Decision Trees

CSE 4308/5360: Artificial Intelligence I

University of Texas at Arlington

An Example of a Learning Problem

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

- The restaurant waiting example, from the textbook.
- Each pattern X_i is a set of 10 attribute values, describing the case of a customer asked to wait at the restaurant.

An Example of a Learning Problem

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

- Some attributes are boolean, like *Alt* (are there alternative restaurants nearby?), *Bar* (does the restaurant have a bar?), *Fri* (is it weekend?), *Hun* (is the customer hungry?), *Rain* (is it raining?), *Res* (was a reservation made?)

An Example of a Learning Problem

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

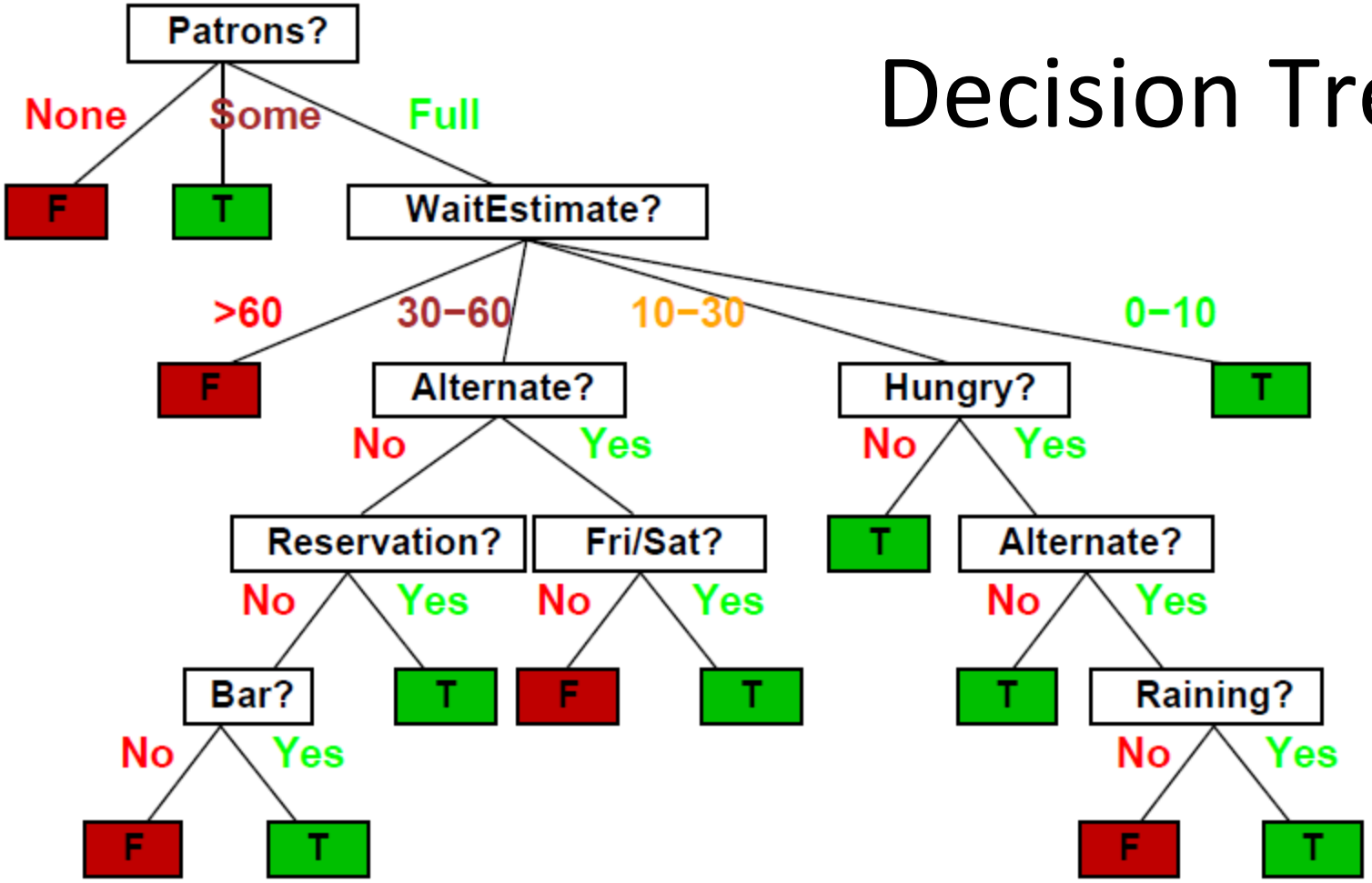
- Attribute *Pat* (how many people are in the restaurant) takes three values.
- Attribute *Price* takes three values. Attribute *Type* takes four values.
- Attribute *Est* (estimated wait time) takes four values.

An Example of a Learning Problem

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

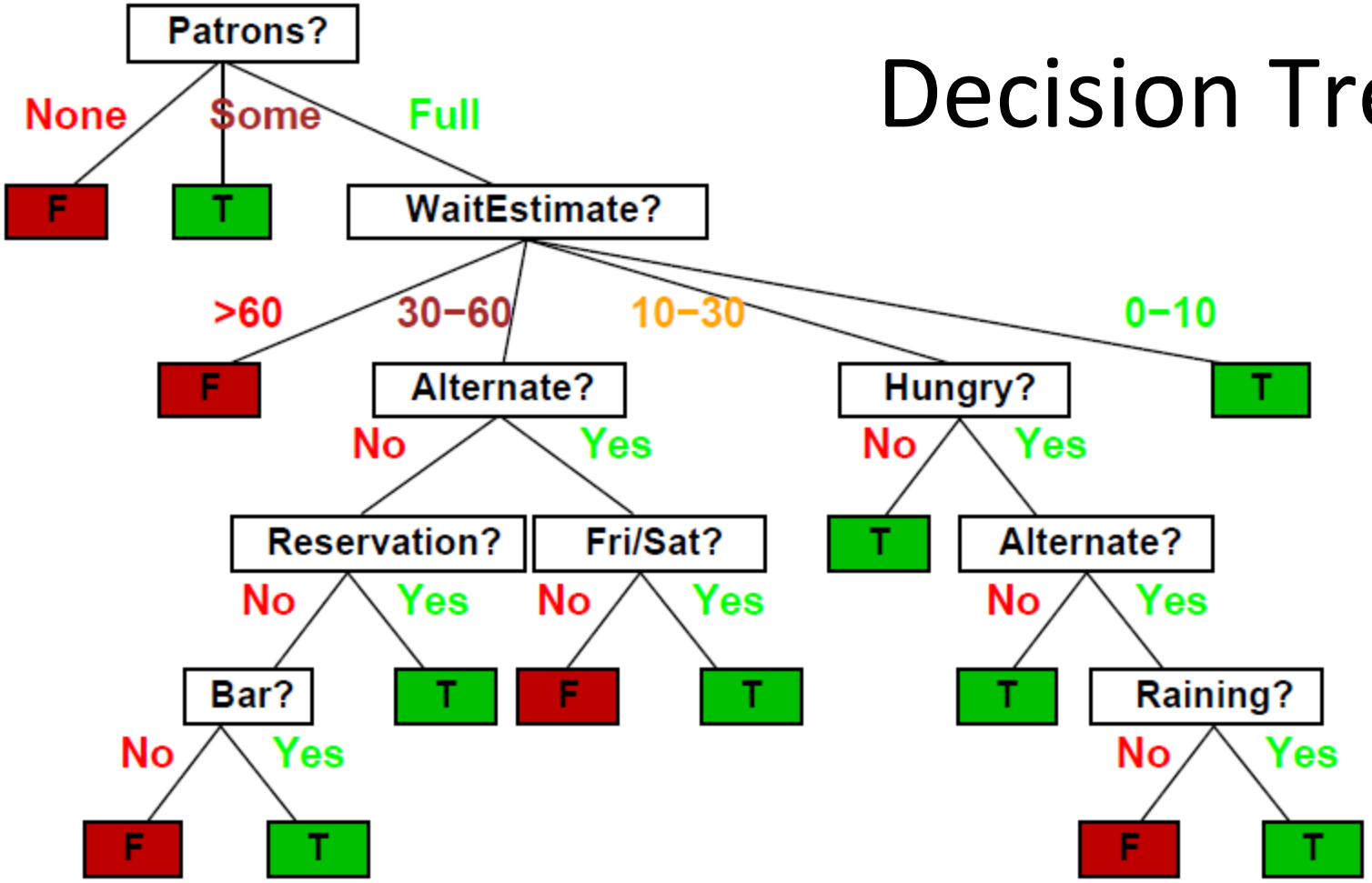
- Using these training examples, we want to learn a function F, mapping each pattern into a boolean answer (**will wait**, or **will not wait**).

Decision Trees



- A decision tree is a type of classifier. The input is a pattern. The output is a class.
- Given a pattern: we start at the root of the tree.
- The current node asks a question about the pattern.
- Based on the answer, we move to a child of the current node.
- When we get to a leaf node, we get the output of the decision tree (a class).

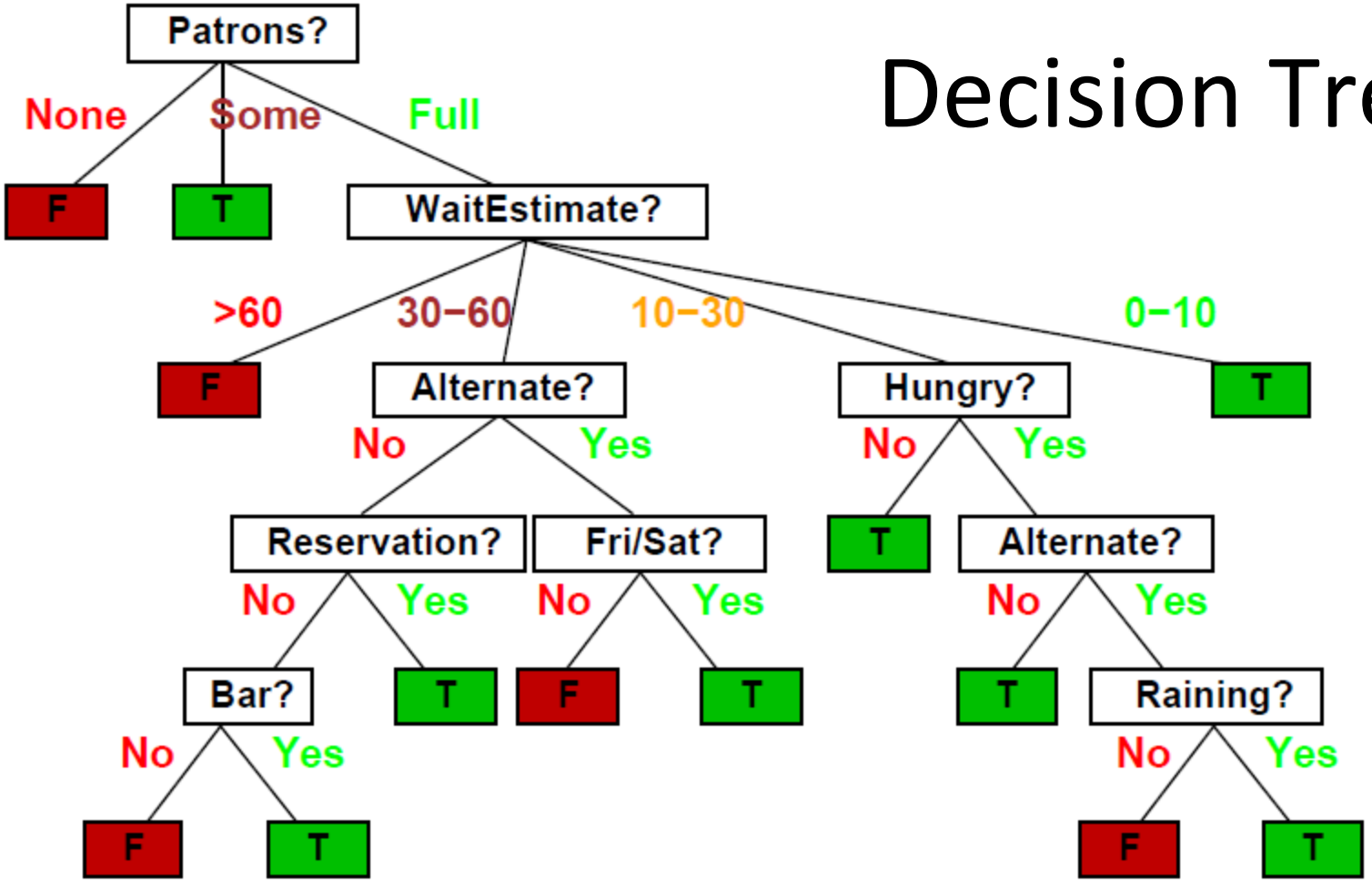
Decision Trees



- For example, what is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Some	$$	T	F	Thai	10-30

Decision Trees

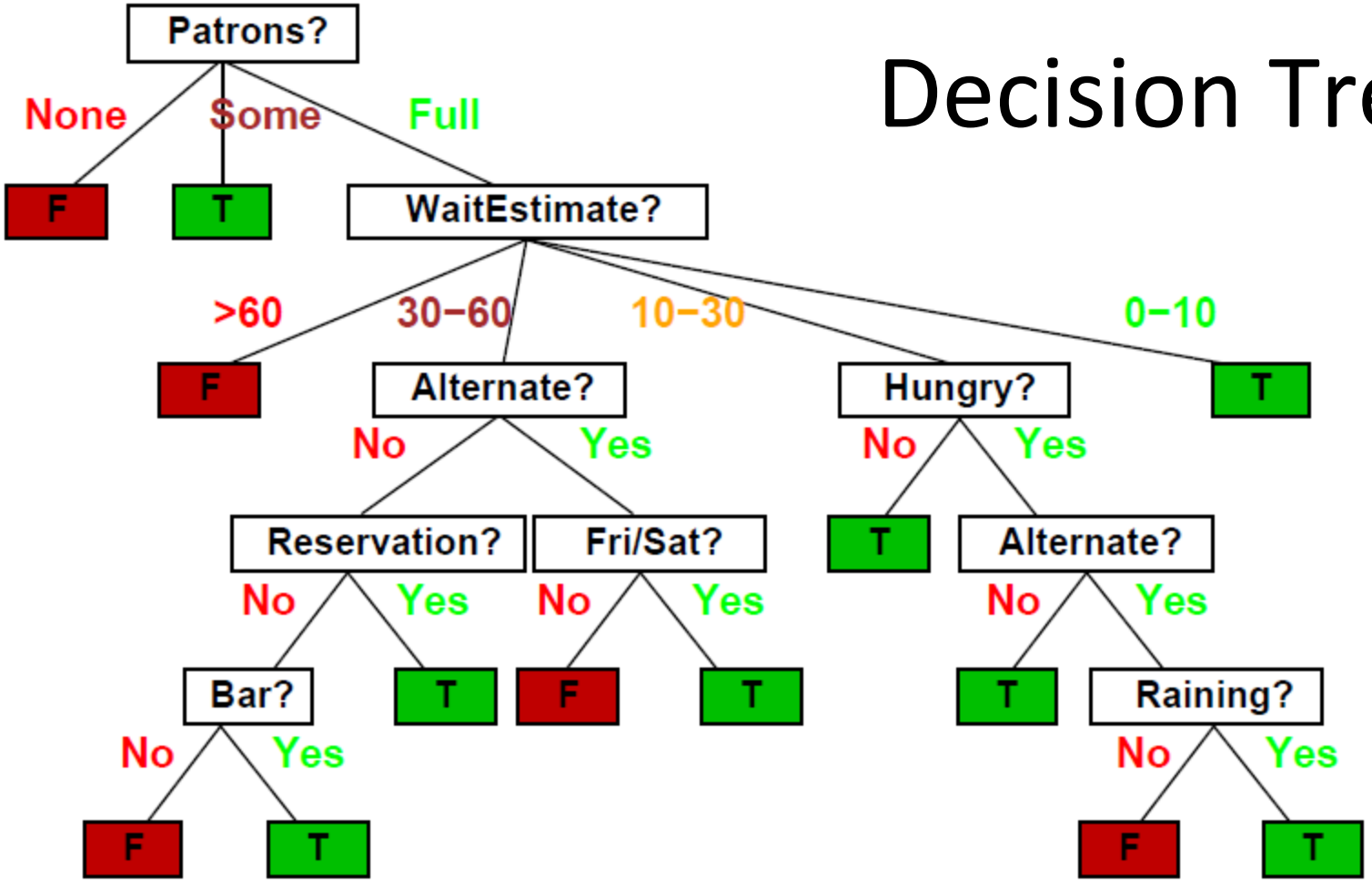


- For example, what is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Some	$$	T	F	Thai	10-30

- First check: ???

Decision Trees

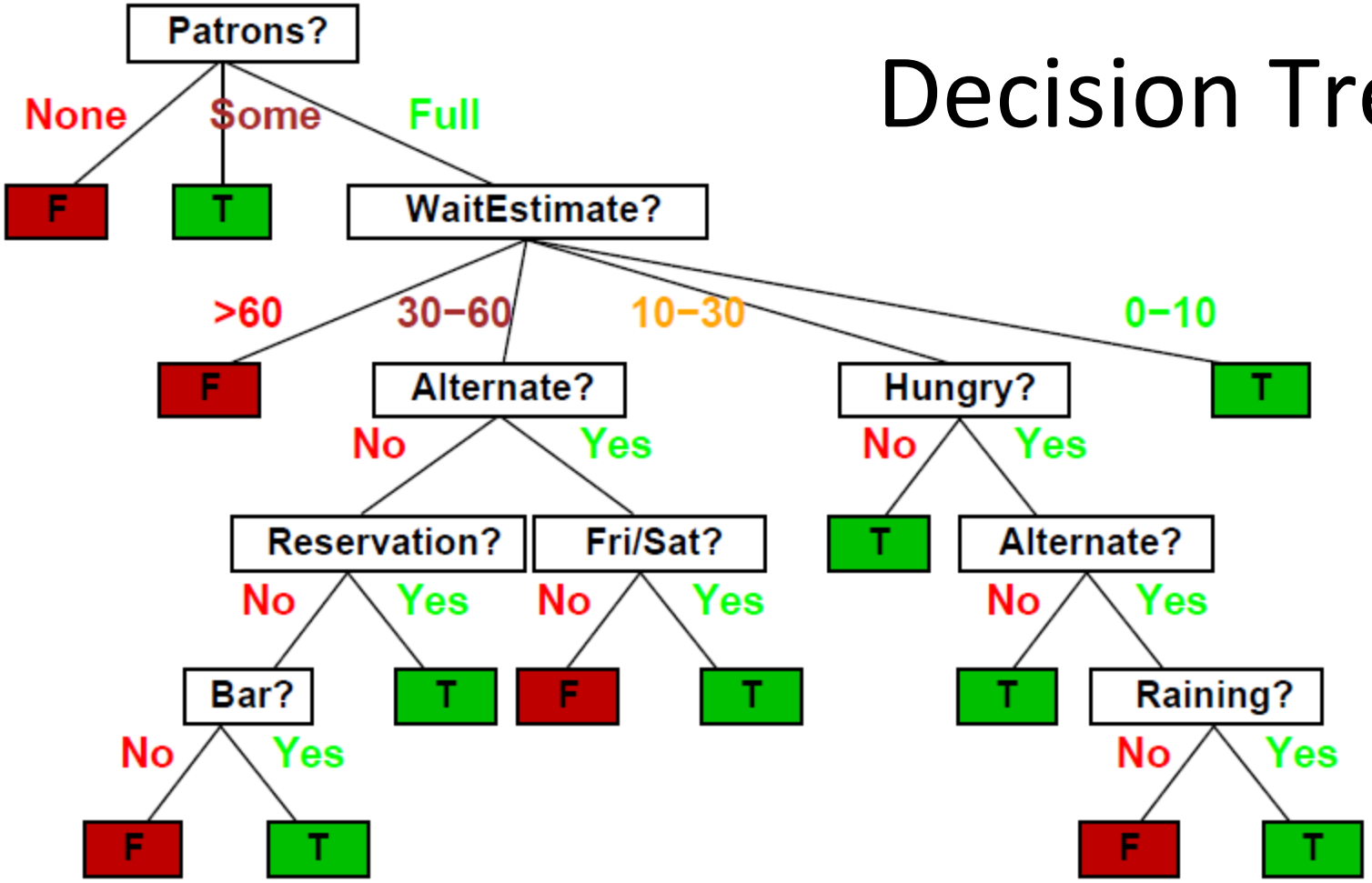


- For example, what is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Some	$$	T	F	Thai	10-30

- First check: value of *Patrons*? **Some**.
- Where do we go next?

Decision Trees

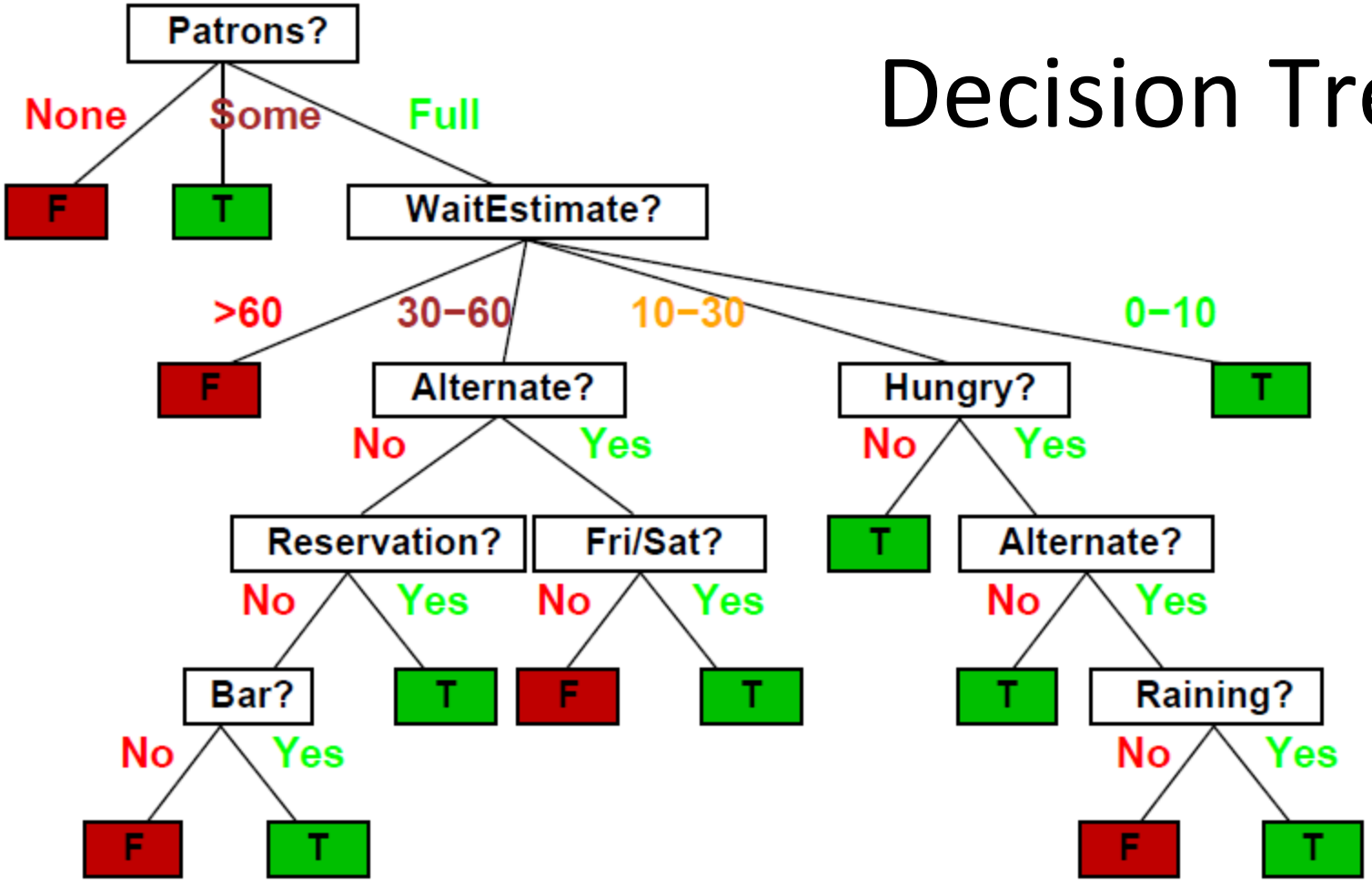


- For example, what is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Some	$$	T	F	Thai	10-30

- First check: value of *Patrons*? **Some**.
- Where do we go next? To the middle child. What happens next?

Decision Trees

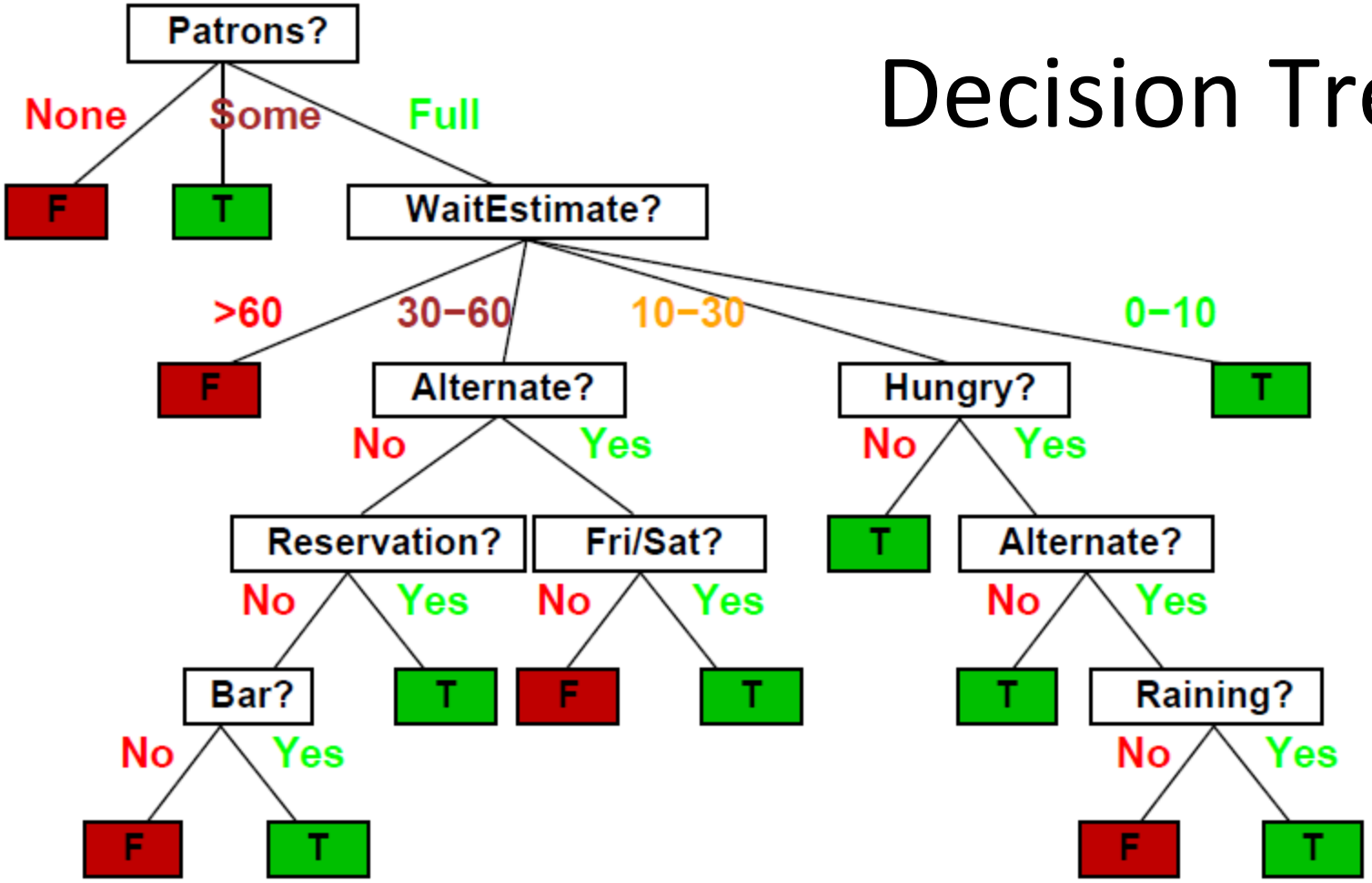


- For example, what is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Some	$$	T	F	Thai	10-30

- First check: value of *Patrons*? **Some**.
- Where do we go next? To the middle child. Leaf node, output is **will wait**.

Decision Trees

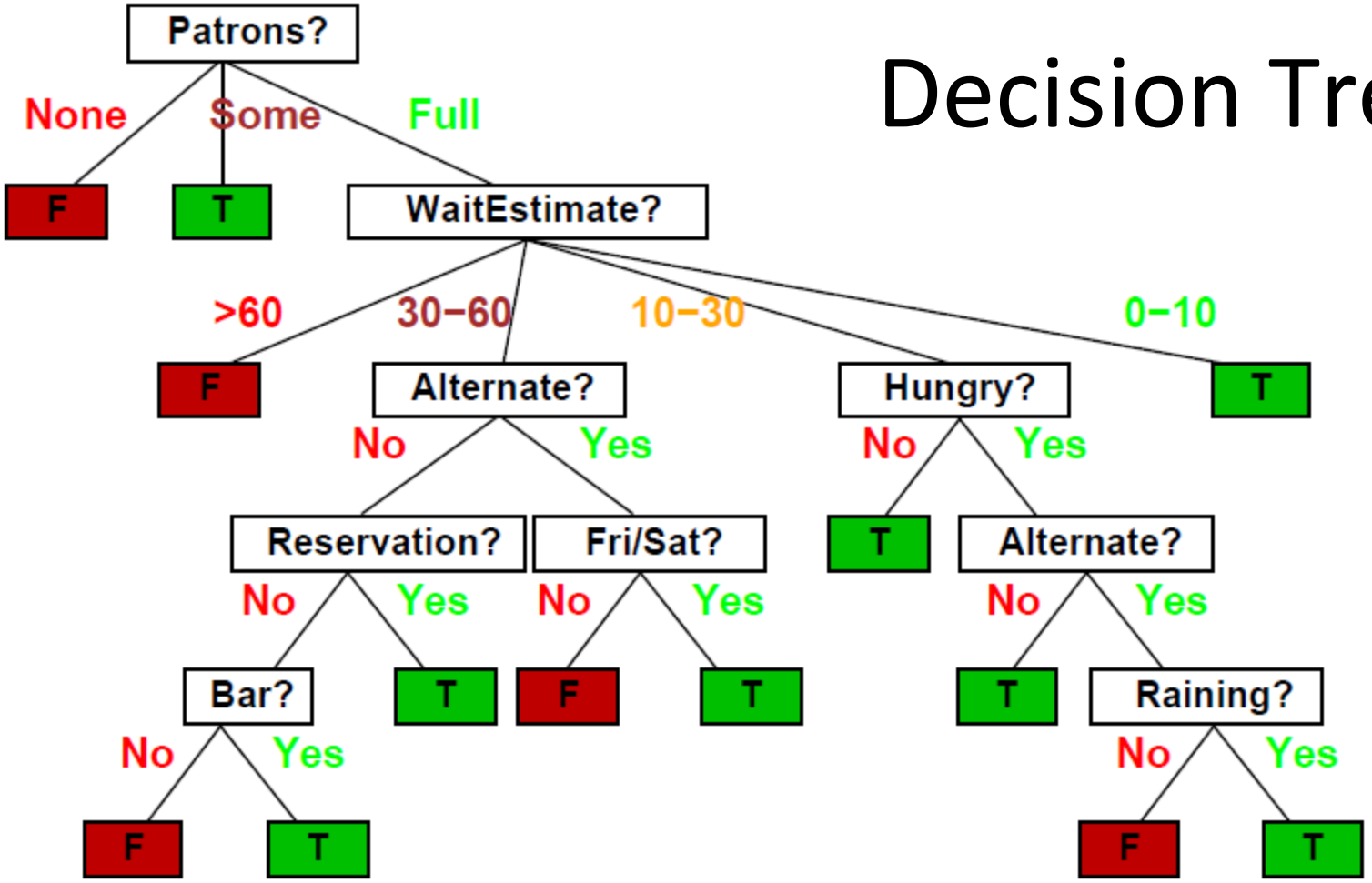


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

- First check: ???

Decision Trees

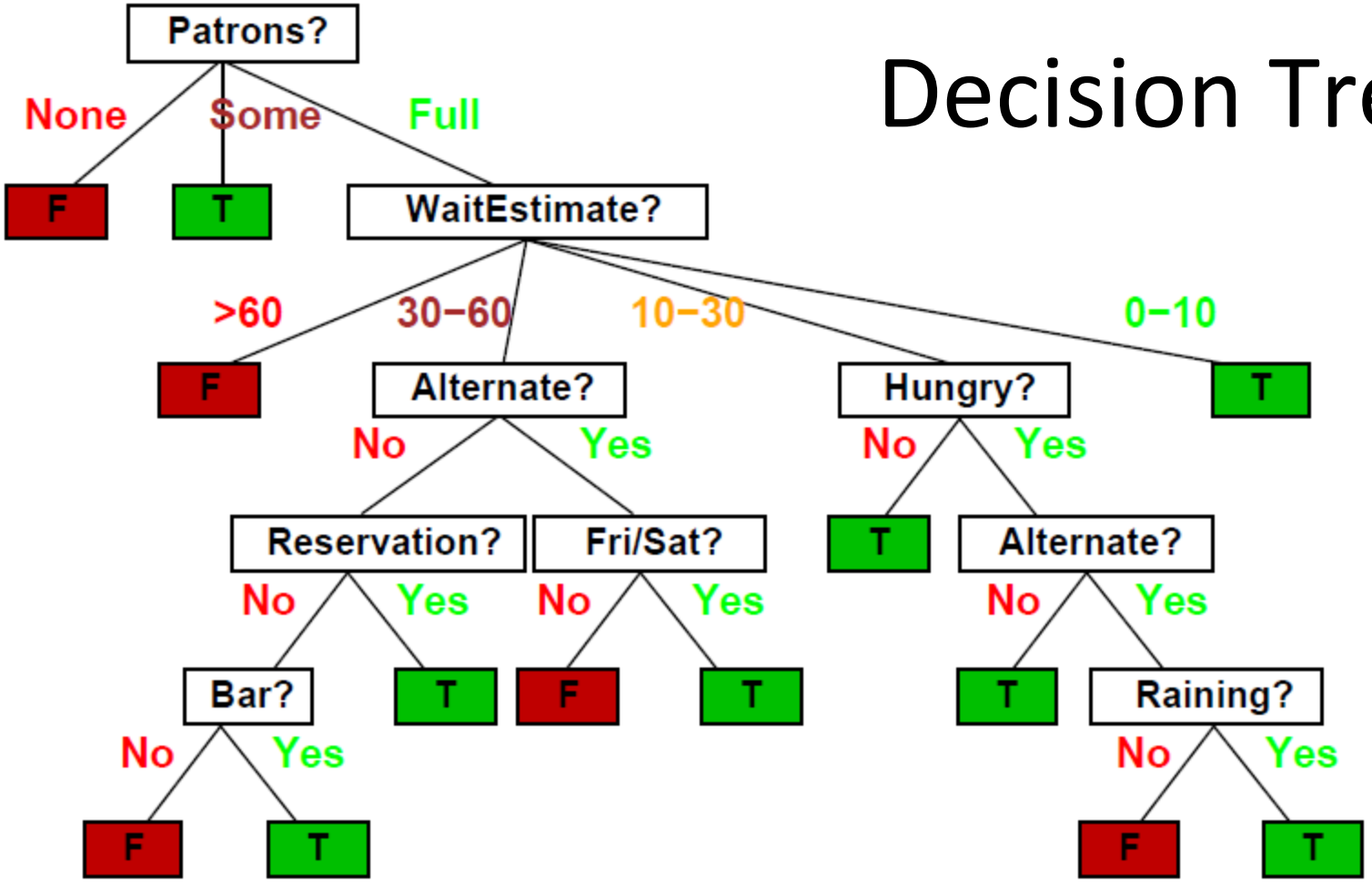


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

- First check: value of *Patrons*? **Full**.
- Where do we go next?

Decision Trees

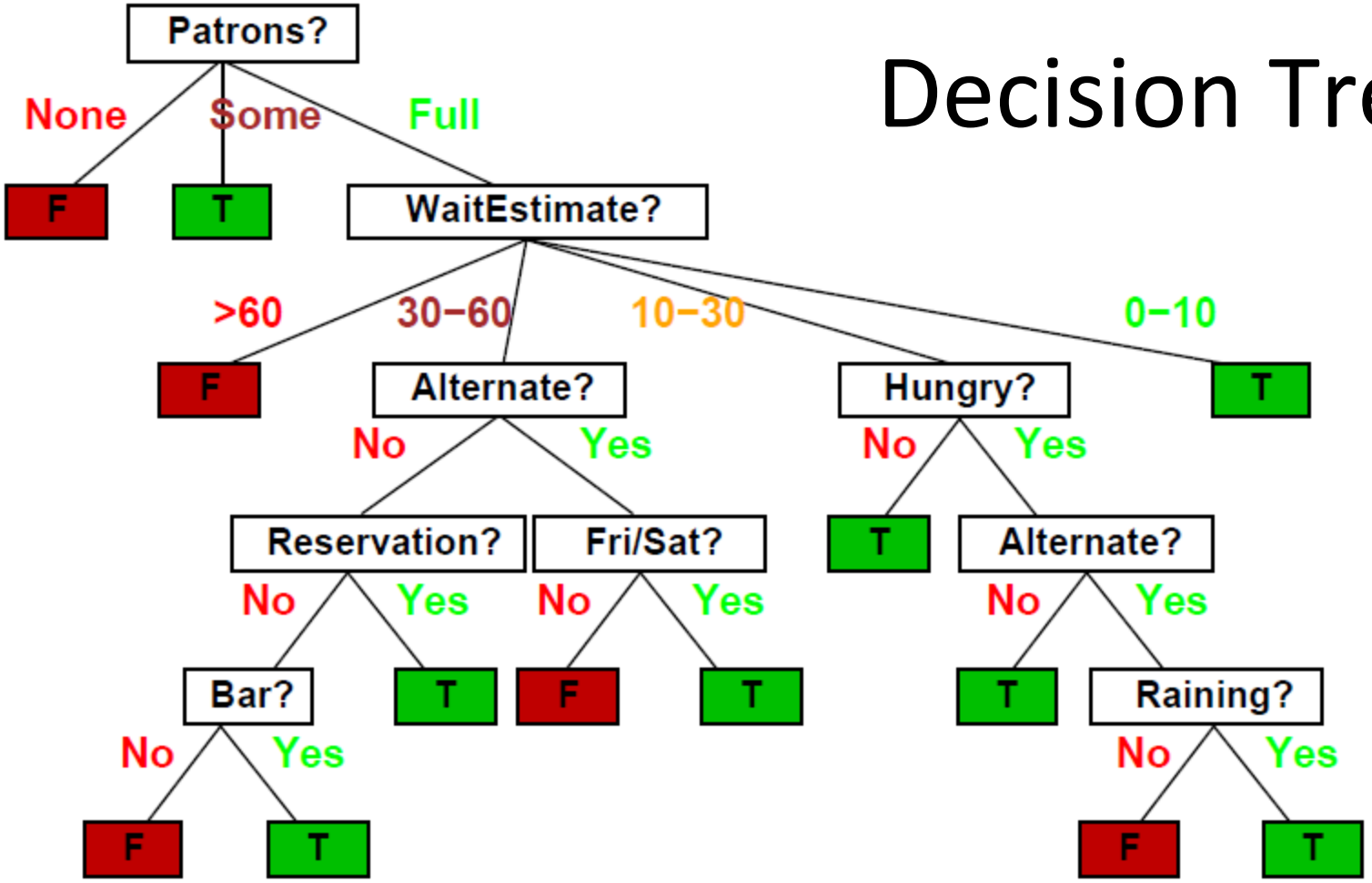


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

- First check: value of *Patrons*? **Full**.
- Where do we go next? To the right child.

Decision Trees

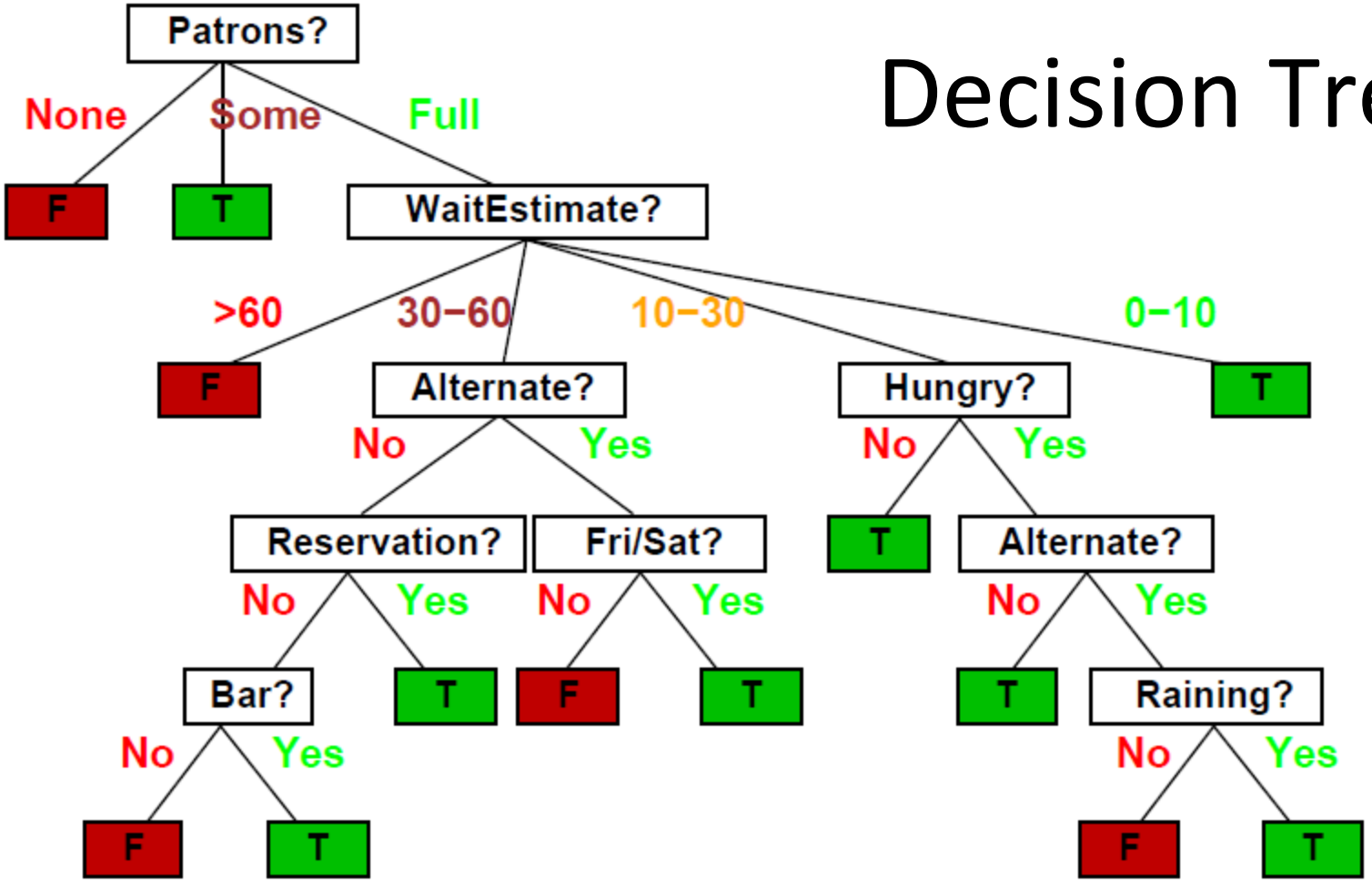


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

- Next check: ???

Decision Trees

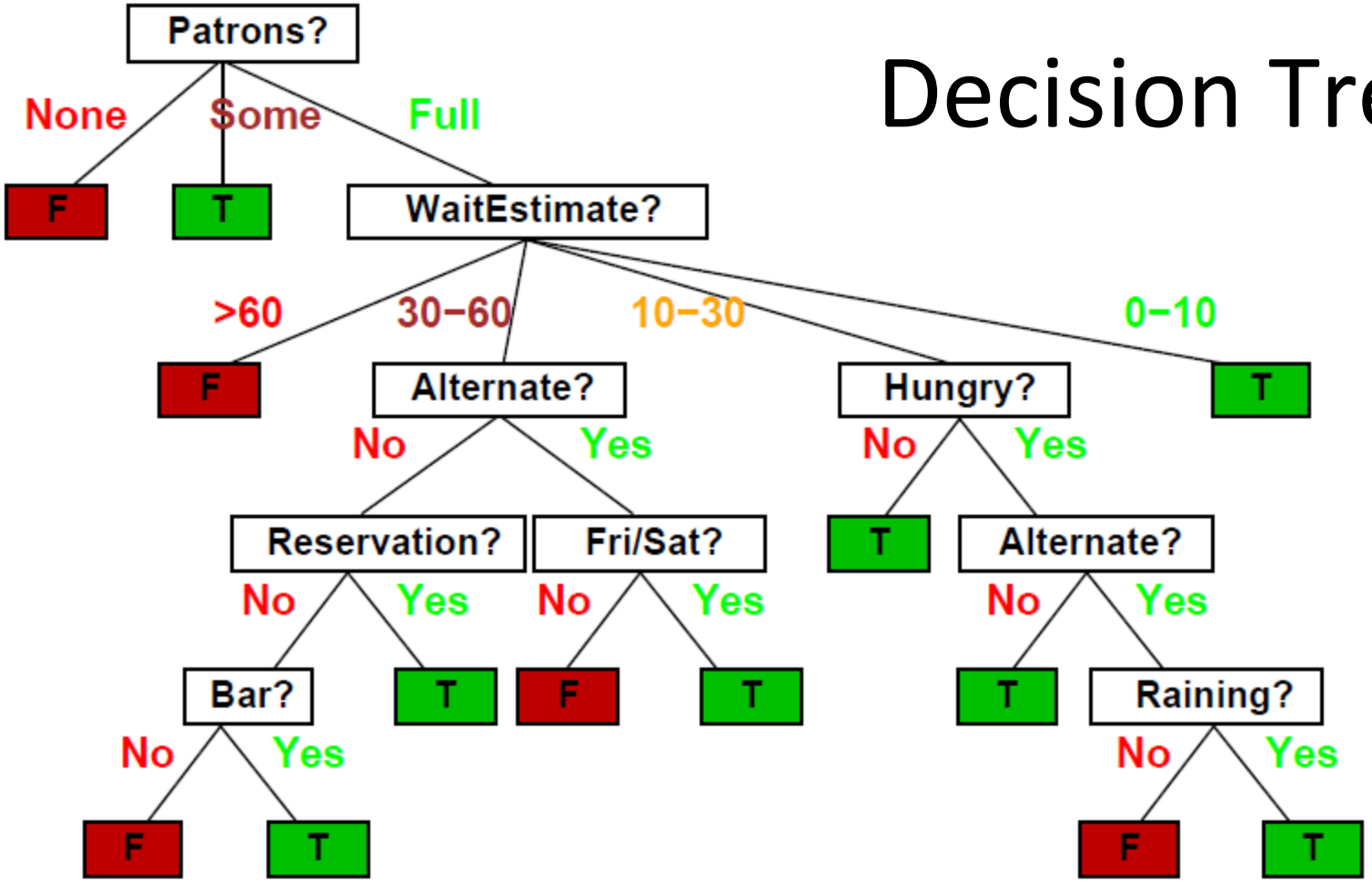


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

- Next check: value of *WaitEstimate*? **10-30**.
- Where do we go next? To the second-from-the-right child.

Decision Trees

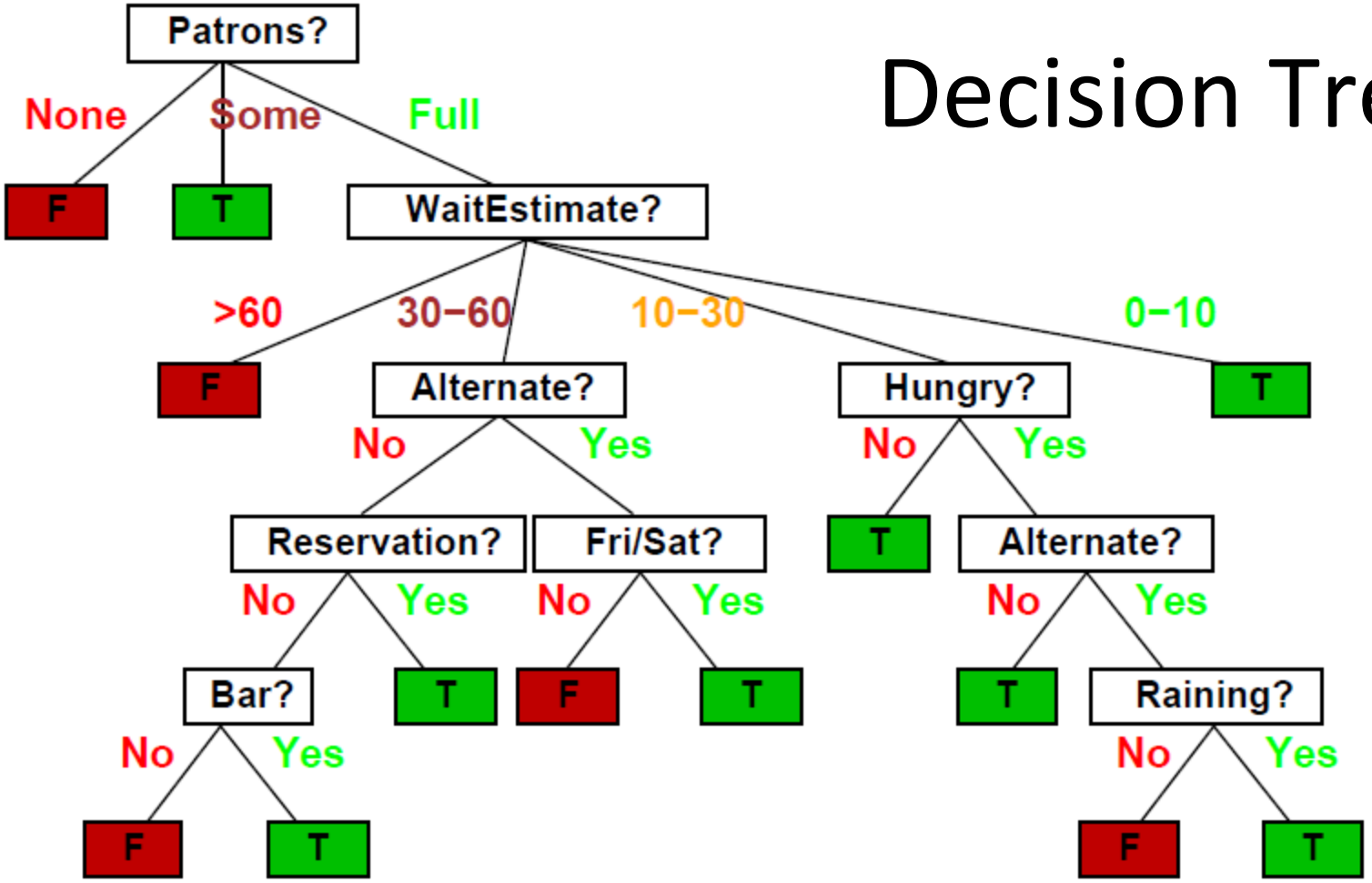


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

- Next check: value of *Hungry*? **False**.
- Where do we go next? To the left child.

Decision Trees

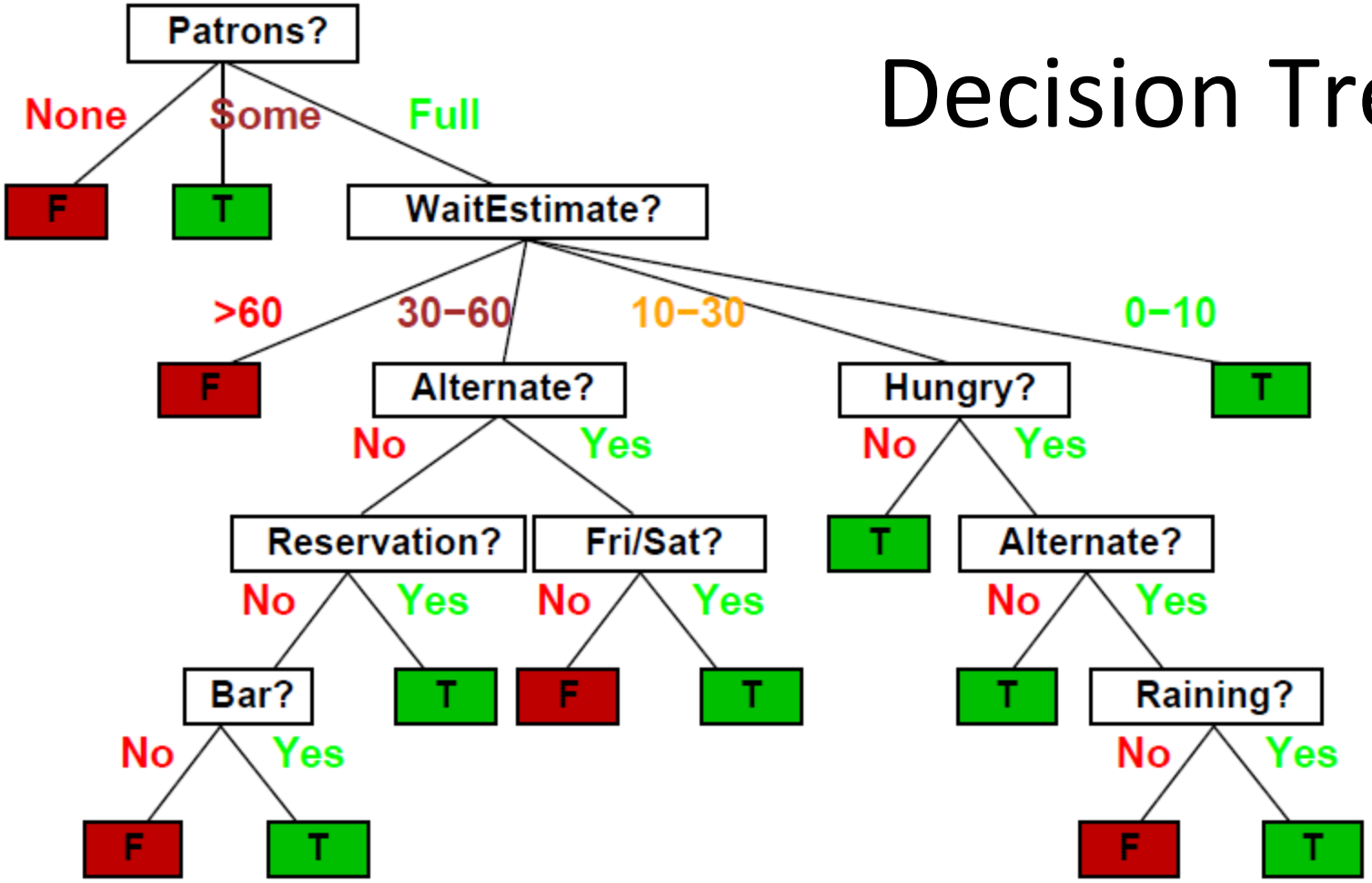


- What is the output of the decision tree on this pattern?

Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*
F	T	F	F	Full	$$	T	F	Thai	10-30

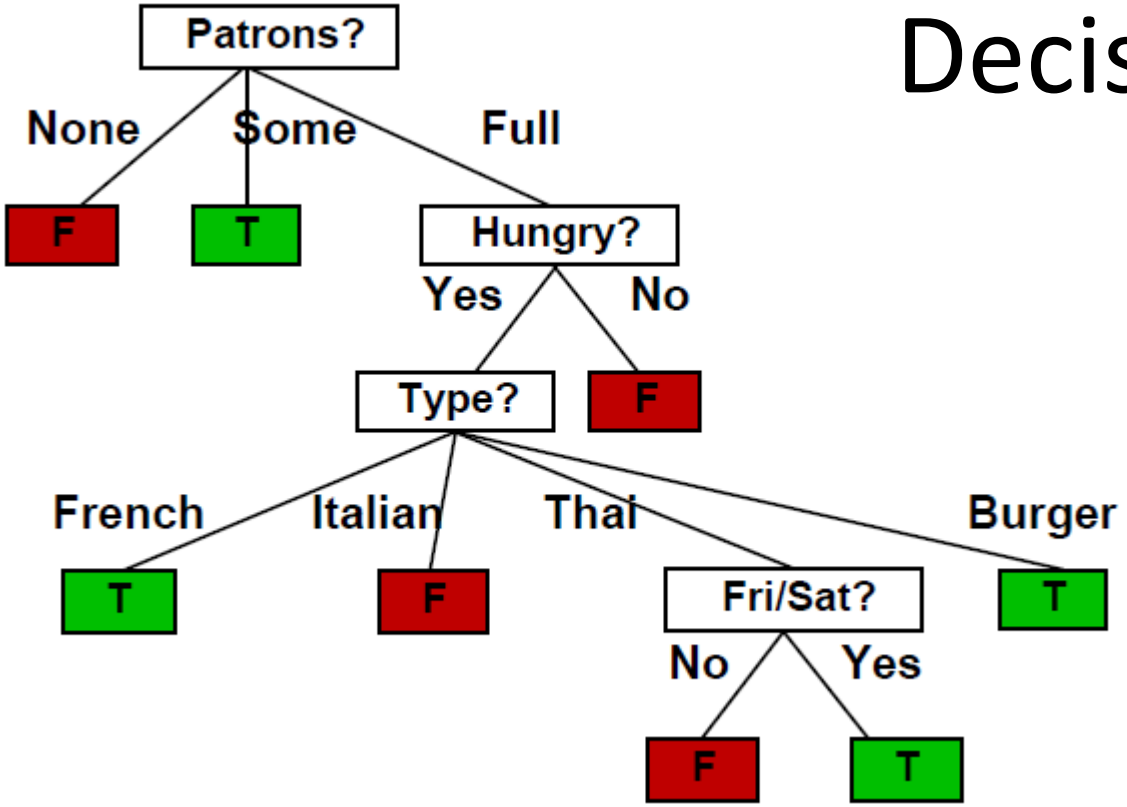
- We arrived at a leaf node.
- Output: **will wait**.

Decision Trees



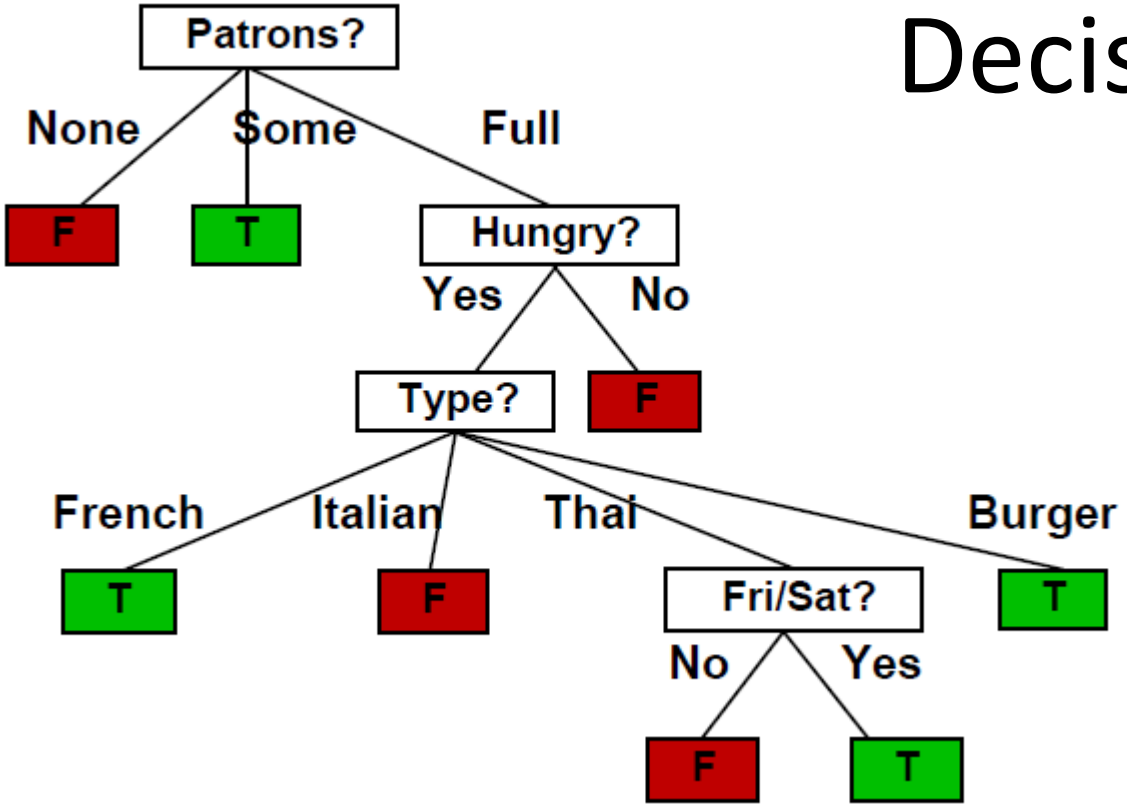
- At this point, it should be clear how to apply a decision tree to classify a pattern.
- Obviously, there are lots and lots of different decision trees that we can come up with.

Decision Trees



- At this point, it should be clear how to apply a decision tree to classify a pattern.
- Obviously, there are lots and lots of different decision trees that we can come up with.
- Here is a different decision tree.

Decision Trees



- The natural question is: how can we construct a good decision tree?
- The next slides address that question.

Decision Tree Learning

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- DTL is the pseudocode that the textbook provides for learning a decision tree.
- Looks simple and short, but (as in TT-Entails?) there are a lot of details we should look into.

Decision Tree Learning

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Notice that the function is recursive (the line of the recursive call is highlighted).
- This function builds the entire tree, and its recursive calls build each individual subtree, and each individual leaf node.

DTL: Arguments

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- *examples*: A set of training examples. Remember, each training example is a pair, consisting of a pattern and a class label.
- *attributes*: A list of attributes that we can choose to test.
- *default*: A default class to output if no better choice is available (details later).

DTL: Return Type

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- The function returns a decision tree.
- However, notice the highlighted line, that says that if there are no examples, we should return *default*. What does that mean?

DTL: Return Type

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- The function returns a decision tree.
- However, notice the highlighted line, that says that if there are no examples, we should return *default*. What does that mean?
- It means we should return a leaf node, that outputs class *default.*

DTL: Return Type

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- This is the only reason we have a *default* argument to the DTL function.
- If there are no examples, obviously we need to create a leaf node.
- The *default* argument tells us what class to store at the leaf node.

DTL: Base Cases

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- First base case: the *examples* are empty. As discussed before, we return a leaf node with output class *default*.
- Second base case: all *examples* have the same class. We just return a leaf node with that class as its output.

DTL: Base Cases

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Third base case: *attributes* is empty. We have run out of questions to ask.
- In this case, we have to return a leaf node. What class should we store there?

DTL: Base Cases

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Third base case: *attributes* is empty. We have run out of questions to ask.
- In this case, we have to return a leaf node, with output class MODE(*examples*).
- MODE is an auxiliary function, that returns the most common class among the examples.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- The highlighted code shows the recursive case.
- The first thing we do, is call CHOOSE-ATTRIBUTE.
- CHOOSE-ATTRIBUTE is an auxiliary function that chooses the attribute we should check at this node.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- We will talk A LOT about the CHOOSE-ATTRIBUTE function, a bit later.
- For now, just accept that this function will do its job and choose an attribute, which we store at variable *best*.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Here we create the *tree* that we are going to return.
- We store in that tree (probably in some member variable) the fact that we will be testing the attribute *best*.
- Next???

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Next, we need to create the children of that *tree*.
- How? With recursive calls to DTL, with appropriate arguments.
- How many children do we create?

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Next, we need to create the children of that *tree*.
- How? With recursive calls to DTL, with appropriate arguments.
- How many children do we create? As many as the values of attribute *best*.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- The highlighted loop creates the children (the subtrees of *tree*).
- Each iteration creates one child.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Remember, each child (subtree) corresponds to a value v_i of *best*.
- To create that child, we need to call DTL with appropriate values for *examples, attributes,* and *default*.
- What examples should we use?

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Remember, each child (subtree) corresponds to a value v_i of *best*.
- To create that child, we need to call DTL with appropriate values for *examples, attributes,* and *default*.
- What examples should we use? The subset of examples where *best* = v_i.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

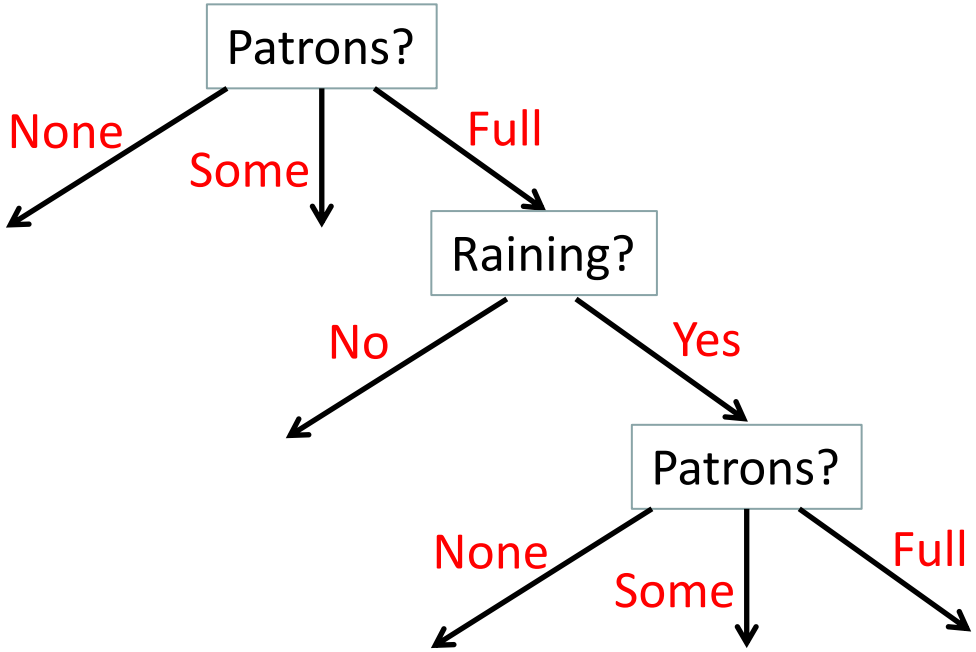
- What attributes do we use?

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

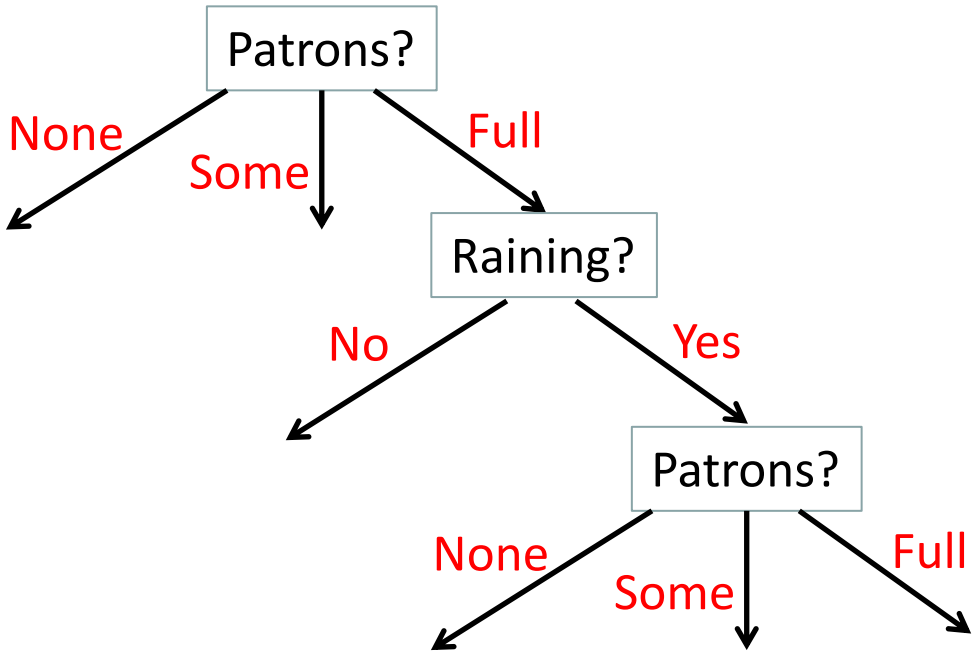
- What attributes do we use? Everything in the *attributes* variable, except for *best*.
- Why are we leaving *best* out?
- What happens if we use the same attribute twice in a path?

Using an Attribute Twice in a Path



- What happens the second time we use Patrons? as the attribute in this example?

Using an Attribute Twice in a Path



- What happens the second time we use Patrons? as the attribute in this example?
- It is useless. All patterns getting to this node already have Patrons? = **Full**.
- Therefore, all patterns will go to the right subtree.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Finally, in our recursive call to DTL, what should be the value of *default*?
- This will be used as output for a leaf node, if $examples_i$ are empty.
- First of all, why would $examples_i$ ever be empty?

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Finally, in our recursive call to DTL, what should be the value of *default*?
- This will be used as output for a leaf node, if $examples_i$ are empty.
- First of all, why would $examples_i$ ever be empty?
- It may just happen that no training examples have $best = v_i$.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- So, what should be the value of *default*?

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- So, what should be the value of *default*?
- It should be MODE(*examples*): the most frequent class among the examples.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Once the recursive call to DTL has returned a subtree, we add that to our tree.

DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

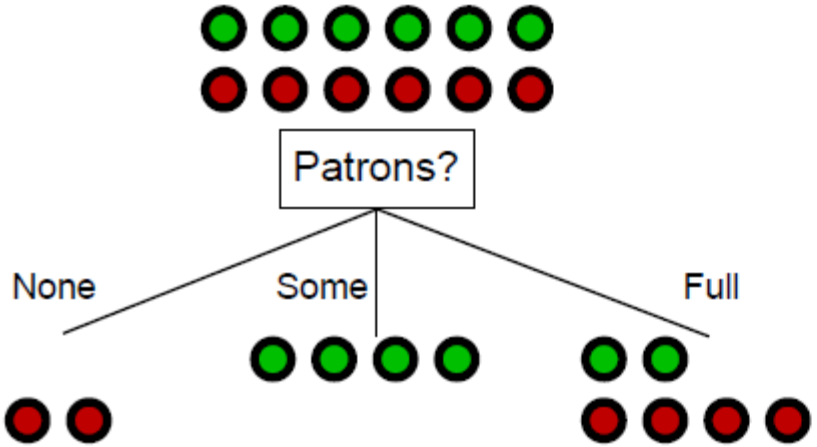
- Finally, once all recursive calls to DTL have returned, we return the tree we have created.
- Are we done?

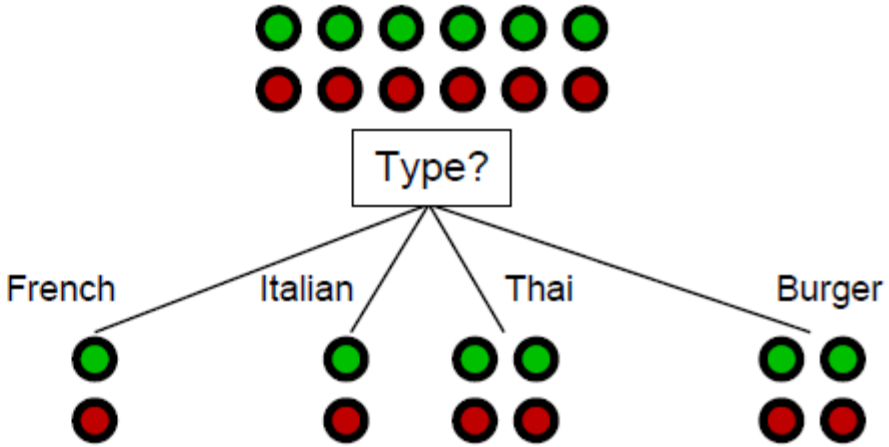
DTL: Recursive Case

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Finally, once all recursive calls to DTL have returned, we return the tree we have created.
- Are we done? Almost, except that we still need to talk how CHOOSE-ATTRIBUTE works.

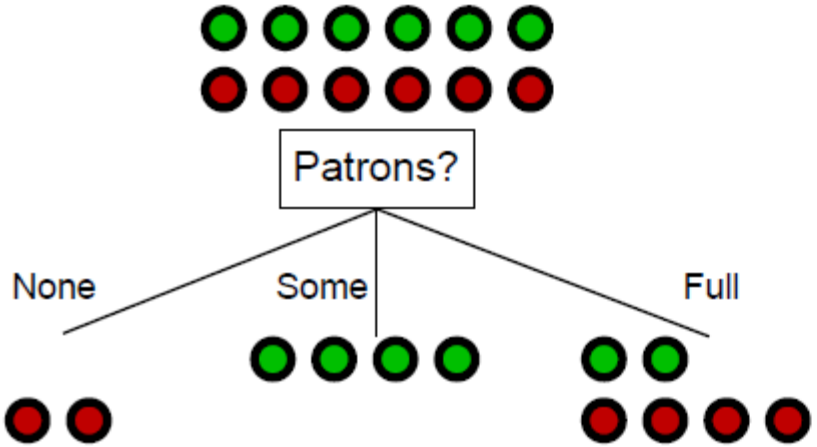
Choosing an Attribute

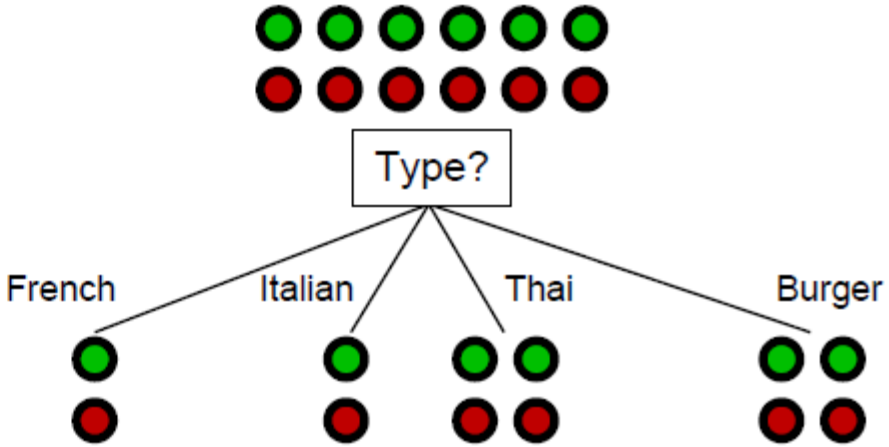




- Here we see two different attributes chosen at the root of the decision tree.
  - At each node, green stands for a training example that waited, red stands for a training example that did not wait.
- Which attribute seems better (more useful) to you?

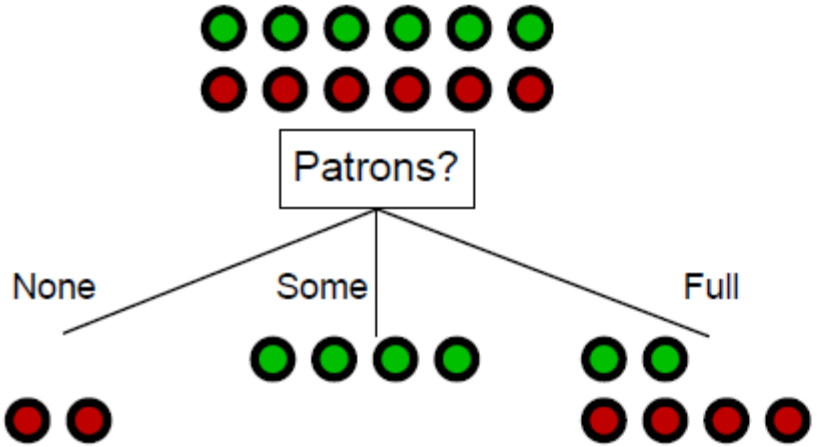
Choosing an Attribute

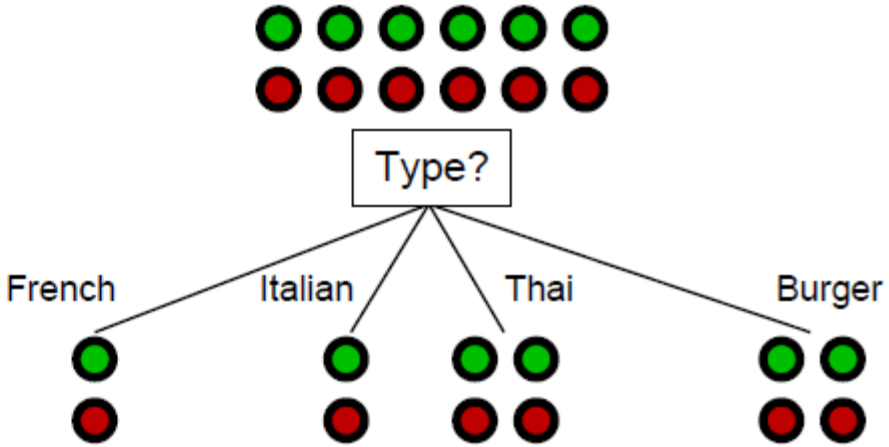




- Here we see two different attributes chosen at the root of the decision tree.
  - At each node, green stands for a training example that waited, red stands for a training example that did not wait.
- Which attribute seems better (more useful) to you?
- The Patrons attribute is more useful, because it separates better the greens from the reds.

Choosing an Attribute





- How can we quantify how well an attribute separates training examples from different classes?
- We need to define two new quantities:
  - Entropy.
  - Information gain.
- These quantities are computed with specific formulas.
- Information gain will be used to choose the best attribute.

Entropy – Two-Class Example

- Suppose that we have a set X of training examples.
  - K_1 examples have class label A.
  - K_2 examples have class label B.
- Let $K = K_1 + K_2$.
- Then the entropy of the set X depends only on the two ratios: $\frac{K_1}{K}$ and $\frac{K_2}{K}$. The entropy H is defined as:

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}\right) = -\frac{K_1}{K}\log_2\frac{K_1}{K} - \frac{K_2}{K}\log_2\frac{K_2}{K}$$

- Note: logarithms in this discussion are always base 2.

Entropy – General Formula

- In the general case:
- Suppose that we have a set X of training examples.
- Suppose there are N different class labels L_1, …, L_N.
  - K_1 examples have class label L_1.
  - K_2 examples have class label L_2.
  - In general, K_i examples have class label L_i.
- Let $K = K_1 + K_2 + \ldots + K_N$.
- Then the entropy H of the set X is given by this formula:

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

Making Sense of Entropy

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- If this is the first time you see the definition of entropy, it probably does not look very intuitive.
- We will look at several specific examples, so that you see how entropy behaves.
- The lowest possible entropy value is 0.
  - The lower the entropy is, the more uneven the distribution of classes is.
  - Zero entropy means all training examples have the same class.
- The highest possible entropy value is $\log_2 N$ (N=number of classes).
  - The higher the entropy is, the more even the distribution of classes is.
  - Entropy $\log_2 N$ means that the number of training examples for each class is equal. $K_1 = K_2 = \ldots = K_N$.

Example 1

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of two classes.
  - 200 examples have class label A.
  - 200 examples have class label B.
- What is the entropy of X?

Example 1

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of two classes.
  - 200 examples have class label A.
  - 200 examples have class label B.
- What is the entropy of X?

$$\mathrm{H}\left(\frac{200}{400}, \frac{200}{400}\right) = -\frac{200}{400} \log_2 \frac{200}{400} - \frac{200}{400} \log_2 \frac{200}{400}$$

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

$$= -0.5 * (-1) - 0.5 * (-1) = 1.$$

- The classes are evenly split.
- Therefore, H has the largest possible value: $\log_2 2 = 1$.

Example 2

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of two classes.
  - 20 examples have class label A.
  - 500 examples have class label B.
- What is the entropy of X?

$$\mathrm{H}\left(\frac{20}{520}, \frac{500}{520}\right) = -\frac{20}{520} \log_2 \frac{20}{520} - \frac{500}{520} \log_2 \frac{500}{520}$$

$$= -0.0385 \log_2 0.0385 - 0.9615 \log_2 0.9615$$

$$= -0.0385 * (-4.6990) - 0.9615 * (\text{-}0.0566) = 0.235.$$

- The classes are pretty unevenly split.
- Therefore, H has a smallish value, relatively close to 0.

Example 3

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of two classes.
  - 20 examples have class label A.
  - 5000 examples have class label B.
- What is the entropy of X?

$$\mathrm{H}\left(\frac{20}{5020}, \frac{5000}{5020}\right) = -\frac{20}{5020} \log_2 \frac{20}{5020} - \frac{5000}{5020} \log_2 \frac{5000}{5020}$$

$$= -0.0040 \log_2 0.0040 - 0.9960 \log_2 0.9960$$

$$= -0.0040 * (-7.9658) - 0.9960 * (-0.0058) = 0.038.$$

- The classes are even more unevenly split.
- Therefore, H is even smaller than it was in Example 2.

Example 4

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of two classes.
  - 0 examples have class label A.
  - 200 examples have class label B.
- What is the entropy of X?

$$\mathrm{H}\left(\frac{0}{200}, \frac{200}{200}\right) = -\frac{0}{200} \log_2 \frac{0}{200} - \frac{200}{200} \log_2 \frac{200}{200}$$

$$= -0 \log_2 0 - 1 \log_2 1$$

$$= -0 * \infty - 1 * 0 = -0 * \infty = ???.$$

- This is an interesting case. Who wins, 0 or infinity?

Example 4

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of two classes.
  - 0 examples have class label A.
  - 200 examples have class label B.
- What is the entropy of X?

$$\mathrm{H}\left(\frac{0}{200}, \frac{200}{200}\right) = -\frac{0}{200} \log_2 \frac{0}{200} - \frac{200}{200} \log_2 \frac{200}{200}$$

$$= -0 \log_2 0 - 1 \log_2 1$$

$$= -0 * \infty - 1 * 0 = -0 * \infty = 0.$$

- This is an interesting case. Who wins, 0 or infinity?
- We will skip the mathematical proof, but 0 wins.
- Makes sense, most uneven split possible → lowest H possible.

Example 5

$$\mathrm{H}\left(\frac{K_1}{K}, \frac{K_2}{K}, \ldots, \frac{K_N}{K}\right) = \sum_{i=1}^{N} -\frac{K_i}{K} \log_2 \frac{K_i}{K}$$

- We have a set X of training examples, of three classes.
  - 50 examples have class label A.
  - 15 examples have class label B.
  - 25 examples have class label B.
- What is the entropy of X?

$$\mathrm{H}\left(\frac{50}{90}, \frac{15}{90}, \frac{25}{90}\right) = -\frac{50}{90}\log_2\frac{50}{90} - \frac{15}{90}\log_2\frac{15}{90} - \frac{25}{90}\log_2\frac{25}{90}$$

$$= -0.556\log_2 0.556 - 0.167\log_2 0.167 - 0.278\log_2 0.278$$

$$= -0.556 * (-0.847) - 0.167 * (-2.582) - 0.278 * (-1.847)$$

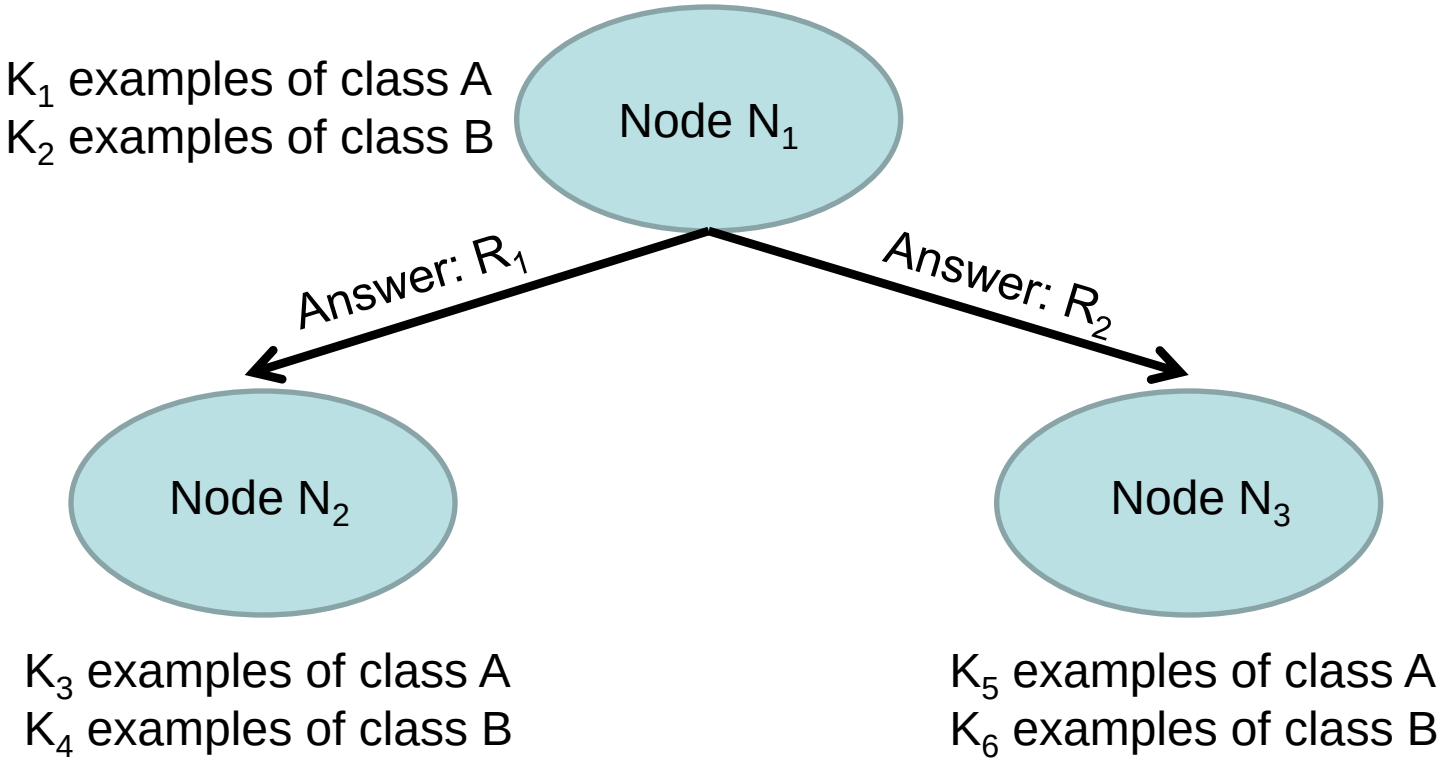
$$= 1.416$$

- Since we have three classes, the entropy can be greater than 1.

Information Gain – Example with Two Classes, Two Values

- Suppose that we have a set X of training examples.
  - K_1 examples have class label A.
  - K_2 examples have class label B.
- Let $K = K_1 + K_2$.
- Suppose that we have an attribute Q_1 with only two possible answers: R_1 and R_2.
- K_3 examples of class A and K_4 examples of class B give answer R_1.
- K_5 examples of class A and K_6 examples of class B give answer R_2.
- Obviously, $K_3 + K_5 = K_1$, $K_4 + K_6 = K_2$

Information Gain – Example with Two Classes, Two Values

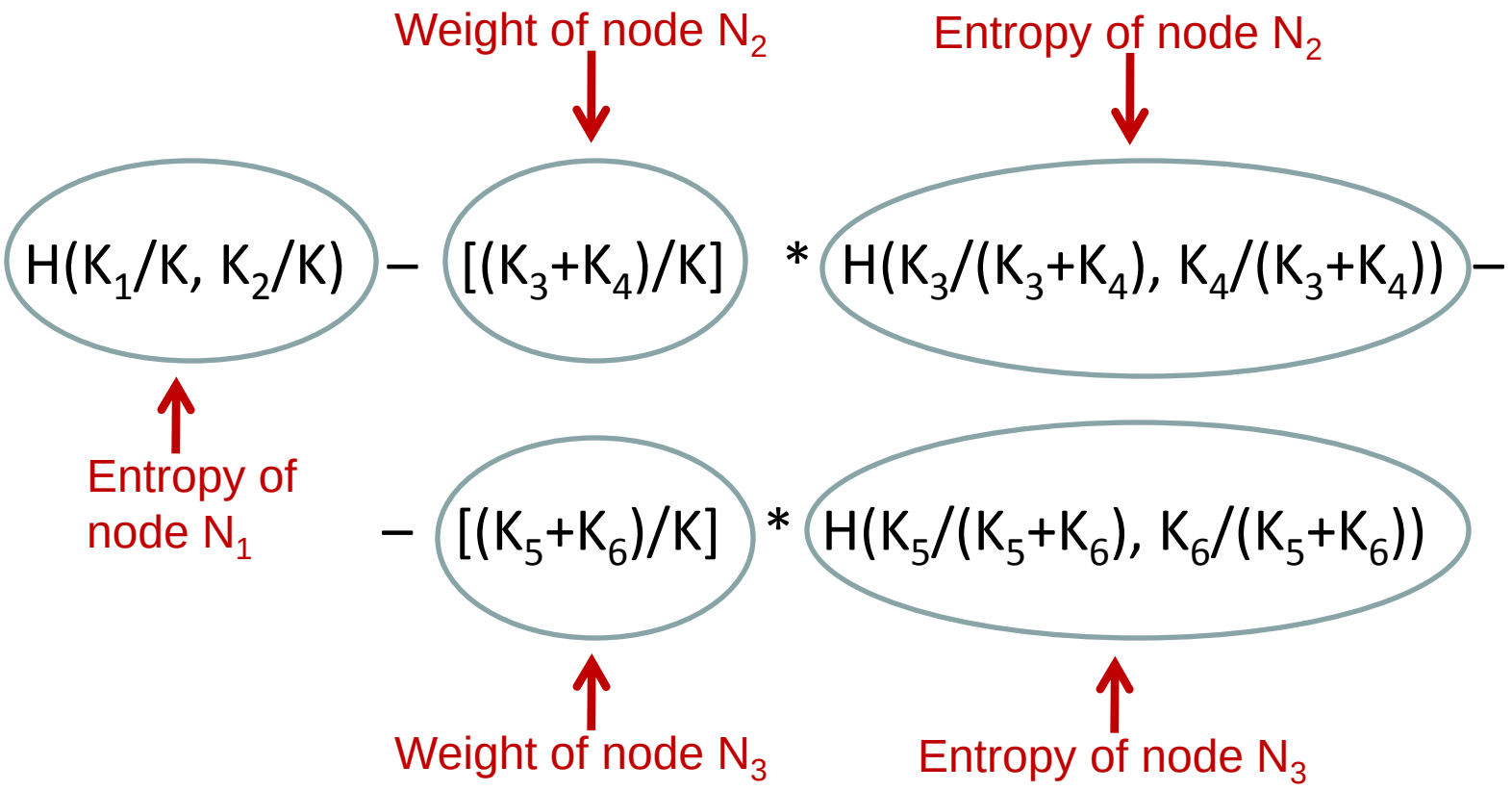


- Information gain of question Q at node N_1 =

Entropy at N_1 – weighted average of entropies at N_2 and N_3

Information Gain – Example with Two Classes, Two Values

- Information gain of question Q at node N_1 =



Information Gain – General Formula

- Setting:
  - We have a parent node R, with a set E of K training examples.
  - We choose a certain attribute T at that node.
  - Attribute T has L values.
  - Based on their values on attribute T, the training examples split into L subsets: E_1, …, E_L.
  - Each E_i has K_i training examples.
- Then, the information gain of attribute T at node R is:

$$I(E, L) = H(E) - \sum_{i=1}^{L} \frac{K_i}{K} H(E_i)$$

- Note: H(E) and $H(E_i)$ refer to the entropy of the training examples at each set, based on their class labels.

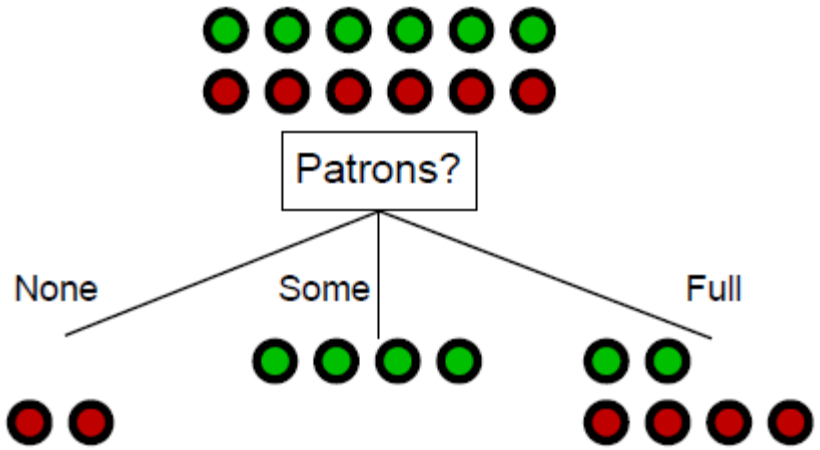
Information Gain – Behavior

$$I(E, L) = H(E) - \sum_{i=1}^{L} \frac{K_i}{K} H(E_i)$$

- Overall, we like attributes that yield as high information gain as possible.
- To get that, we want the entropy of the subsets E_i to be small.
  - This happens when the attribute splits the training examples nicely, so that different classes get concentrated on different subsets E_i.

Information Gain – Example 1

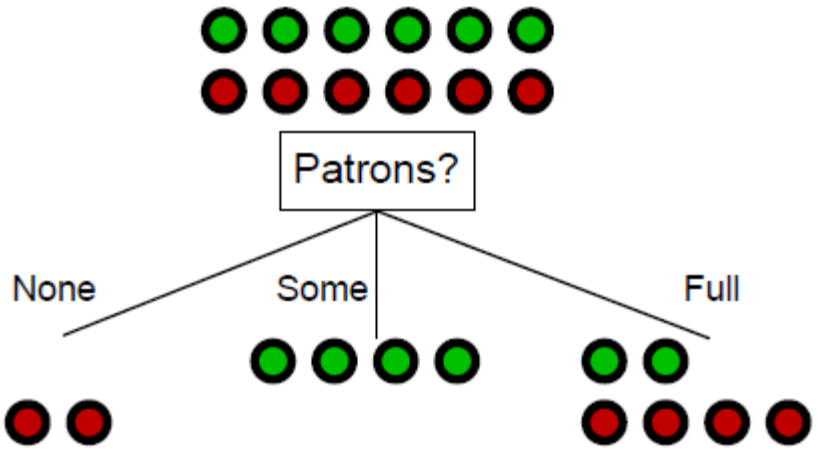
$$I(E,L) = H(E) - \sum_{i=1}^{L} \frac{K_i}{K} H(E_i)$$



- What is the information gain in this example?
  - H(E) = ???
  - $H(E_1)$ = ???
  - $H(E_2)$ = ???
  - $H(E_3)$ = ???

Information Gain – Example 1

$$I(E, L) = H(E) - \sum_{i=1}^{L} \frac{K_i}{K} H(E_i)$$



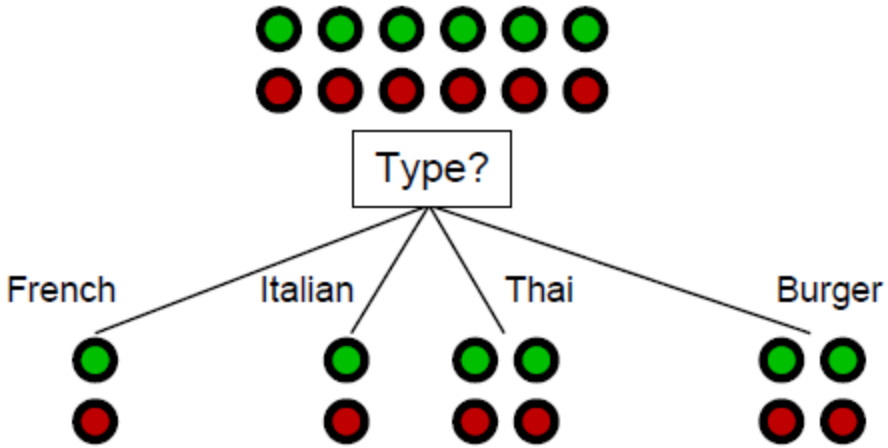
- What is the information gain in this example?
  - H(E) = 1, since classes are evenly split at the top.
  - $H(E_1) = 0$, since we only have **will not wait** cases on the left child.
  - $H(E_2) = 0$, since we only have **will wait** cases on the middle child.
  - $H(E_3) = -\frac{2}{6}\log_2\left(\frac{2}{6}\right) - \frac{4}{6}\log_2\left(\frac{4}{6}\right) = 0.9183.$

$$I(E, \text{Patrons}) = H(E) - \frac{2}{12} * H(E_1) - \frac{4}{12} * H(E_2) - \frac{6}{12} * H(E_3)$$

$$= 1 - \frac{2}{12} * 0 - \frac{4}{12} * 0 - \frac{6}{12} * 0.9183 = 0.5409$$

Information Gain – Example 2

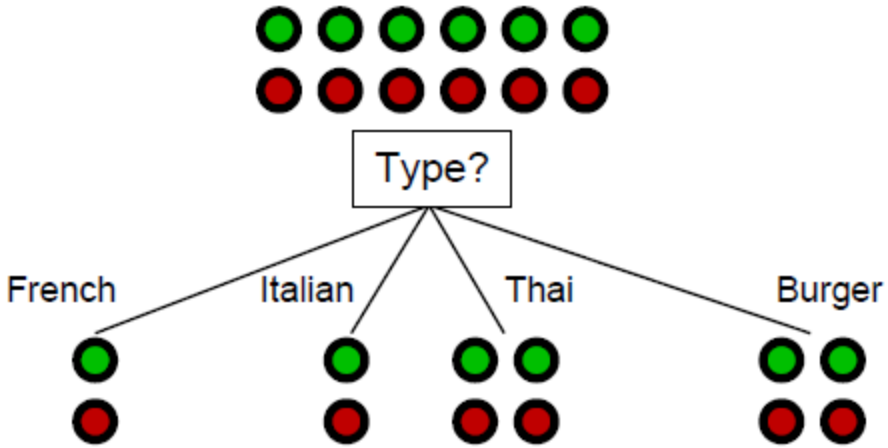
$$I(E, L) = H(E) - \sum_{i=1}^{L} \frac{K_i}{K} H(E_i)$$



- What is the information gain in this example?
  - H(E) = ???
  - $H(E_1)$ = ???
  - $H(E_2)$ = ???
  - $H(E_3)$ = ???
  - $H(E_4)$ = ???

Information Gain – Example 2

$$I(E, L) = H(E) - \sum_{i=1}^{L} \frac{K_i}{K} H(E_i)$$



- What is the information gain in this example?
  - H(E) = 1, since classes are evenly split.
  - $H(E_1) = H(E_2) = H(E_3) = H(E_4) = 1$, classes at all children are also evenly split.

$$I(E, \text{Type}) =$$

$$= H(E) - \frac{2}{12} * H(E_1) - \frac{2}{12} * H(E_2) - \frac{4}{12} * H(E_3) - \frac{4}{12} * H(E4)$$

$$= 1 - \frac{2}{12} * 1 - \frac{2}{12} * 1 - \frac{4}{12} * 1 - \frac{4}{12} * 1 = 0$$

CHOOSE-ATTRIBUTE

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same class then return the class
    else if attributes is empty then return MODE(examples)
    else
        best = CHOOSE-ATTRIBUTE(attributes, examples)
        tree = a new decision tree with root test best
        for each value v_i of best do
            examples_i = {elements of examples with best = v_i}
            subtree = DTL(examples_i, attributes - best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- We can now finally specify what CHOOSE-ATTRIBUTE does:
- CHOOSE-ATTRIBUTE returns the attribute that achieves the highest information gain on the given examples.

Recap on Decision Trees

- Decision trees are a popular pattern recognition methods.
- Learning a decision tree is done by recursively:
  - Picking an attribute at the root.
  - Sending the training examples to different children of the root, based on their values on the chosen attribute.
  - Learning each of the children.
- Choosing attributes is based on information gain.
  - We prefer attributes that concentrate different classes on different children.