CSC 361: Artificial Intelligence

Learning

The Importance of Learning

- An agent is learning if it **improves** its performance upon **observing** the world.
- Why is it important to learn ?
  - The designer can not anticipate the all the situations in which the agent be. For example, a robot navigating a maze.
  - The designer can not anticipate all changes over time. For example, stock market.
  - Sometimes the designers have no idea how to program the solution themselves. For example: face recognition.

Types of Learning

- In order to learn, the agent needs to observe the world → **feedback.**
- The different types of feedback determine the different types of learning:
  - Supervised learning
  - Unsupervised learning
  - Semi-supervised learning
  - Reinforcement learning

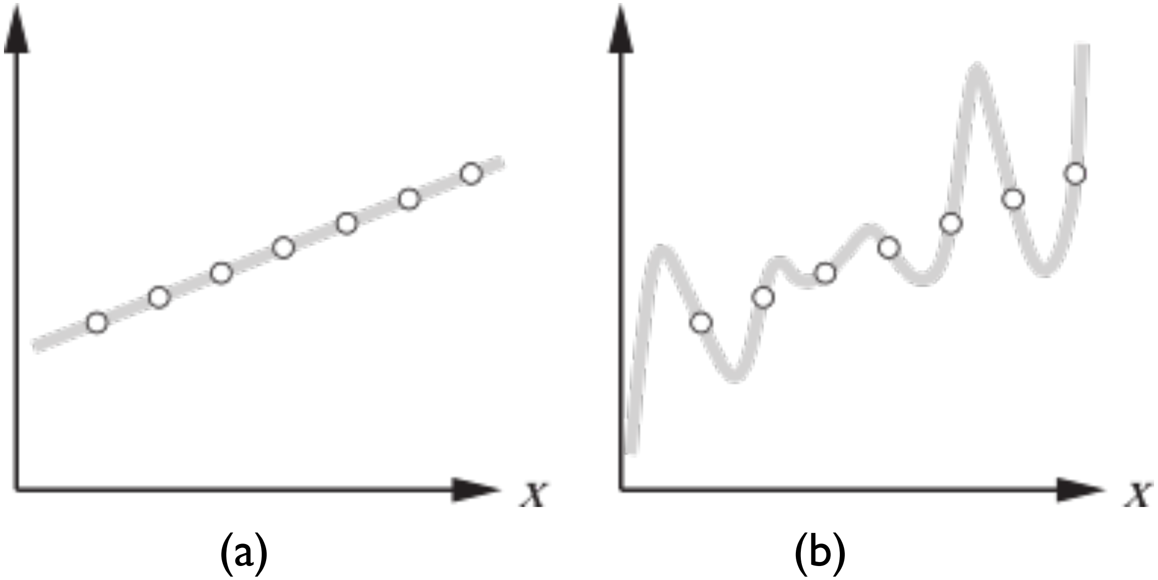
Types of Learning

- **Supervised learning:** The agent observes a set of input-output examples (**labeled examples)** and learns a map from inputs to outputs.
  - **Classification**: output is discrete (e.g., spam email)
  - **Regression**: output is real-valued (e.g., stock market)
- **Unsupervised learning:** No explicit feedback is given, only the inputs (**unlabeled examples**). The agent learns patterns in the input.
  - **Clustering**: grouping data into K groups. (e.g. clustering images of fish into different species)

Supervised Learning

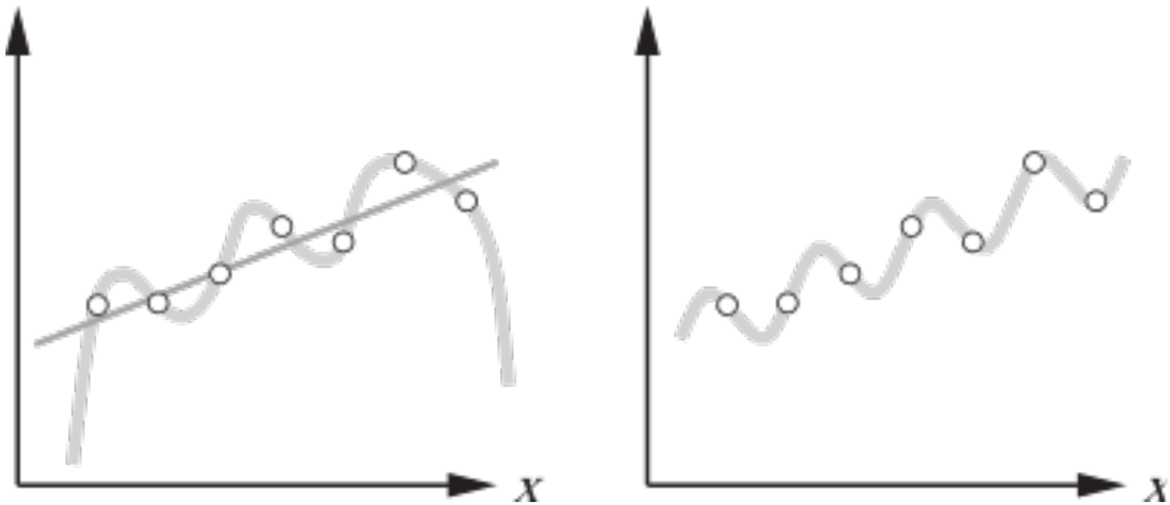
- Given a **training set** of N example input-output pairs:
  (x_1, y_1), (x_2, y_2), … (x_N, y_N),
  where, $y_j = f(x_j)$, where f is unknown function, the goal is to find a function **h** that **approximates** f.
- The function **h** is called a **hypothesis**.
- How to measure the accuracy of **h**?
  - We give a **test set** of examples, which is different from the training set.
  - The hypothesis **generalizes** well if it correctly predicts the output for the test set.

How to Choose the Hypothesis ?



- First, select the **hypothesis space**: in this case, the set of polynomials.
- (a): The line is **consistent** with the data.
- (b): The high-degree polynomial is also consistent. With the data.
- **Ockham's razor**: Choose the simplest hypothesis which is consistent with the data.

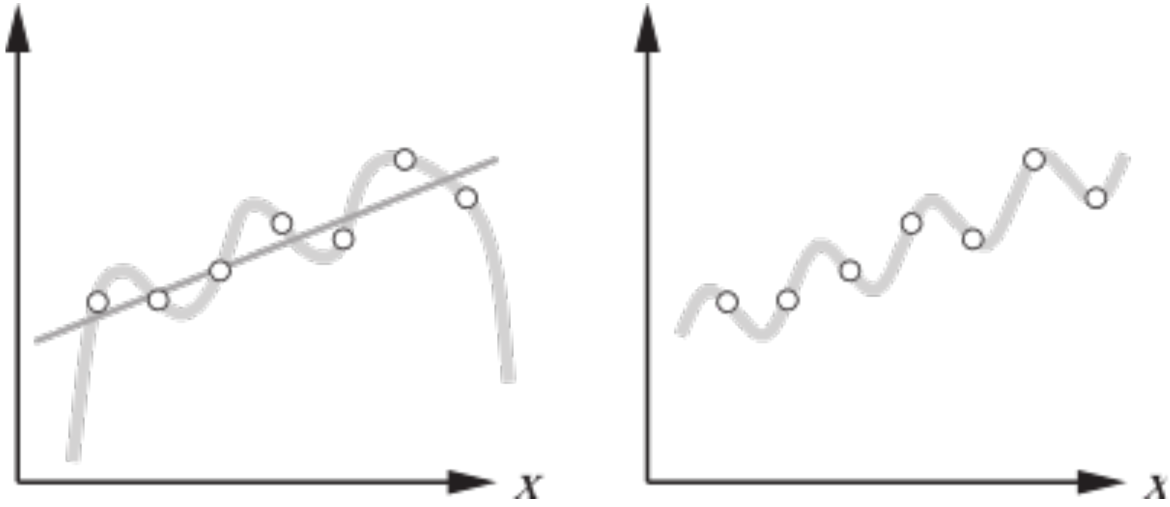
How to Choose the Hypothesis ?



(c) (d)

- (c): Do we choose the line or the 6-degree polynomial?
  - The line detects a pattern and will **generalize** well.
  - The 6-degree polynomial does not detect any pattern.
  - Choose the hypothesis that generalizes well, even if it is not consistent with the data.

How to Choose the Hypothesis ?



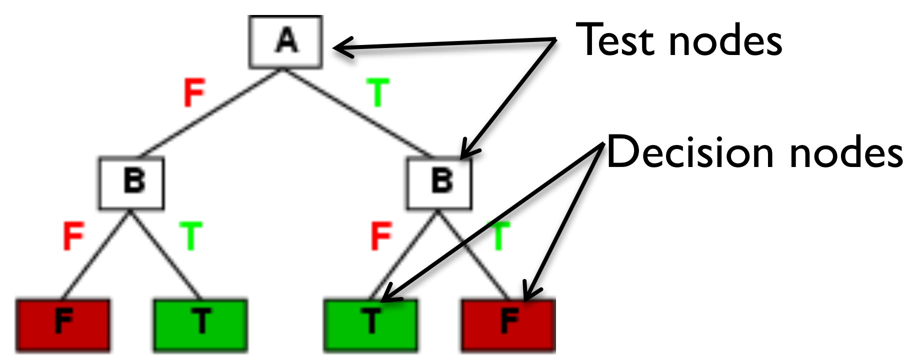
(c) (d)

- (d): a x + b + c sin(x) is consistent with the data → The choice of the hypothesis space is important.
  - The learning problem is **realizable** if the hypothesis space contains the true function (we can not know this of course, because f is unknown).
  - Complex hypothesis space → better hypothesis but complex search.
  - Simple hypothesis space → simple search, but less good hypothesis.

Decision Trees

- A **decision tree** represents a function that has multiple inputs but a single output.
  - We focus on discrete input and Boolean output (**Boolean classification**)
- A decision tree reaches the decision by a set of tests on the **attributes** (the inputs). Thus the internal nodes are the tests and the leaf nodes are the decisions.
- Example:

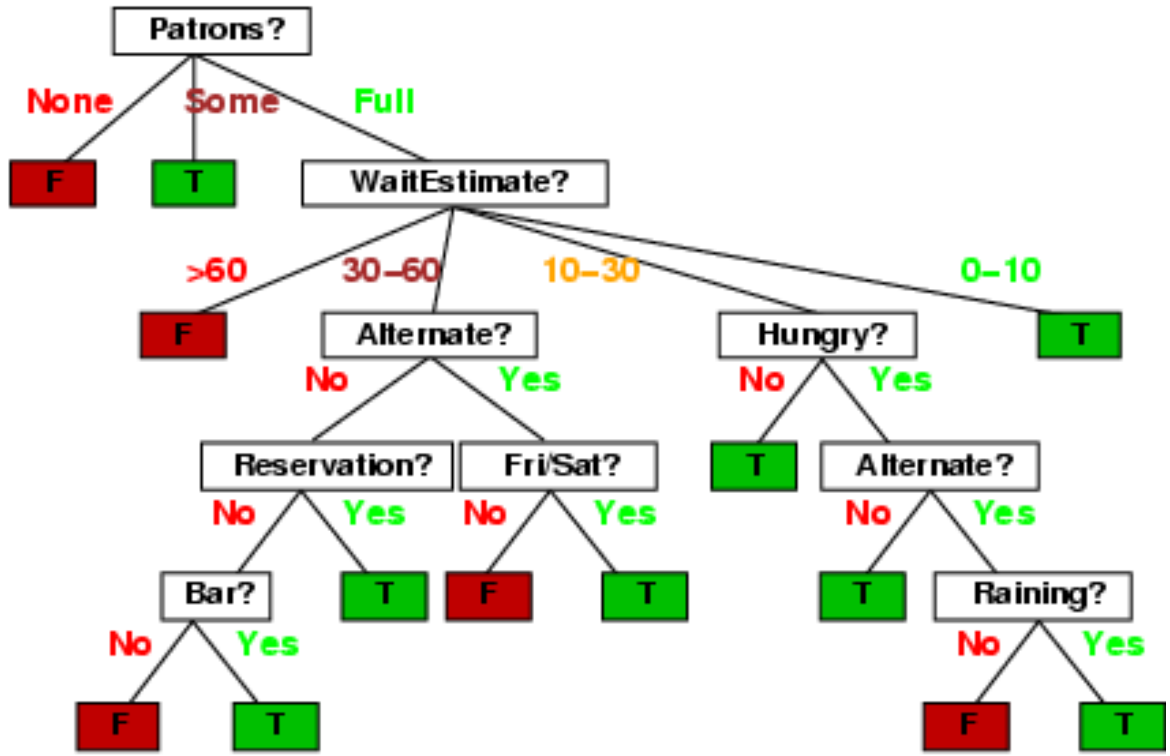
A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Decision Trees

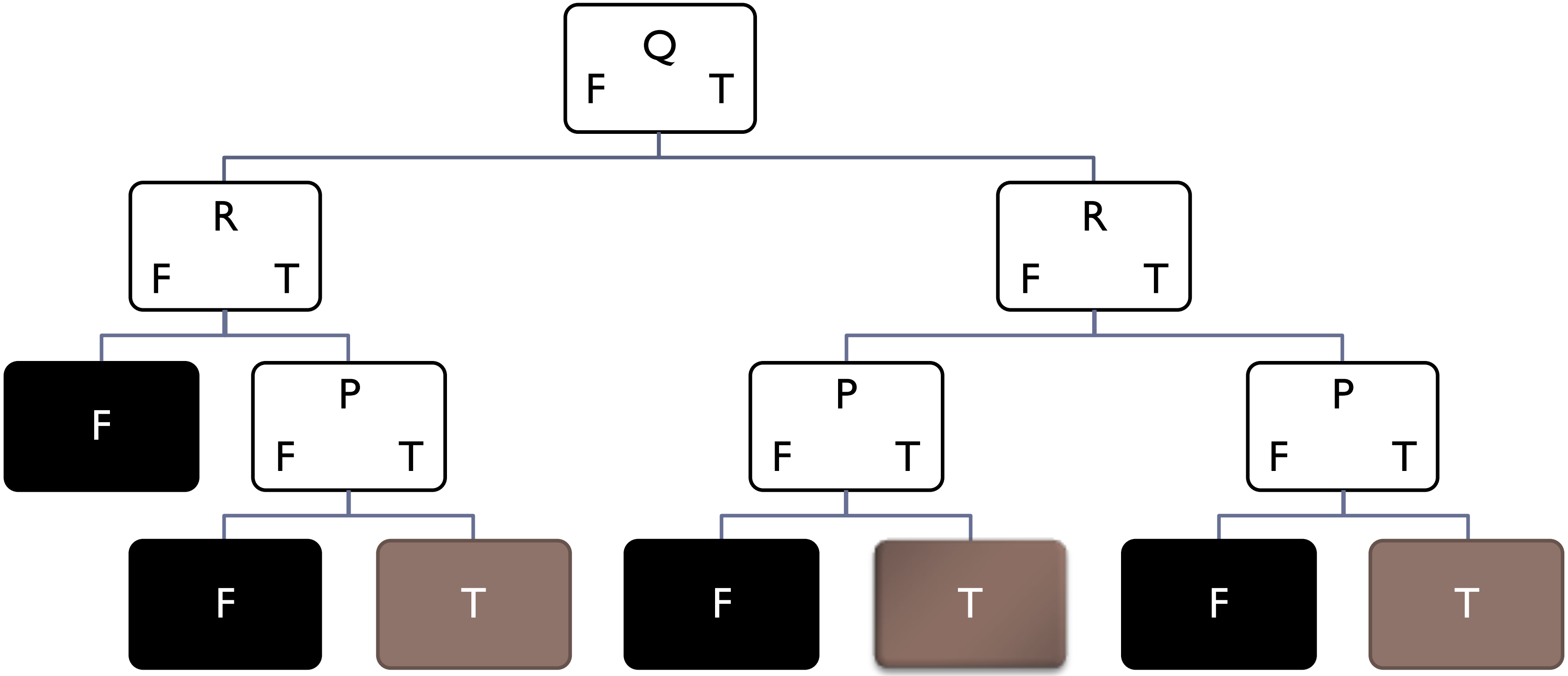
- A more complex example: deciding to wait at a restaurant:
- The attributes :
  1. **Alternate**: whether there is a suitable alternative restaurant nearby.
  2. **Bar**: whether the restaurant has a comfortable bar area to wait in.
  3. **Fri I Sat**: true on Fridays and Saturdays.
  4. **Hungry**: whether we are hungry.
  5. **Patrons**: how many people are in the restaurant (values are None, Some, and Full).
  6. **Price**: the restaurant's price range ($, $$, $$$).
  7. **Raining**: whether it is raining outside.
  8. **Reservation**: whether we made a reservation.
  9. **Type**: the kind of restaurant (French, Italian, Thai, or burger).
  10. **WaitEstimate**: the wait estimated by the host (0-10 minutes, 10-30, 30-60, or >60).

Decision Trees



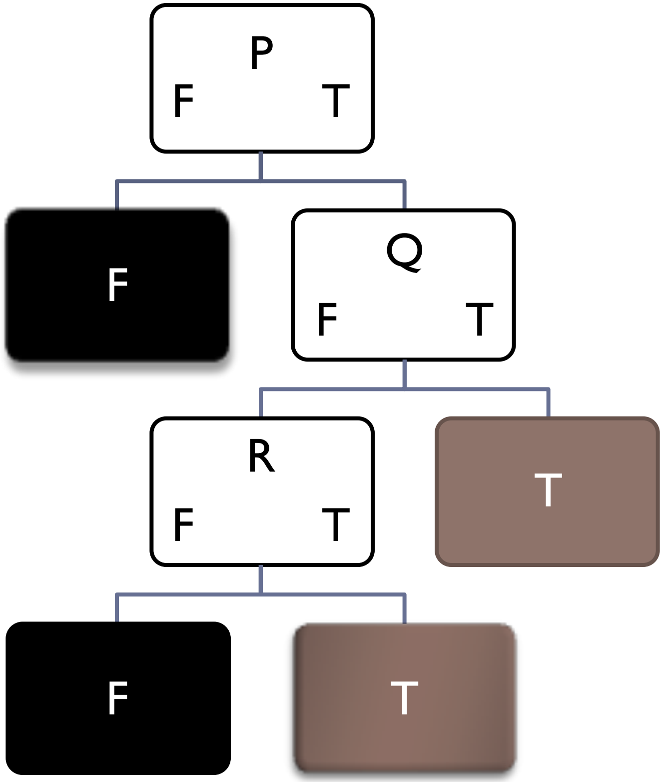
- This is the real function.
- Our goal is to learn this function from examples.

Decision Trees



A decision tree for the function: $P \wedge (Q \vee R)$.
The order of the attributes: Q, R, P

Decision Trees



Smaller number of nodes → The order is important

A decision tree for the function: P ∧ (Q ∨ R).
The order of the attributes: P, Q, R

Learning Decision Trees: Example 1

Training set for P ∧ (Q ∨ R)

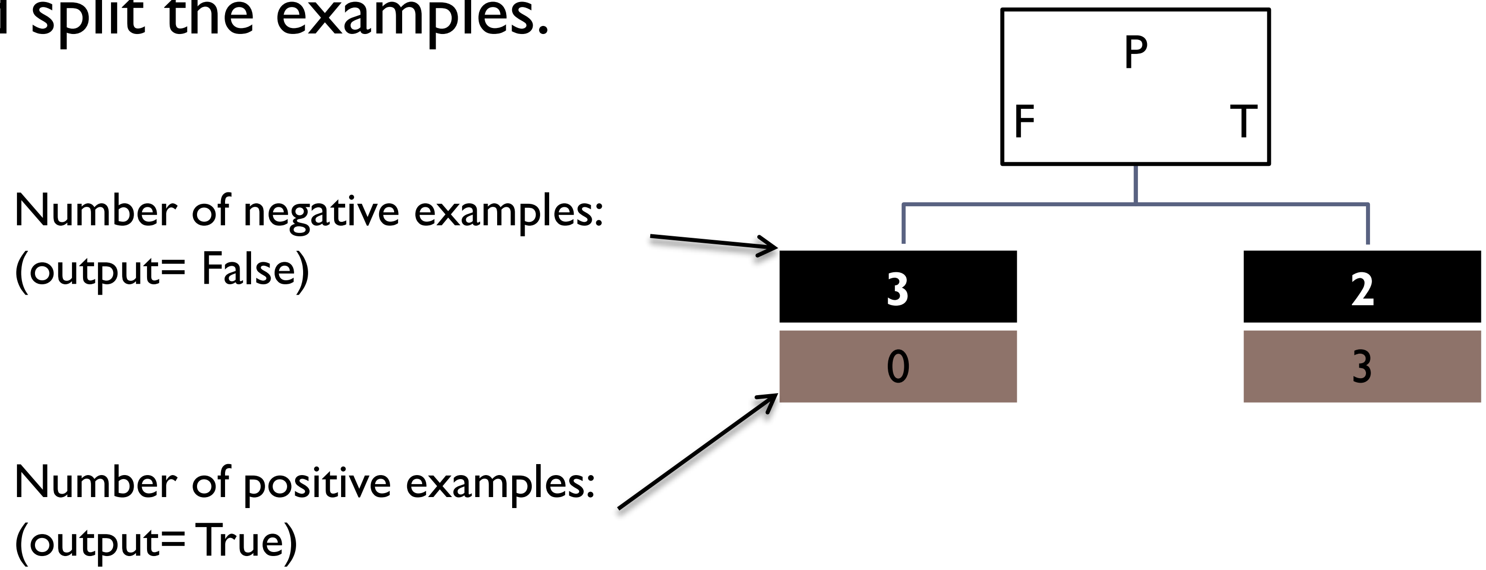
Example	P	Q	R	Output
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	0	1
8	1	1	0	0

Noise

Notice that some combinations of inputs do not appear

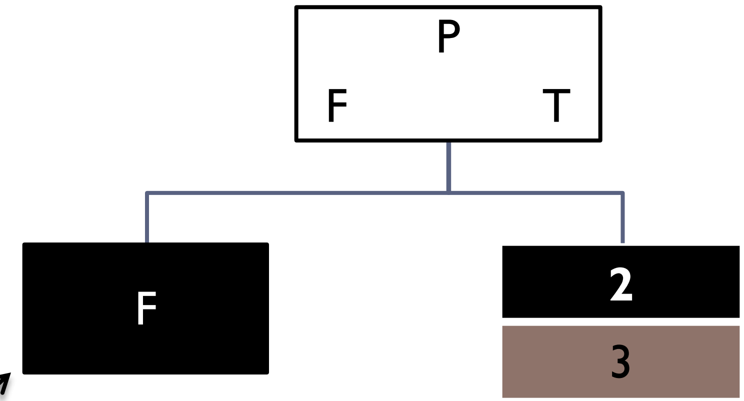
Learning Decision Trees: Example 1

- Choose the most important attribute (how?): in this case it is P, and split the examples.



Learning Decision Trees: Example1

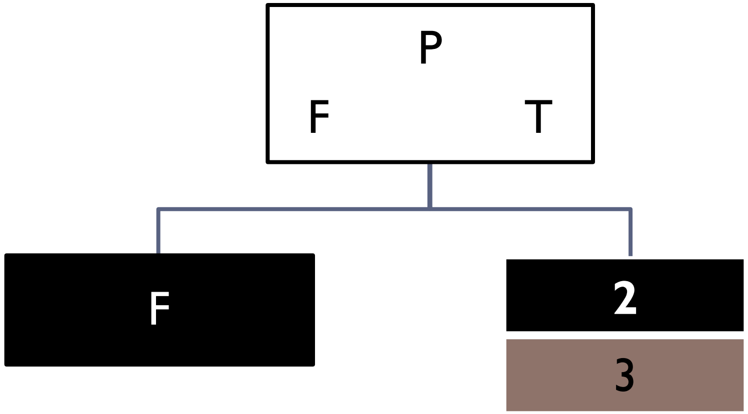
- When P is False all the examples have the same classification (all false) → We stop and make a decision node with the value False.



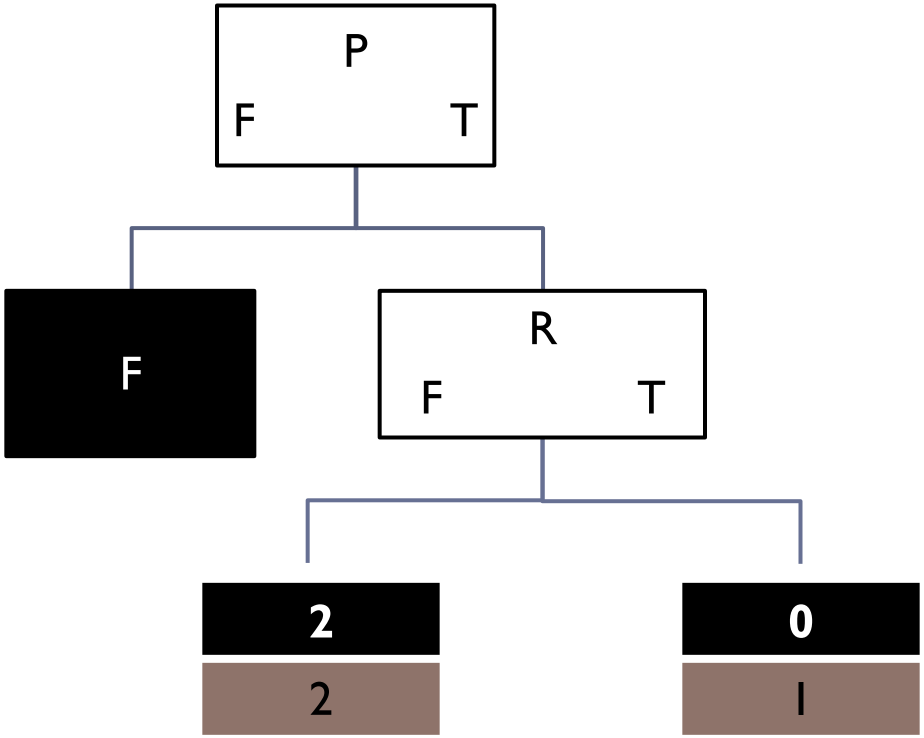
All examples are negative

Learning Decision Trees: Example 1

- For the node P=True, we have both positive and negative examples, so we choose an attribute (the most important: how ?). In this case it R

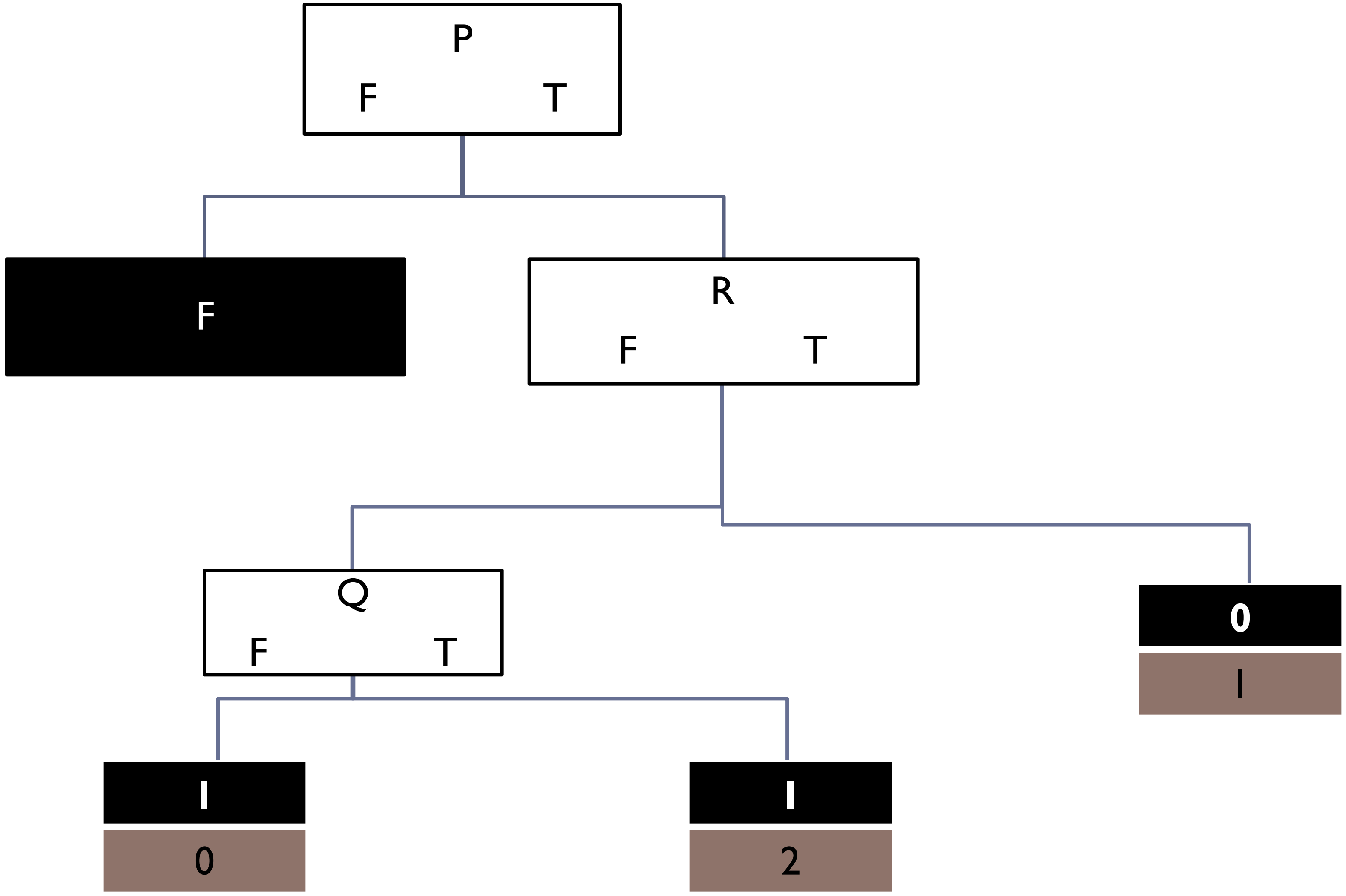


Learning Decision Trees: Example 1

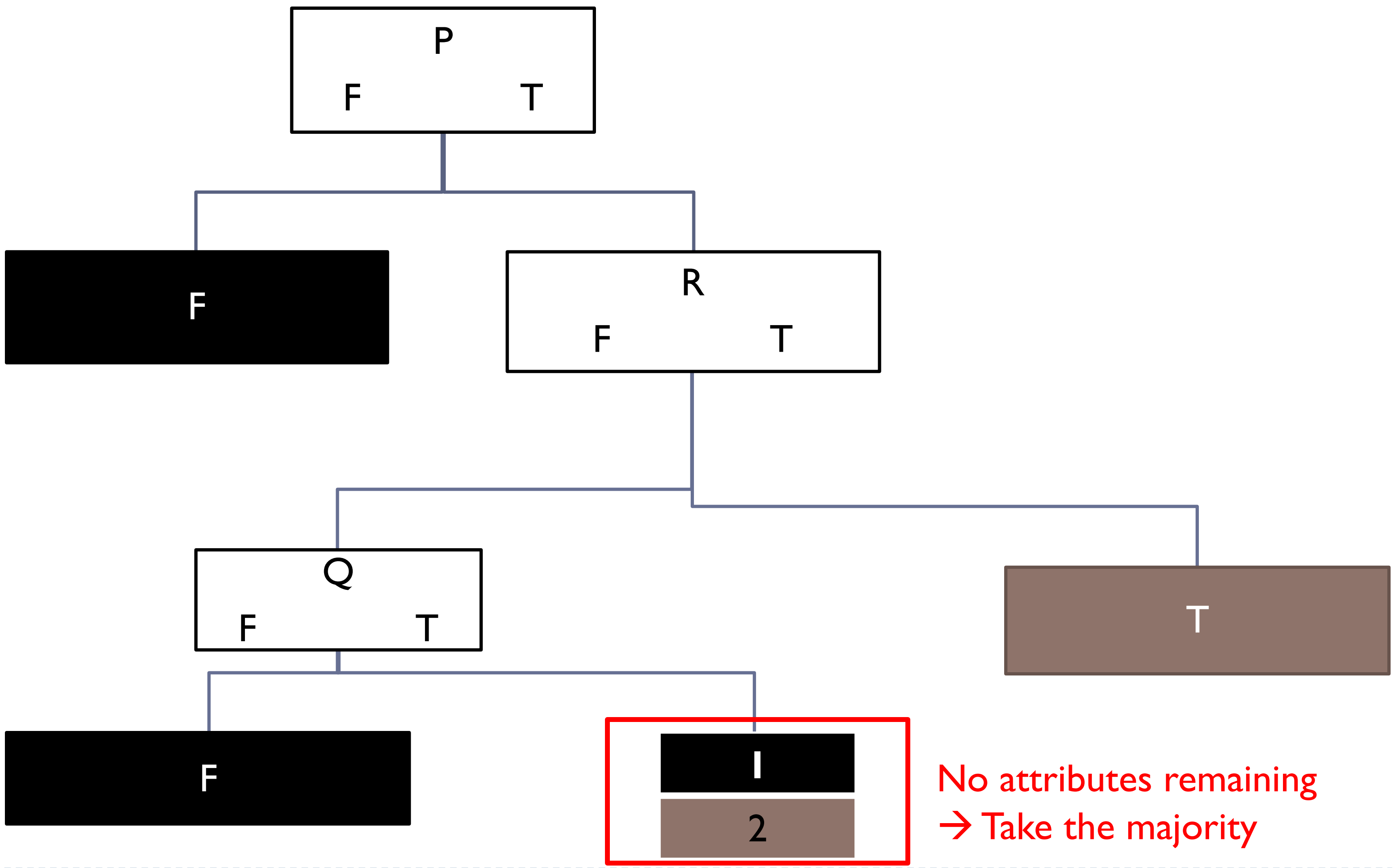


- For the node R=False, we have both positive and negative examples, so we choose an attribute: Q

Learning Decision Trees: Example 1

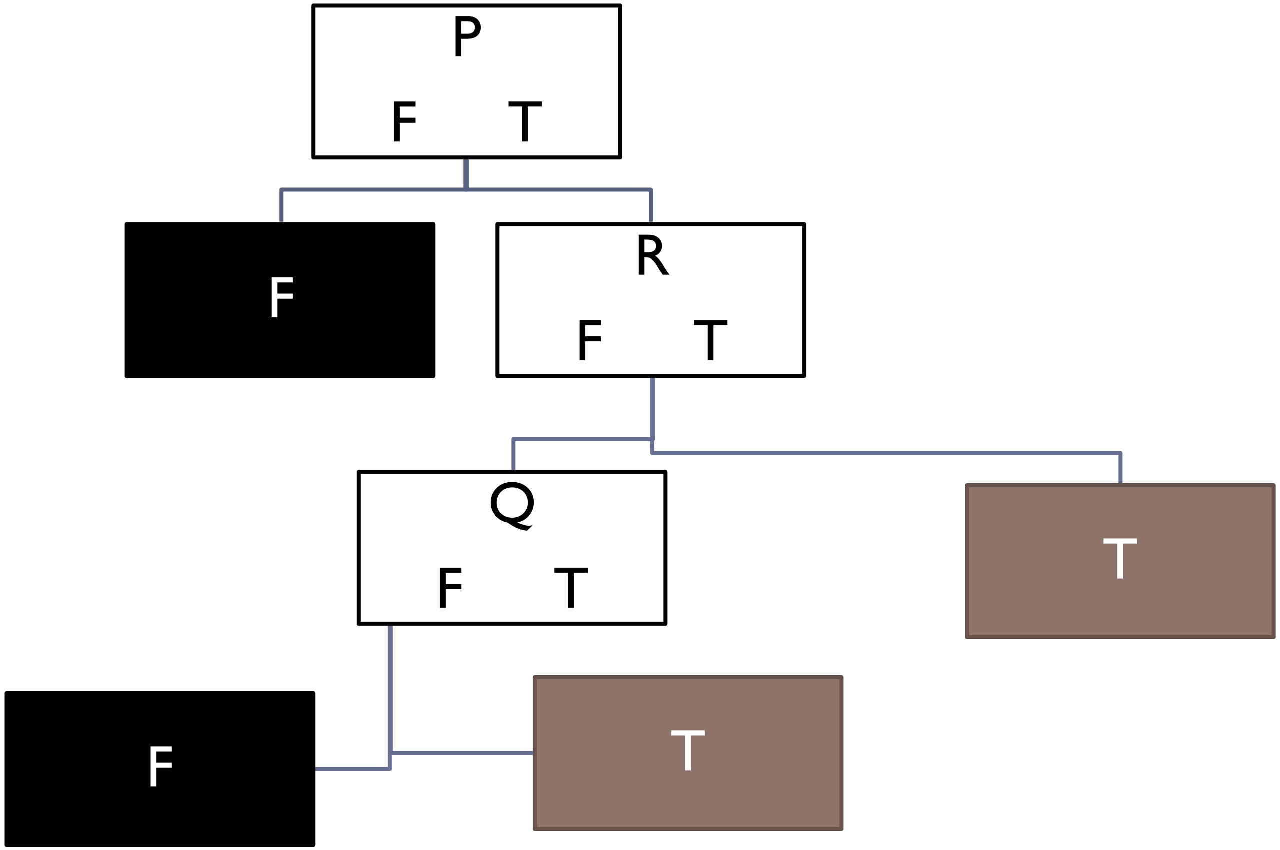


Learning Decision Trees: Example 1



No attributes remaining
→ Take the majority

Learning Decision Trees: Example 1



We obtained the true function in this case , but not always the case

Learning Decision Trees: Example 2

Training set for P ∧ (Q ∨ R)

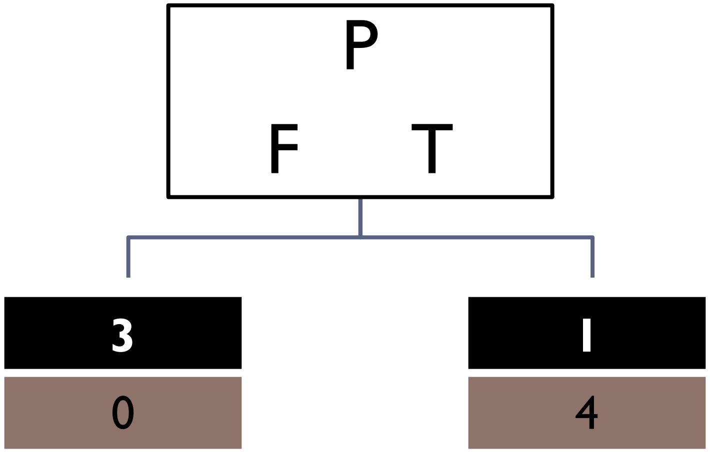
Example	P	Q	R	P ∧ (Q ∨ R)
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	1	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	0	1
8	1	1	0	0

Noise

Notice that some combinations of inputs do not appear

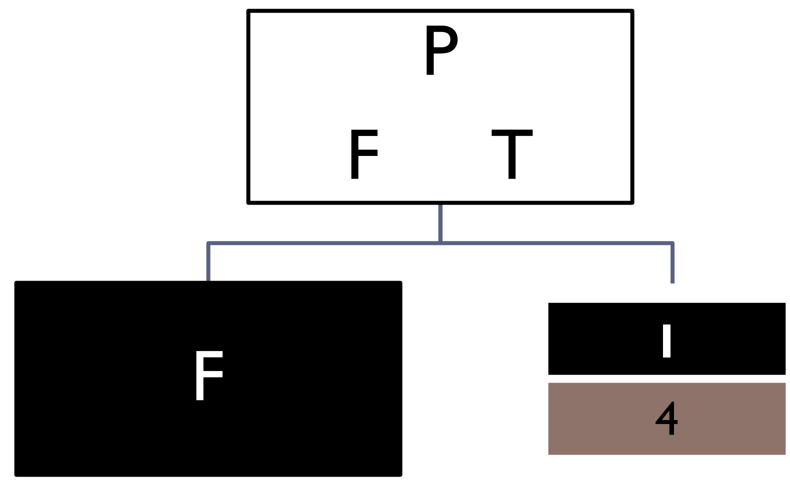
Learning Decision Trees: Example 2

- Choose the most important attribute (how?): in this case it is P, and split the examples.



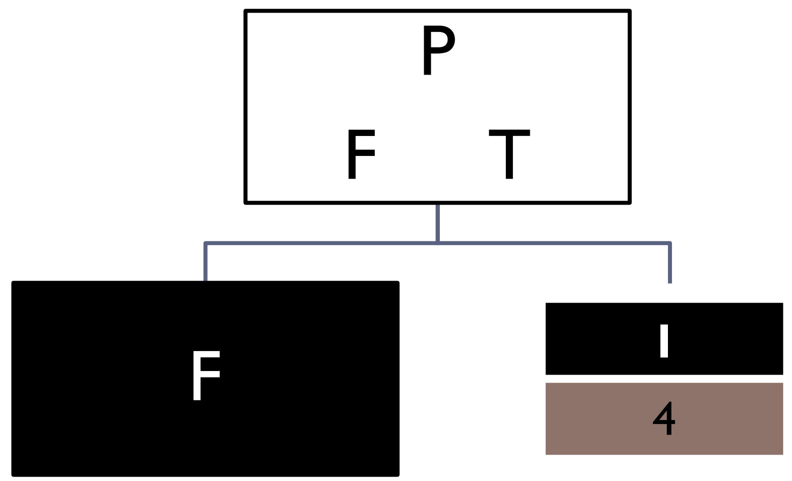
Learning Decision Trees: Example 2

- When P is False all the examples have the same classification (all false) → We stop and make a decision node with the value False.

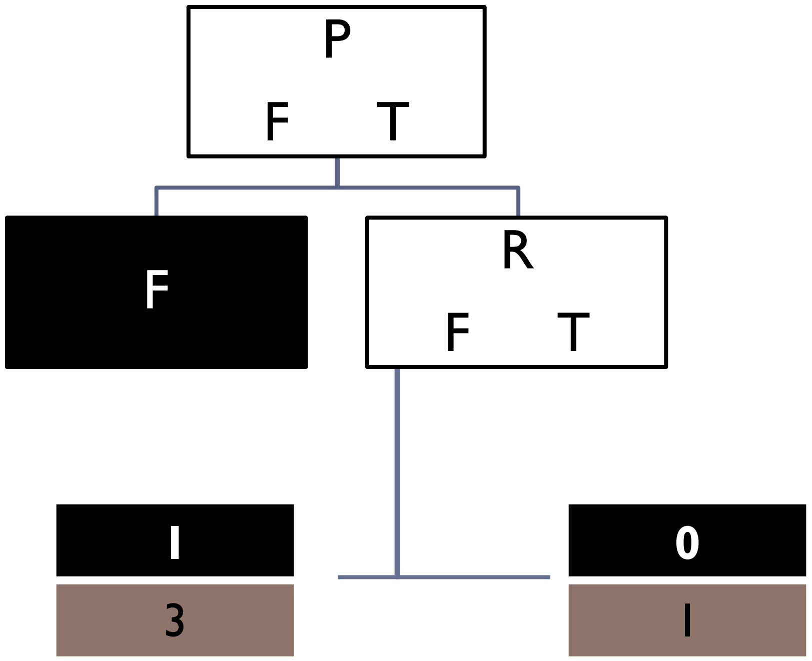


Learning Decision Trees: Example 2

- For the node P=True, we have both positive and negative examples, so we choose an attribute (the most important: how ?). In this case it R

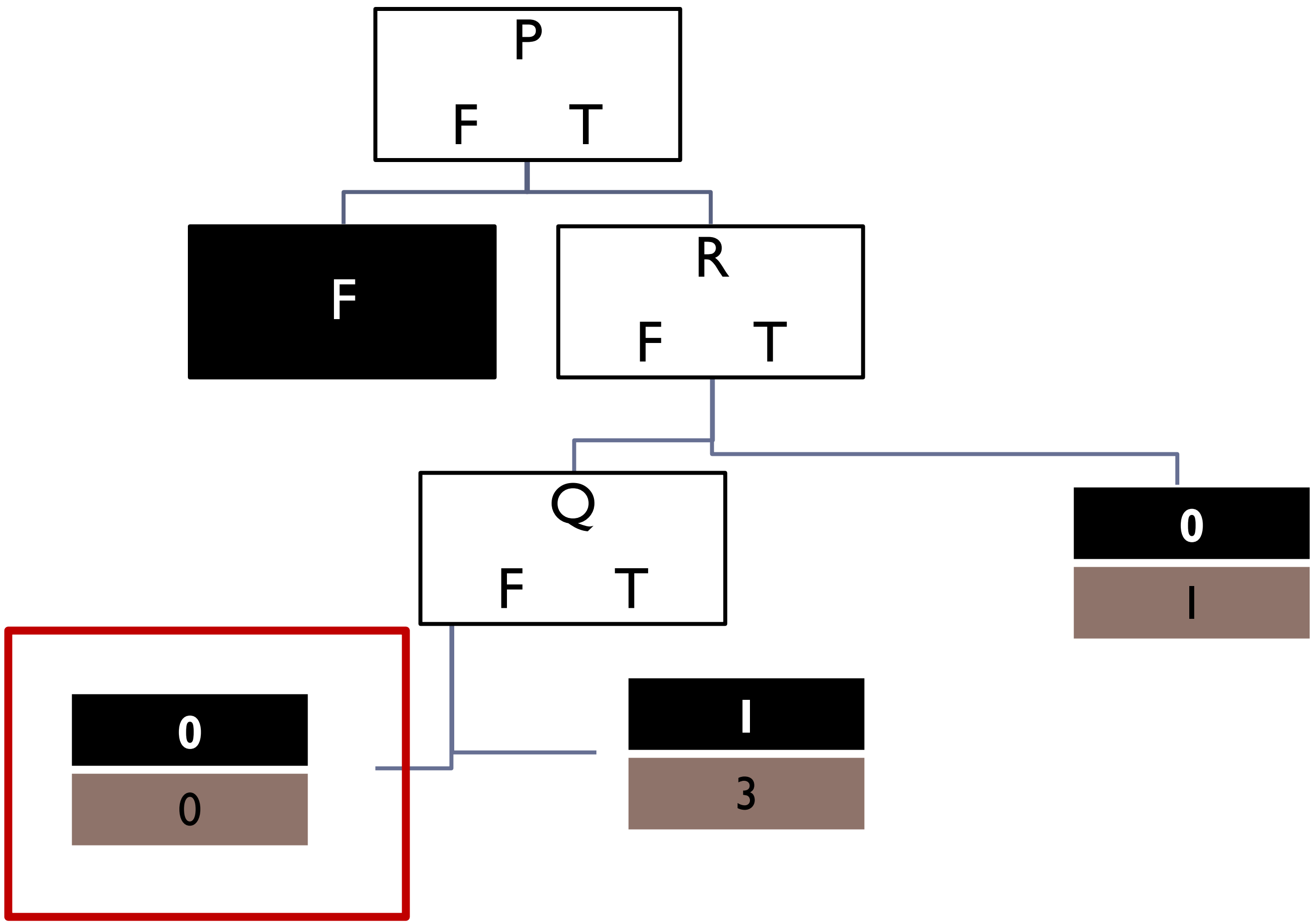


Learning Decision Trees: Example 2



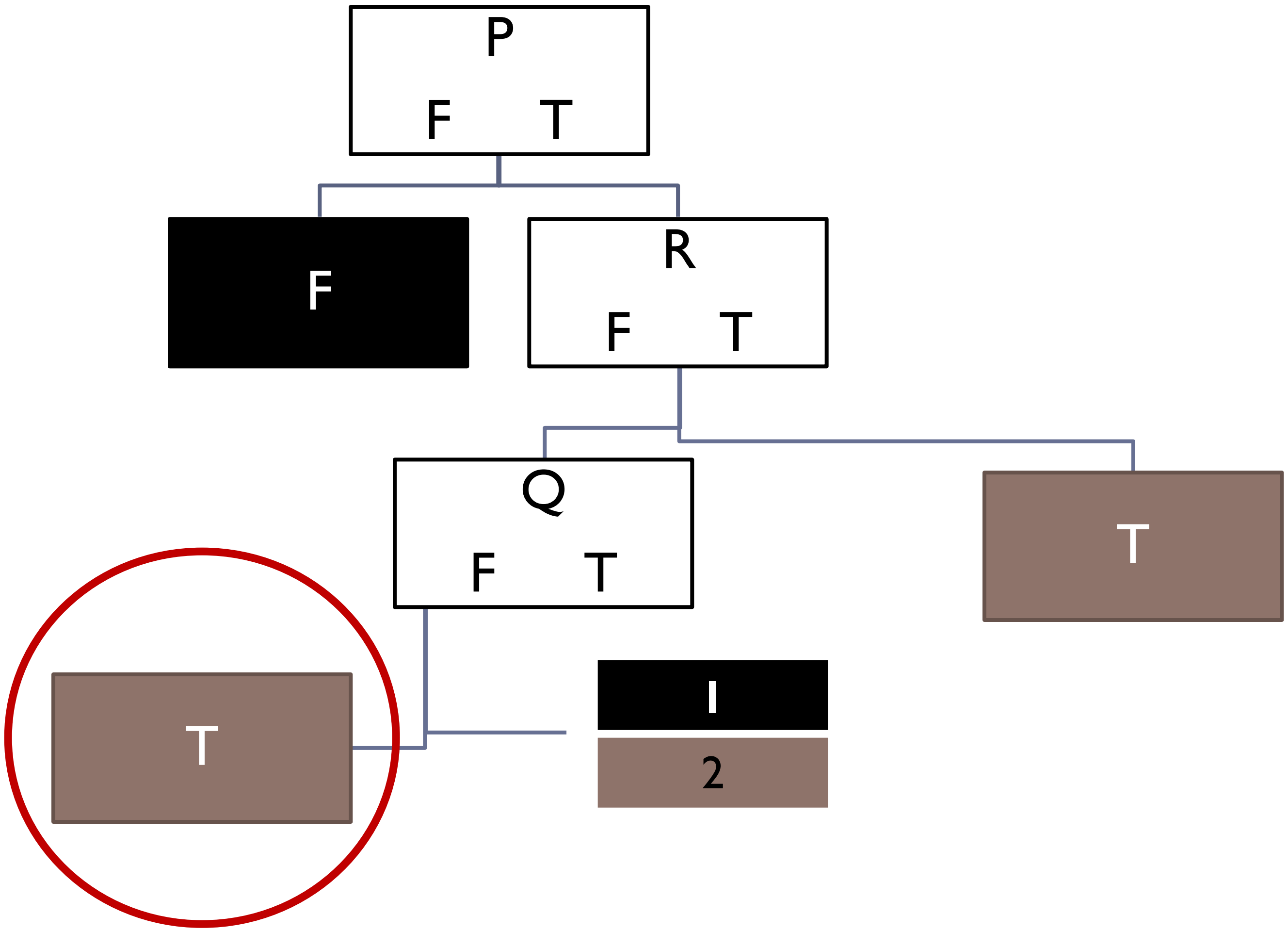
- For the node R=False, we have both positive and negative examples, so we choose an attribute: Q

Learning Decision Trees: Example 2



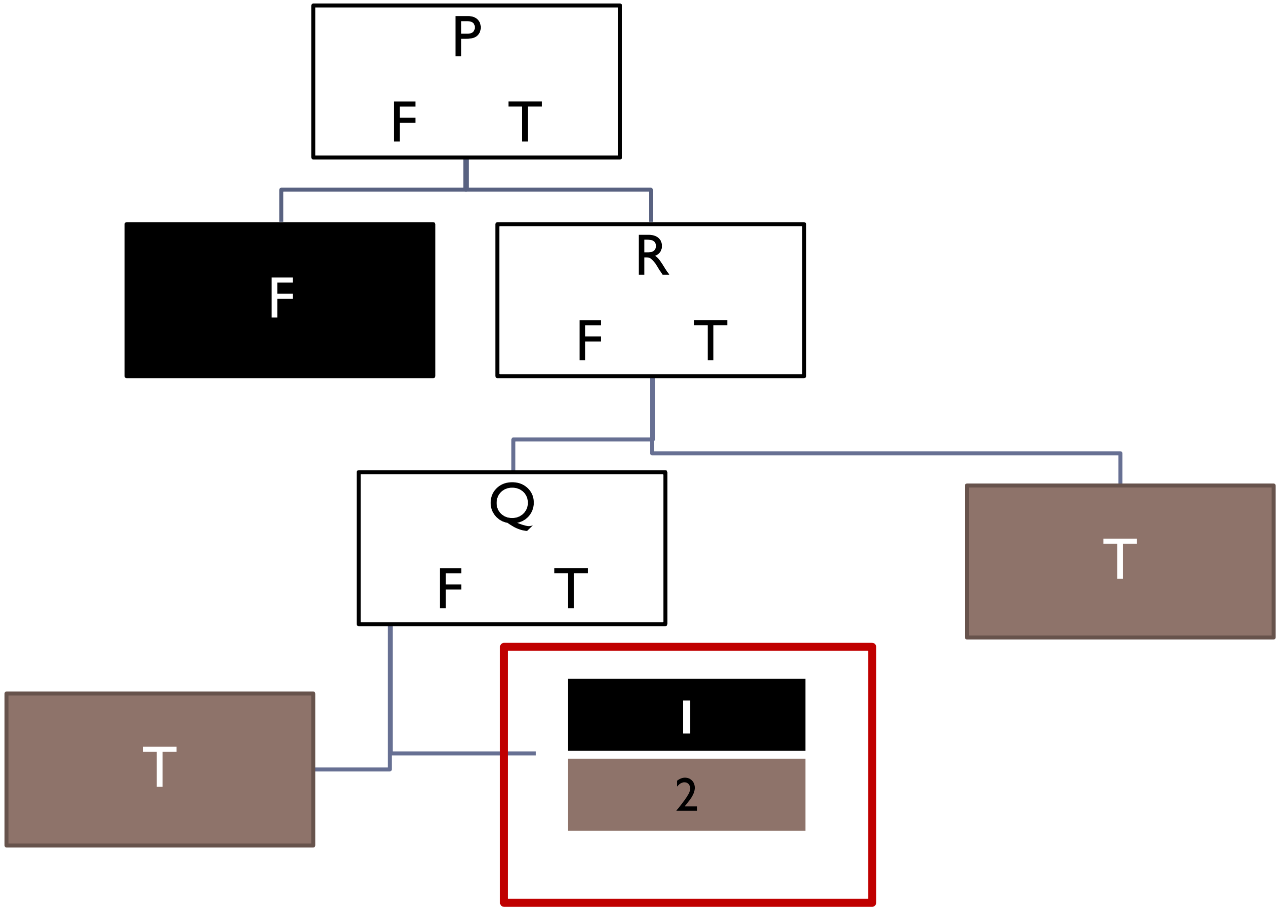
No examples → Take the majority at the parent

Learning Decision Trees: Example 2



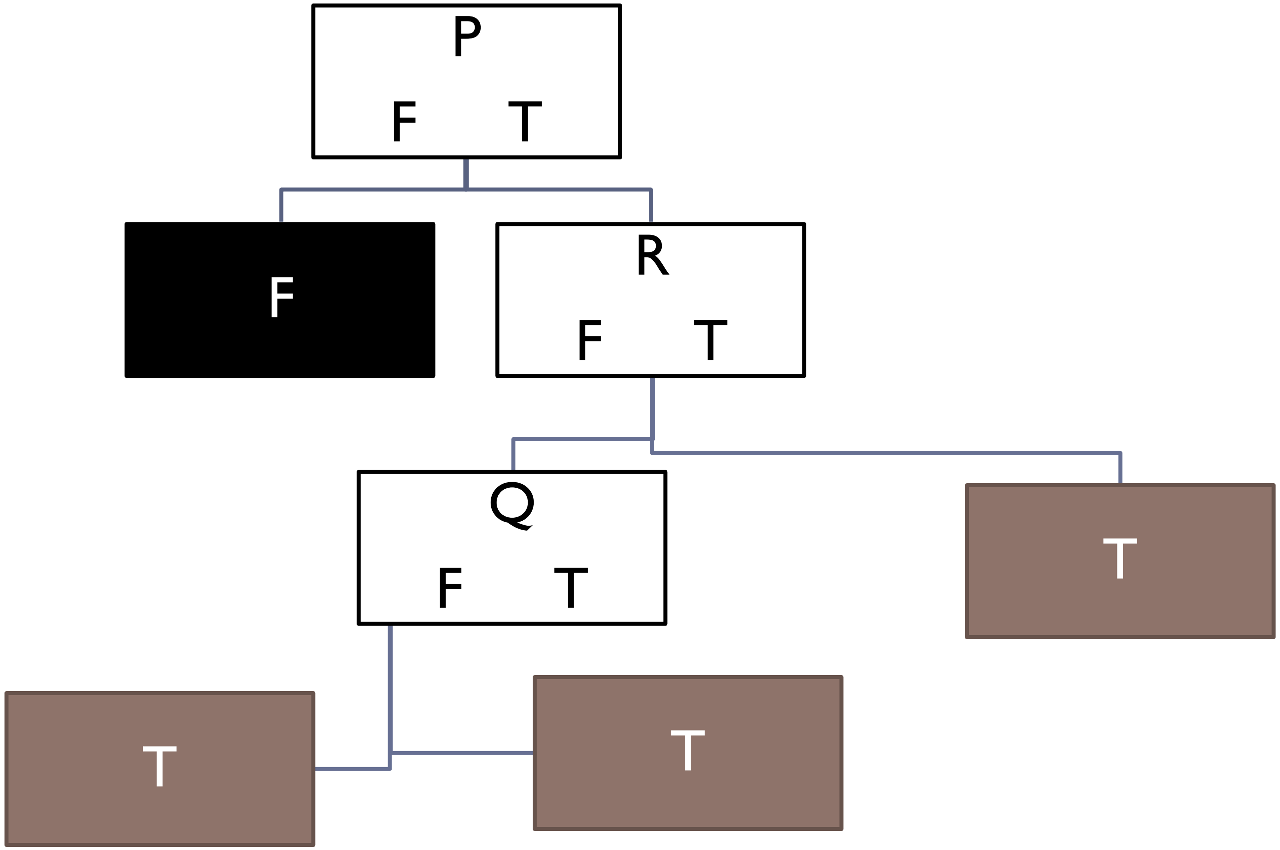
Not similar to the true function :
not enough examples

Learning Decision Trees: Example 2



No attributes (noise) → Majority

Learning Decision Trees: Example 2



The resulting tree is different from the true one

Learning Decision Trees

- Greedy algorithm for leaning decision trees (recursive):

1. If the remaining **examples** are **all** positive (or **all** negative), then we are done: we can answer Yes or No.
2. If there are **some** positive and **some** negative examples, then choose the best attribute to split them.
3. If there are **no** examples left, it means that no example has been observed for this combination, and we return a **default value: the plurality classification** of all the examples that were used in constructing the node's parent.
4. If there are **no attributes** left, but both positive and negative examples, it means that these examples have exactly the same description, but different classifications. We return a **default value:** the **plurality classification of the remaining examples**.

Learning Decision Trees (ID3)

function DECISION-TREE-LEARNING(*examples, attributes, parent_examples*) **returns** a tree

if *examples* **is empty then return** PLURALITY-VALUE(*parent_examples*)
else if all *examples* have the same classification **then return the** classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
    $A \leftarrow \mathrm{argmax}_{a \in attributes}$ IMPORTANCE(*a, examples*)
    tree ← a new decision tree with root test *A*
    for each value v_k of A **do**
        $exs \leftarrow \{e : e \in examples$ and $e.A = v_k\}$
        subtree ← DECISION-TREE-LEARNING(*exs, attributes* − *A, examples*)
        add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
    return *tree*

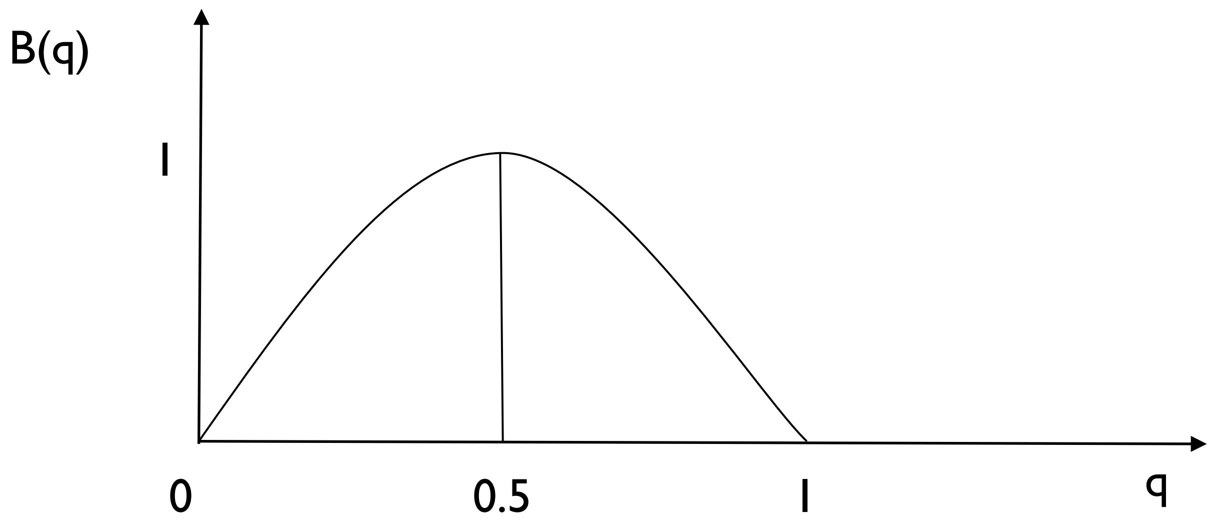
How to Choose the Attributes ?

- **Entropy**: A measure of randomness of a random variable.
- For the case of a binary random variable X (two outcomes, for example flipping a coin), with the probability of one event q (the other must 1-q), the entropy is:

$$H(X) = -\left(q\log_2(q) + (1-q)\log_2(1-q)\right)$$

- For simplicity, we denote it by B(q)
  - Example: a fair coin: q= 0.5 → H= 1
  - Example: a loaded coin q= 0.99 → H= 0.05
  - A fair coin is more random than a loaded coin. We gain more information by knowing the result of flipping a fair coin than a loaded coin.

Entropy of a Binary Random Variable



How to Choose the Attributes ?

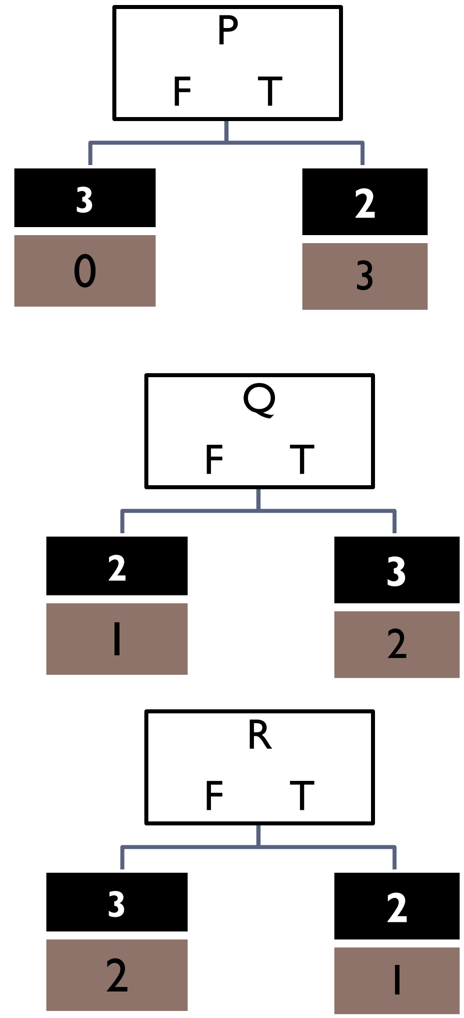
- At the beginning, we have a number of positive (p) and negative (n) examples. This can be seen as a binary random variable with q= p/(p+n)
- Choose the attribute that gives the largest information possible about the function
- → We choose the attribute which if tested gives the maximum reduction in entropy (maximum gain in information).
- Each attribute has k possible values, for each value k we have a set of positive (p_k) and negative (n_k) examples. This can be seen as a binary random variable with q= $p_k/(p_k + n_k)$. We compute the entropy for each value and take the average. The coefficient for each value is proportional to the size of its examples: $(p_k + n_k)/(p+n)$.
- This average is called the remainder (how much information remains after testing this attribute): For an attribute A with d values:

$$Remainder\,(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$$

- Choose the attribute with the smallest remainder

Choosing Attributes: Example 1

- Remainder(p)= (0+3) /8 *B(0/3) + (3+2)/8 * B(3/5) = 0.6
- Remainder(Q)= (1+2)/8* B(1/3) + (2+3)/8 * B(2/5) = 0.95
- Remainder(R)= (2+3)/8 * B(2/5) + (1+2)/8 * B(1/3)= 0.95
- → We choose P



Choosing Attributes: Example

- Remainder(Q)= (1+1)/5 * B(1/2) + (2+1)/5 * B(2/3)= 0.95
- Remainder(R)= (2+2)/5 *B(2/4)+ (1+0)/5*B(1/1) = 0.8
- → We choose R