Lab10

November 18, 2021

1 106119029 , AiML Lab 10

Collab Link

Exercise 1 : Decision Trees Construct a decision tree corresponding to each of the following Boolean functions. The examples x D have one attribute for each Boolean variable A, B, . . . in the formula; the target concept c(x) is the truth value of the formula itself. Assume the set D contains examples with all possible combinations of attribute values.

Hint: It may be helpful to write out the set D, i.e., the truth table for the variables and formula, first.

(a) A ňB

(b) A (B C)

(c) A XOR B

(d) (A B) (C D)

(e) (A B) (C XOR D)

[5]:
```
from typing import List, Dict, Optional
import pprint
```

[ ]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc inkscape
!pip install pypandoc
!pip install graphviz
!apt-get install graphviz
!pip install gvmagic
%load_ext gvmagic
```

This function `generate_tt` will generate the truth table for the given variables, applying the given functions

[6]:
```
def generate_tt(variables: List[str], func):
    sz = len(variables)
    table = []
    for i in range(0, 1 << (sz)):
        new_row = dict()
        for j in range(sz):
            val = False
```

```
            if ((i & (1 << j)) != 0):
                val = True
            new_row[variables[j]] = val
        table.append((new_row, func(new_row)))

    return table
```

These three functions are helpers to make nodes while generating a decision tree. A Node can be simply thought of as a dictionary with 'name', 'yes' branch and 'no' branch and also 'is_leaf' to indicate if its a leaf or not. The other attributes in make_node function is for reduction in the 2nd pass of the tree.

[7]:
```
def make_node(name):
    return {'name': name, 'yes': None, 'no': None, 'yes_vals': [], 'no_vals':␣
 ↪[], 'is_leaf': False}


def make_leaf(name):
    return {'name': name,  'is_leaf': True}


def make_node_2nd_pass(name):
    return {'name': name, 'yes': None, 'no': None, 'is_leaf': False}
```

This will be generating a brute force decision tree for us,using all the possible variable.

[8]:
```
def generate_decision_tree(variables: List[str], accumulated_values: Dict[str,␣
 ↪bool],  func):
    if len(variables) == 0:
        ans = func(accumulated_values)
        return (make_leaf(str(ans)), [ans])
    else:
        nd = make_node(variables[0])
        accumulated_values[variables[0]] = True
        nd['yes'], nd['yes_vals'] = generate_decision_tree(
            variables[1:], accumulated_values, func)
        accumulated_values[variables[0]] = False
        nd['no'], nd['no_vals'] = generate_decision_tree(
            variables[1:], accumulated_values, func)
        accumulated_values.pop(variables[0])
        return (nd, nd['yes_vals'] + nd['no_vals'])
```

This is the reduction function. This will take in the decision tree, and try to reduce it as much as possible, removing the need for branches if possible. For example, in a decision tree of (A and B) ,we dont need to use B in the 'no' branch of the A variable because if A is no(i.e False), the final result is anyways going to be False.

[9]:
```
def second_pass_to_remove_redundancy(node: Dict) -> Dict:
    if node['is_leaf']:
        return node
```

```
    else:
        mp = make_node_2nd_pass(node['name'])
        if node['yes_vals'] == node['no_vals']:
            return second_pass_to_remove_redundancy(node['no'])

        if all(x == node['yes_vals'][0] for x in node['yes_vals']):
            mp['yes'] = make_leaf(str(node['yes_vals'][0]))
        else:
            mp['yes'] = second_pass_to_remove_redundancy(node['yes'])
        if all(x == node['no_vals'][0] for x in node['no_vals']):
            mp['no'] = make_leaf(str(node['no_vals'][0]))
        else:
            mp['no'] = second_pass_to_remove_redundancy(node['no'])
        return mp
```

[16]:
```
%load_ext gvmagic
import graphviz
def make_dot_graph(g,root,id = 0):
    g.node(str(id))
    if root['is_leaf']:
        return
    else:
        g.node(str(2*id + 1),label=root['yes']['name'])
        g.node(str(2*id + 2),label=root['no']['name'])
        g.edge(str(id),str(2*id + 1), label='yes')
        g.edge(str(id),str(2*id + 2),label='no')
        make_dot_graph(g,root['yes'],2*id + 1)
        make_dot_graph(g,root['no'], 2*id + 2)
```

```
The gvmagic extension is already loaded. To reload it, use:
  %reload_ext gvmagic
```

This will print the tree.

[17]:
```
def print_tree(root, tabs=0):
    if root['is_leaf']:
        print(tabs*"    ", "Leaf: "+root['name'])
        return
    else:
        print(tabs*"    ", root['name'], 'yes')
        print_tree(root['yes'], tabs+1)
        print(tabs*"    ", root['name'], 'no')
        print_tree(root['no'], tabs+1)
```

[18]:
```
graph = ""
```

This function will show the truth table for the given expression and then goes onto print the decision tree after making it.

[23]:
```
def show_decision_tree_and_truth_table(expr, variables, func):
    global graph
    print("="*80)
    print(expr)
    print("\nTRUTH TABLE FOR THE GIVEN EXPRESSION : {}".format(expr))
    print(' '.join(map(lambda x: "{:<10}".format(x), variables + ["Result"])))
    for x in generate_tt(variables, func):
        print(' '.join(map(lambda y: "{:<10}".format(
            str(y)),  list(x[0].values()) + [x[1]])))

    x = generate_decision_tree(variables, dict(), func)
    x = second_pass_to_remove_redundancy(x[0])
    print_tree(x)
    print("="*80)
    g = graphviz.Digraph(strict=True)
    make_dot_graph(g,x)
    graph = g.source
    %dotstr graph
    print()
```

All the inputs given for the question

[24]:
```
inputs = [
    ("A and (not B)", ['A', 'B'], lambda h: h['A'] and not h['B']),
    ("A or (B and C)", ['A', 'B', 'C'],
     lambda h: h['A'] or (h['B'] and h['C'])),
    ("A xor B", ['A', 'B'], lambda h: h['A'] ^ h['B']),
    ("(A and B) or (C and D)", ['A', 'B', 'C', 'D'],
     lambda h: (
        h['A'] and h['B']) or (h['C'] and h['D'])),
    ("( A or B ) and (C xor D)", ['A', 'B', 'C', 'D'],
        lambda h: (h['A'] or h['B']) and (h['C'] ^ h['D']))
]
```

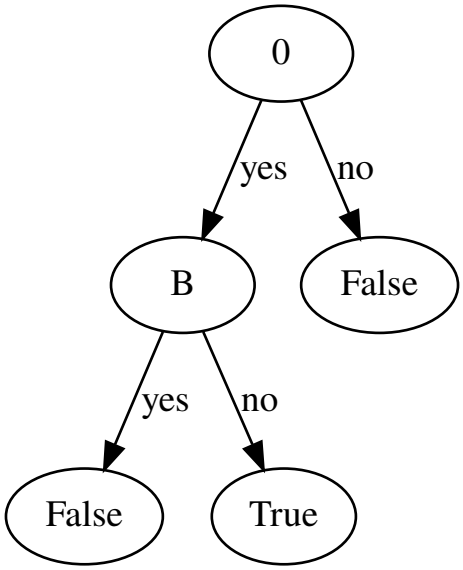
1.1 A and (not B)

[25]:
```
show_decision_tree_and_truth_table(*(inputs[0]))
```

```
================================================================================
A and (not B)

TRUTH TABLE FOR THE GIVEN EXPRESSION : A and (not B)
A          B          Result
False      False      False
True       False      True
False      True       False
True       True       False
 A yes
```

```
    B yes
        Leaf: False
    B no
        Leaf: True
A no
    Leaf: False
================================================================================
```



1.2 A or (B and C)

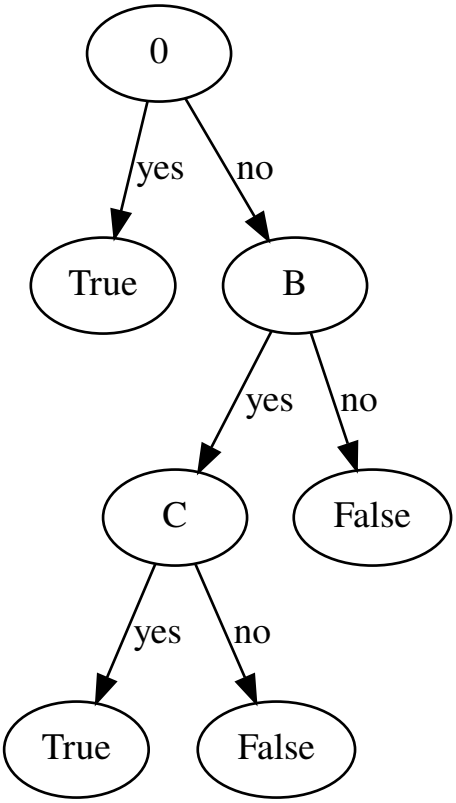
[26]:
```
show_decision_tree_and_truth_table(*(inputs[1]))
```

```
================================================================================
A or (B and C)

TRUTH TABLE FOR THE GIVEN EXPRESSION : A or (B and C)
A          B          C          Result
False      False      False      False
True       False      False      True
False      True       False      False
True       True       False      True
False      False      True       False
True       False      True       True
False      True       True       True
True       True       True       True
```

```
A yes
    Leaf: True
A no
    B yes
        C yes
            Leaf: True
        C no
            Leaf: False
    B no
        Leaf: False
```

```
================================================================================
```

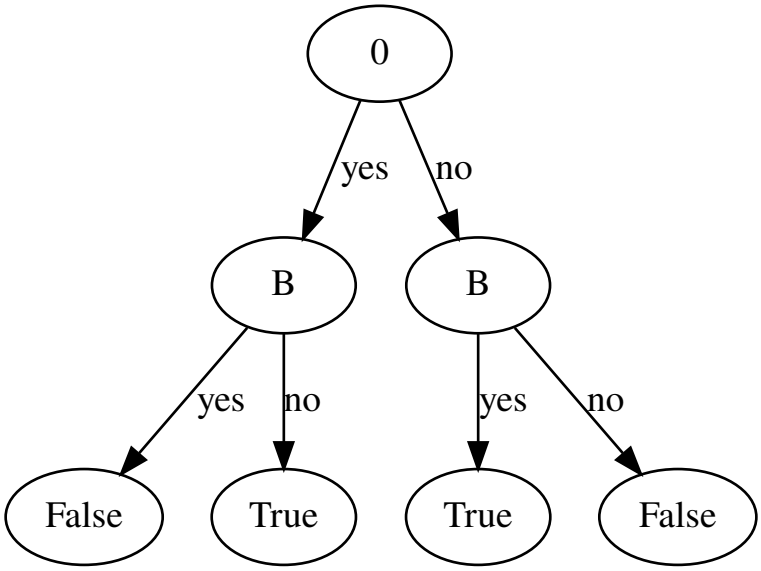


1.3 A xor B

[27]:
```
show_decision_tree_and_truth_table(*(inputs[2]))
```

```
================================================================================
A xor B
```

```
TRUTH TABLE FOR THE GIVEN EXPRESSION : A xor B
A          B          Result
False      False      False
True       False      True
False      True       True
True       True       False
 A yes
     B yes
         Leaf: False
     B no
         Leaf: True
 A no
     B yes
         Leaf: True
     B no
         Leaf: False
================================================================================
```

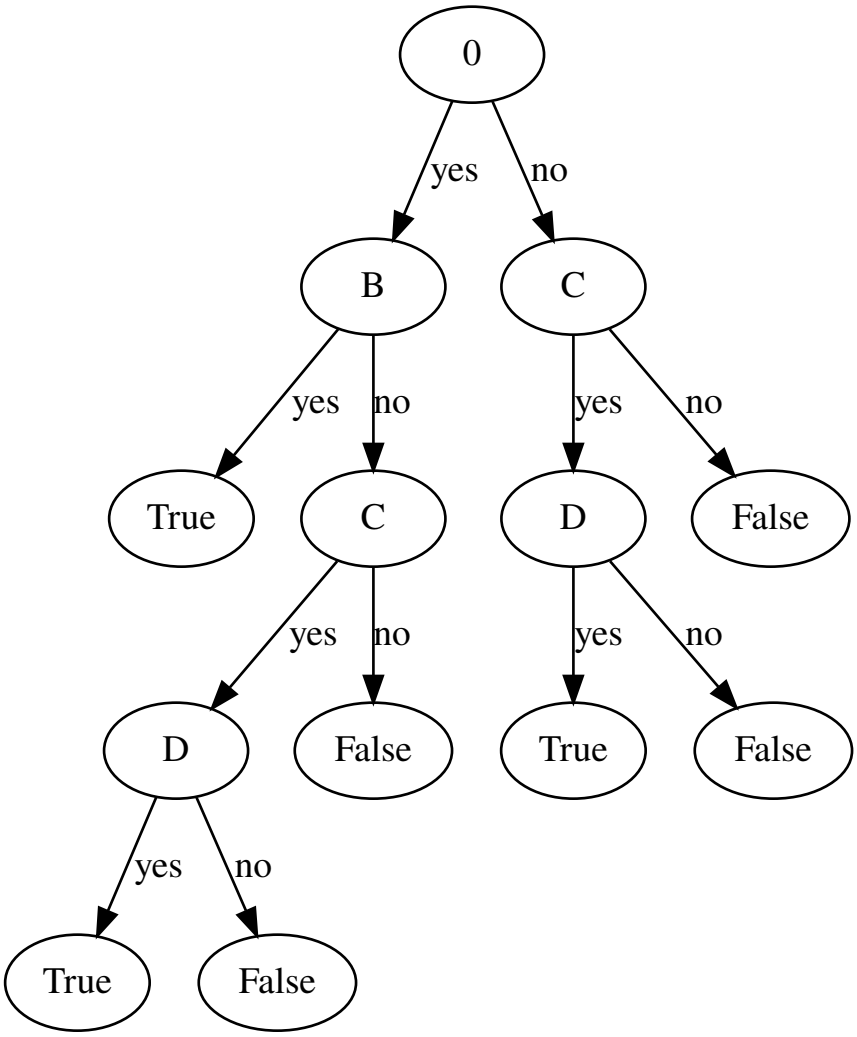


1.4 (A and B) or (C and D)

[28]:
```
show_decision_tree_and_truth_table(*(inputs[3]))
```

```
================================================================================
(A and B) or (C and D)
```

```
TRUTH TABLE FOR THE GIVEN EXPRESSION : (A and B) or (C and D)
A          B          C          D          Result
False      False      False      False      False
True       False      False      False      False
False      True       False      False      False
True       True       False      False      True
False      False      True       False      False
True       False      True       False      False
False      True       True       False      False
True       True       True       False      True
False      False      False      True       False
True       False      False      True       False
False      True       False      True       False
True       True       False      True       True
False      False      True       True       True
True       False      True       True       True
False      True       True       True       True
True       True       True       True       True
 A yes
     B yes
         Leaf: True
     B no
         C yes
             D yes
                 Leaf: True
             D no
                 Leaf: False
         C no
             Leaf: False
 A no
     C yes
         D yes
             Leaf: True
         D no
             Leaf: False
     C no
         Leaf: False
================================================================================
```



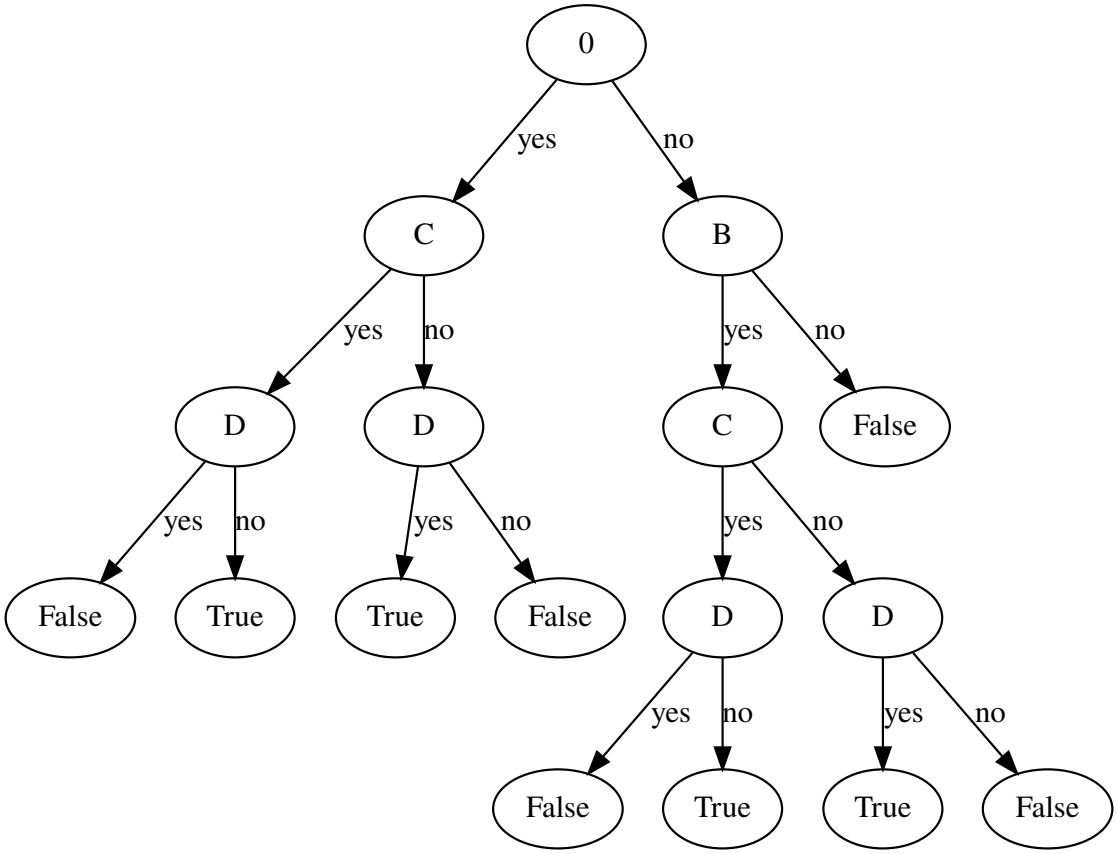
1.5 (A or B) and (C xor D)

[29]:
```
show_decision_tree_and_truth_table(*(inputs[4]))
```

```
================================================================================
( A or B ) and (C xor D)

TRUTH TABLE FOR THE GIVEN EXPRESSION : ( A or B ) and (C xor D)
A          B          C          D          Result
False      False      False      False      False
True       False      False      False      False
False      True       False      False      False
True       True       False      False      False
```

```
False      False      True       False      False
True       False      True       False      True
False      True       True       False      True
True       True       True       False      True
False      False      False      True       False
True       False      False      True       True
False      True       False      True       True
True       True       False      True       True
False      False      True       True       False
True       False      True       True       False
False      True       True       True       False
True       True       True       True       False
 A yes
     C yes
         D yes
             Leaf: False
         D no
             Leaf: True
     C no
         D yes
             Leaf: True
         D no
             Leaf: False
 A no
     B yes
         C yes
             D yes
                 Leaf: False
             D no
                 Leaf: True
         C no
             D yes
                 Leaf: True
             D no
                 Leaf: False
     B no
         Leaf: False
================================================================================
```


0
yes
no
C
B
yes
no
yes
no
D
D
C
False
yes
no
yes
no
yes
no
False
True
True
False
D
D
yes
no
yes
no
False
True
True
False

[ ]:

[ ]:

Question 2

DATE
PAGE No.

2.a) Ans

Using t as miscalculation error :

Miscalculation error = $1 - \max\left(P\left(\frac{i}{t}\right)\right)$

∴ in t1 :

For choice $D_{1,1}$

$P(C_1) = \frac{1}{4}$

$P(C_2) = \frac{1}{4}$

$P(C_3) = 0$

$P(C_4) = 0$

For choice $D_{1,2}$

$P(C_1) = 0$

$P(C_2) = 0$

$P(C_3) = \frac{1}{4}$

$P(C_4) = \frac{1}{4}$

$T_{misclassification} = 1 - \frac{1}{4} : \frac{3}{4}$ for $D_{1,1}$ and

$\Rightarrow \frac{3}{4} - \frac{1}{4} - \frac{1}{4} = \frac{1}{4}$

In t2

For choice $D_{2,1}$

$P(C_1) = \frac{4}{12}$

$P(C_2) = \frac{2}{12}$

$P(C_3) = \frac{6}{12}$

$P(C_4) = 0$

For choice $D_{2,2}$

$P(C_1) = \frac{2}{12}$

$P(C_2) = \frac{4}{12}$

$P(C_3) = 0$

$P(C_4) = \frac{6}{12}$

∴ $T_{misclassification} = 1 - \frac{1}{2} = \frac{1}{2}$ for both

$\Rightarrow \frac{3}{4} - \frac{1}{4} - \frac{1}{4} = \frac{1}{4}$

2.a. continued

Using τ as ~~error~~ entropy :

$$\Rightarrow \text{Entropy} = \bullet - \sum_{1}^{4} P(x_i) \times \log_2(P(x_i))$$

$$\therefore D_{1,1} \Rightarrow -\left(\frac{1}{2}\log\frac{1}{2} + \frac{1}{2}\log\frac{1}{2} + 0+0\right)$$

$$= 1$$

$$\therefore D_{1,2} \Rightarrow -\left(0 + 0 + \frac{1}{2}\log\frac{1}{2} + \frac{1}{2}\log\frac{1}{2}\right) = 1$$

$$\therefore D_{2,1} \Rightarrow -\left(\frac{4}{12}\log\frac{4}{12} + \frac{2}{12}\log\frac{2}{12} + \frac{6}{12}\log\frac{6}{12} + 0\right)$$

$$= 1.45$$

$$\therefore D_{2,2} \Rightarrow -\left(\frac{4}{12}\log\frac{4}{12} + \frac{2}{12}\log\frac{2}{12} + \frac{6}{12}\log\frac{6}{12} + 0\right)$$

$$= 1.45$$

$$\therefore \Delta\tau_{entropy} = 1 + 1 - \frac{1}{2}(1.45) - \frac{1}{2}(1.45)$$

$$= 0.54$$

$$\therefore \Rightarrow \Delta\tau_{miscalculation} = 0.25$$

$$\Rightarrow \Delta\tau_{entropy} = 0.54$$

Therefore t_1 has a better split than t_2 because it has more information gain.

2.b. Ans

→ t1 is a better decision tree because there is less entropy after the decision tree has been made and the classes seem to be split very uniformly and hence has less impurity.

2.c. Ans

ID3 is a classification algorithm that follows a greedy approach of building a decision tree, and it selects the best attribute by choosing on that yields maximum info gain or minimum entropy.

From the calculations,

$$\text{Entropy}(t_1) < \text{Entropy}(t_2)$$

$$1 < 1.45$$

→ Hence we can say the ID3 would make the tree t1 and not t2.