# Recitation 2
# Decision Trees

10-301/10-601: Introduction to Machine Learning

01/27/2023

## 1 Programming: Tree Structures and Algorithms

**Topics Covered:**

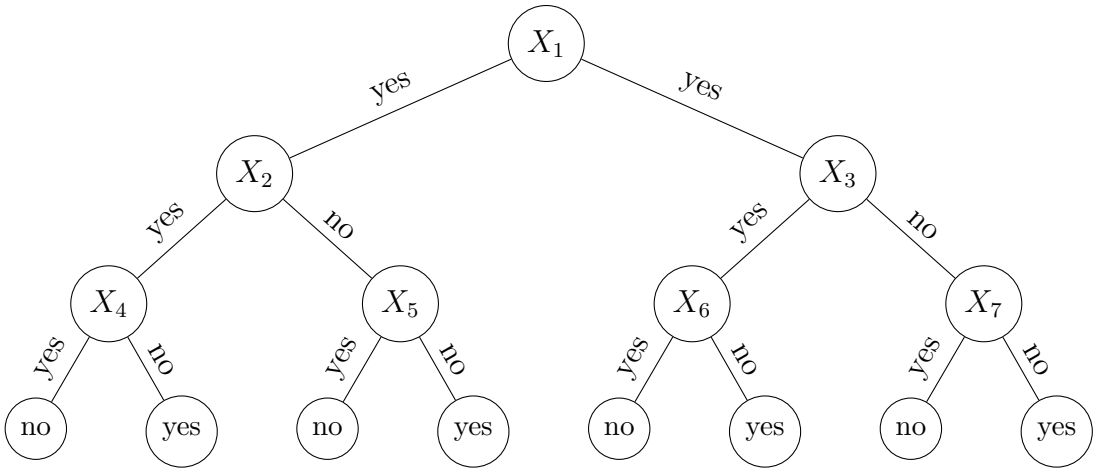
- Depth of nodes and trees
- Recursive traversal of trees
  - Depth First Search
    - Pre-order Traversal
    - In-order Traversal
    - Post-order Traversal
  - Breadth First Search (Self Study)
- Debugging in Python

**Questions:**

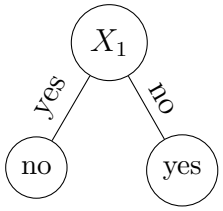
1. Depth of a tree definition

2. Depth of a node definition

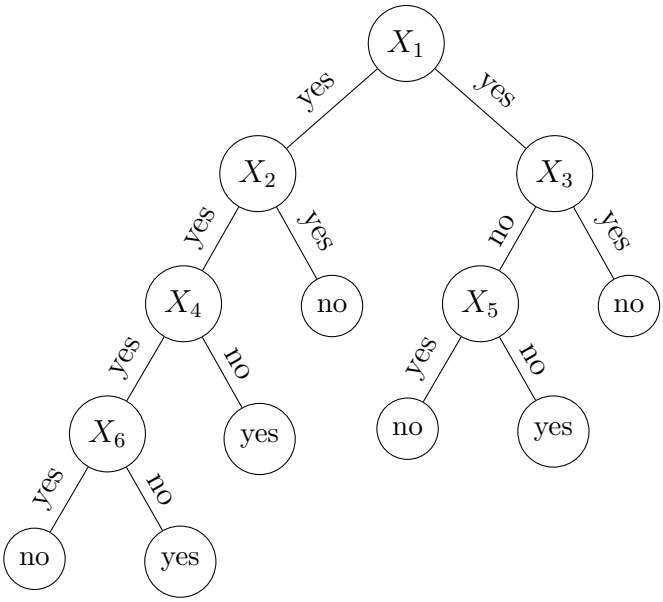
3. What is the depth of tree A? What is the depth of node $X_4$ in tree A?




4. What is the depth of tree B?




5. What is the depth of tree C? What are the depths of nodes $X_1$ and $X_5$ in tree C?



6. In-class coding and explanation of Depth First Traversal in Python.
   Link to the code:
   https://colab.research.google.com/drive/1KypCp2tPDad4gdHjL1FH4DqbBnM5CfCr?usp=sharing

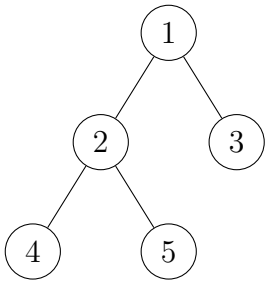
**Pre-order, Inorder and Post-order Tree Traversal**

```
# This class represents an individual node
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

def traversal1(root):
    if root is not None:
        # First recurse on left child
        traversal1(root.left)
        # then recurse on right child
        traversal1(root.right)
        # now print the data of node
        print(root.val, end='\t')

def traversal2(root):
    if root is not None:
        # First print the data of node
        print(root.val, end='\t')
        # Then recurse on left child
        traversal2(root.left)
        # Finally recurse on right child
        traversal2(root.right)

def traversal3(root):
    if root is not None:
        # First recurse on left child
        traversal3(root.left)
        # then print the data of node
        print(root.val, end='\t')
        # now recurse on right child
        traversal3(root.right)

def build_a_tree():
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.left.right = Node(5)
    return root
```

```
if __name__ == '__main__':
    root = build_a_tree()
    print('traversal1 of the binary tree is: ')
    traversal1(root)
    print()
    print('traversal2 of the binary tree is: ')
    traversal2(root)
    print()
    print('traversal3 of the binary tree is: ')
    traversal3(root)
```

Now, identify which traversal function is pre-order, in-order, post-order DFS:

- traversal1() is
- traversal2() is
- traversal3() is




**Code Output**

```
traversal1 of the binary tree is:


traversal2 of the binary tree is


traversal3 of the binary tree is
```

# 2 ML Concepts: Mutual Information

**Information Theory Definitions:**

- $H(Y) = -\sum_{y \in values(Y)} P(Y = y) \log_2 P(Y = y)$
- $H(Y \mid X = x) = -\sum_{y \in values(Y)} P(Y = y|X = x) \log_2 P(Y = y|X = x)$
- $H(Y \mid X) = \sum_{x \in values(X)} P(X = x) H(Y \mid X = x)$
- $I(X;Y) = H(Y) - H(Y \mid X)$

**Exercises**

1. Calculate the entropy of tossing a fair coin.

2. Calculate the entropy of tossing a coin that lands only on tails. *Note:* $0 \cdot \log_2(0) = 0$.

3. Calculate the entropy of a fair dice roll.

4. When is the mutual information $I(X;Y) = 0$?

**Used in Decision Trees:**

| Outlook ($X_1$) | Temperature ($X_2$) | Humidity ($X_3$) | Play Tennis? ($Y$) |
|---|---|---|---|
| sunny | hot | high | no |
| overcast | hot | high | yes |
| rain | mild | high | yes |
| rain | cool | normal | yes |
| sunny | mild | high | no |
| sunny | mild | normal | yes |
| rain | mild | normal | yes |
| overcast | hot | normal | yes |

1. Using the dataset above, calculate the mutual information for each feature $(X_1, X_2, X_3)$ to determine the root node for a Decision Tree trained on the above data.
   - What is $I(Y; X_1)$?
   - What is $I(Y; X_2)$?
   - What is $I(Y; X_3)$?
   - What feature should be split on at the root node?

2. Calculate what the next split should be.

3. Draw the resulting tree.

# 3 ML Concepts: Construction of Decision Trees

In this section, we will go over how to construct our decision tree learner on a high level. The following questions will help guide the discussion:

1. What exactly are the tasks we are tackling?
2. What are the inputs and outputs at training time? At testing time?
3. At each node of the tree, what do we need to store?
4. What do we need to do at training time?
5. What happens if max depth is 0?
6. What happens if max depth is greater than the number of attributes?

# 4 Programming: Debugging with Trees

### pdb and common commands

- `import pdb; pdb.set_trace()` (`breakpoint()` also allowed as per PEP 553)
- `p variable` (print value of variable)
- `n` (next)
- `s` (step into subroutine)
- `ENTER` (repeat previous command)
- `q` (quit)
- `l` (list where you are)
- `b` (breakpoint)
- `c` (continue)
- `r` (continue until the end of the subroutine)
- `!code` (run Python code)

### Real Practice

These are some (contrived) examples based on actual bugs previous students had. Link to the code: https://colab.research.google.com/drive/1KypCp2tPDad4gdHjL1FH4DqbBnM5CfCr?usp=sharing

### Buggy Code

```
# Reverse the rows of a 2D array
def reverse_rows(original):
    rows = len(original)
    cols = len(original[0])

    new = [[0] * cols] * rows

    for i in range(rows):
        for j in range(cols):
            new_index = rows - i
            new[new_index][j] = original[i][j]

    return new


if __name__ == '__main__':
    a = [[1, 2],
         [3, 4],
         [5, 6]]
    print(reverse_rows(a))
```

### Buggy Code

```
import numpy as np

# biggest_col takes a binary 2D array and returns the index of the
# column with the most non-zero values. In case of a tie, return
# the smallest index.
def biggest_col(mat):
    num_col = len(mat[0])
    max_count = -1
    max_index = -1

    # iterate over the columns of the matrix
    for col in range(num_col):
        # counts the number of nonzero values
        count = np.count_nonzero(mat[:, col])
        # change max if needed
        if count >= max_count:
            max_count = count
            max_index = col

    return max_index


# Helper function that returns the number of nonzero elements in
# mat in column col.
def get_count(mat, col):
    num_row = len(mat)
    count = 0
    for row in range(num_row):
        count += (mat[row][col] == 0)
    return count


if __name__ == '__main__':
    # Expected answer: column index 2
    mat = [[1, 0, 0, 1],
           [0, 1, 1, 1],
           [1, 0, 0, 0],
           [0, 1, 1, 1],
           [0, 0, 1, 0]]
    assert biggest_col(mat) == 2
```