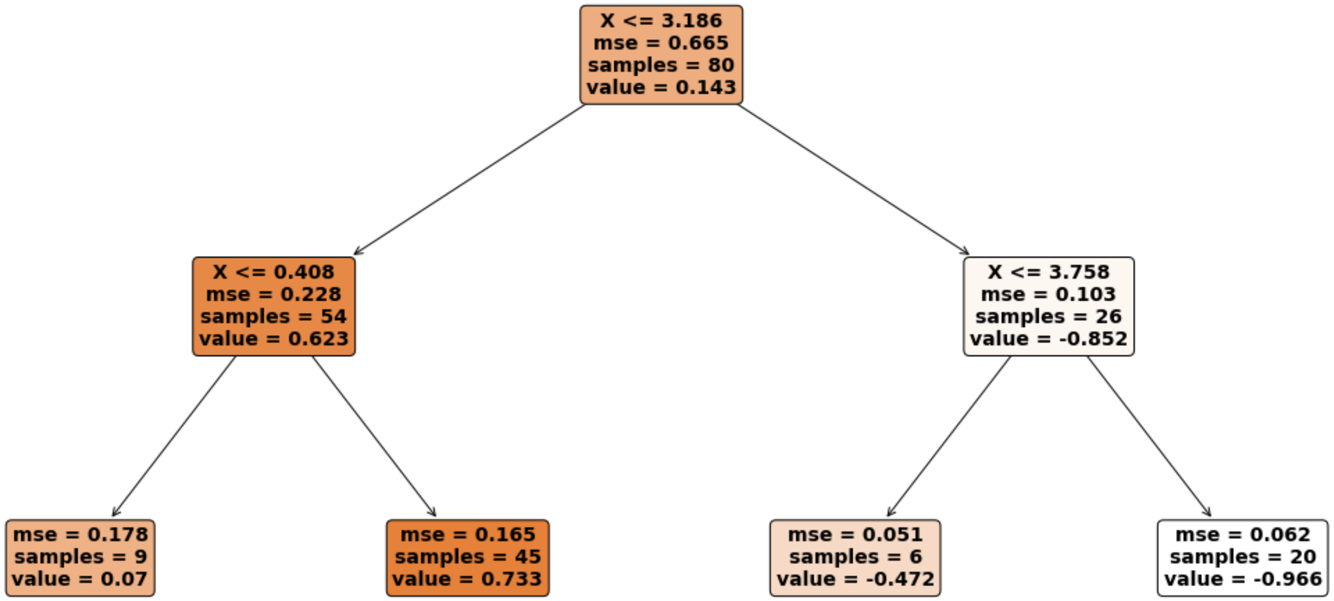
The deeper tree results in a very complex "line".

Here is the shallower (depth 2) tree, corresponding to the graph on the left.

- Each test paritions the horizontal axis

```
In [16]: trees_fig[0]

Out[16]:
```



Return to parent notebook

## Hyper parameters for Decision Trees

### Hyper parameters to control overfitting

You can combat overfitting with several parameters

- `max_depth`: maximum depth of tree
- `min_samples`: minimum size (no. of observations) to split a node
- `min_samples_leaf`: minimum number of samples for a leaf

## Other hyper parameters

Recall that our threshold $t_{\mathrm{n},j}$ for feature $\mathbf{x}_j$ was drawn from $V_j$

- Set of distinct values of $\mathbf{x}_j$ in the training set

This can be quite large. Perhaps defining a smaller number of threshholds may work even better.

- The choice of threshholds is a hyper paramter

Return to parent notebook

```
In [17]: print("Done")
```

```
Done
```