Recitation 2
Decision Trees

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

09/10/2021

1 Programming: Tree Structures and Algorithms

Topics Covered:

- Depth and height of trees
- Recursive traversal of trees
  - Depth First Search
    - Pre Order Traversal
    - Inorder Traversal
    - Post Order Traversal
  - Breadth First Search (Self Study)
- Debugging in Python

Questions:

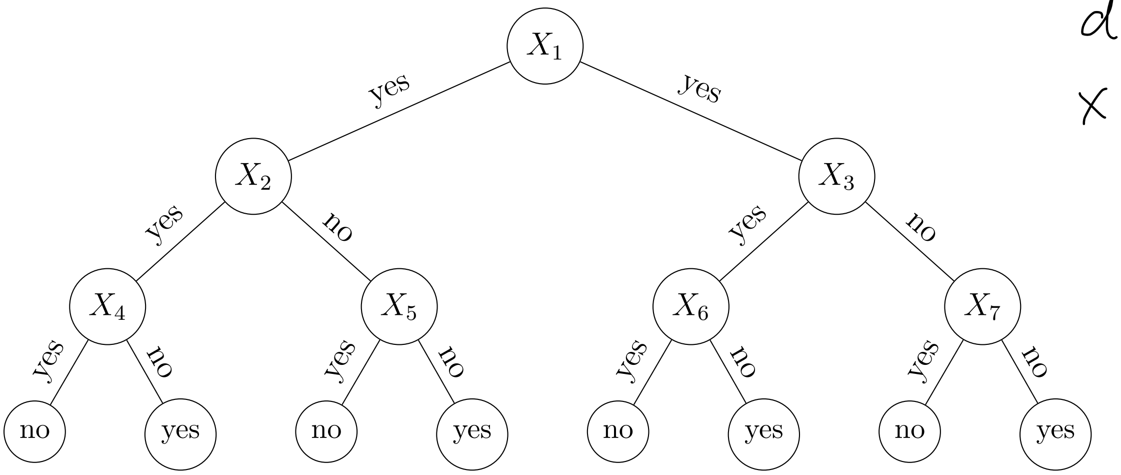
1. Depth of a tree definition

   # of edges in longest path from root

2. Depth of a node definition

   # of edges in the path from root to node

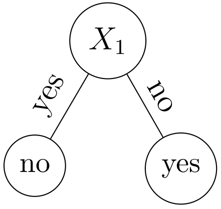
3. What is the depth of tree A? What is the depth of node X_4 in tree A?



$d = 3$

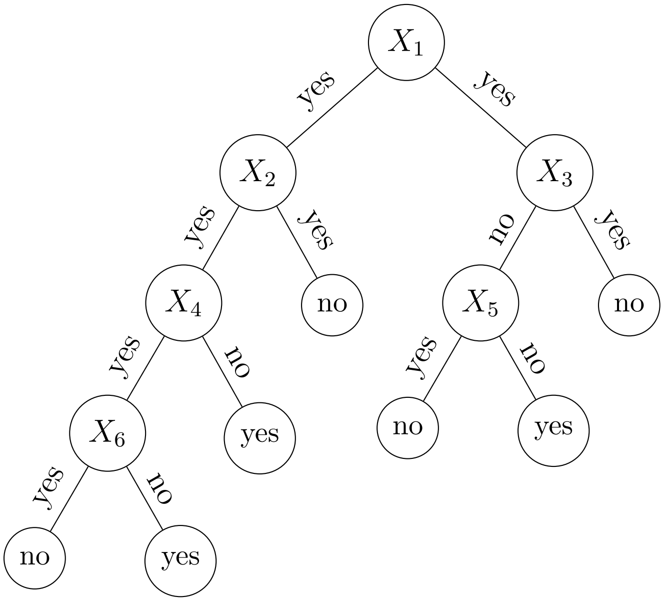
$x_4 = 2$

4. What is the depth of tree B?



1

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



$d = 4$

$x_1 = 0$

$x_5 = 2$

6. In class coding and explanation of Depth First Traversal in Python.
Link to the code: https://colab.research.google.com/drive/11OjtswvTVxY1Jxvko75X6_U_-Dfsh4ZQ?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
```

base case

```
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, "\t",end="")
```

L R S

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

S L R

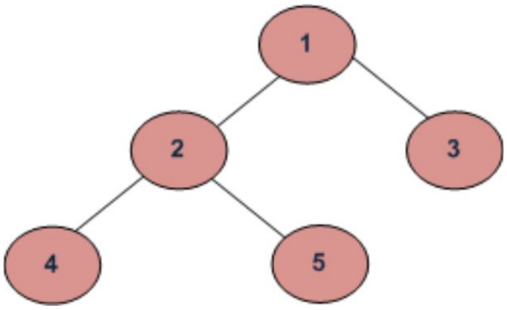
```
def traversal3(root):
    if root is not None:
        # First recur on left child
        traversal3(root.left)
        # then print the data of node
        print(root.val, "\t",end="")
        # now recur on right child
        traversal3(root.right)
```

L S R

```
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 binary tree is: ")
traversal1(root)
print("\n")
print ("traversal2 of binary tree is: ")
traversal2(root)
print("\n")
print ("traversal3 of binary tree is: ")
traversal3(root)
```



Code Output

```
LRS Traversal1 of binary tree is:
                                  4,5,2,3,1

SLR Traversal2 of binary tree is 1,2,4,5,3


LSR Traversal3 of binary tree is 4,2,5,1,3
```

Now, identify which traversal function is Pre-Order, In-Order, Post-Order DFS respectively :

- traversal1() is post order
- traversal2() is pre order
- traversal3() is inorder

2 ML Concepts: Mutual Information

Information Theory Definitions:

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

Exercises

1. Calculate the entropy of tossing a fair coin.

head: $\frac{1}{2}$

tails: $\frac{1}{2}$

$$= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

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

head: 0

tails: 1

$$= -0\log_2(0) - 1\log_2(1) = 0$$

3. Calculate the entropy of a fair dice roll.

1: $\frac{1}{6}$
2: $\frac{1}{6}$
3: $\frac{1}{6}$
4: $\frac{1}{6}$
5: $\frac{1}{6}$
6: $\frac{1}{6}$

$$-\left(\frac{1}{6}\log_2\left(\frac{1}{6}\right) \cdot 6\right) = -\log_2\left(\frac{1}{6}\right)$$

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

$$\Rightarrow H(x) - H(X|Y) = 0$$

$$\Rightarrow H(X) = H(X|Y)$$

Used in Decision Trees:

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

$H(Y) = -\left(\frac{2}{8}\log_2\left(\frac{2}{8}\right) + \frac{6}{8}\log_2\left(\frac{6}{8}\right)\right)$

$=$

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.

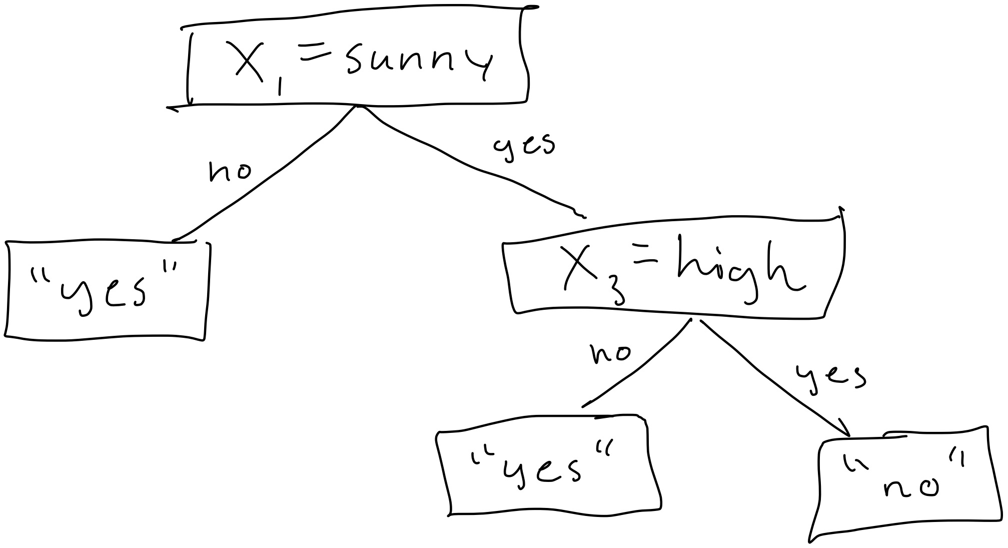
   → entropy of rain, overcast, & sunny

   - What is $I(Y; X_1)$? $\underbrace{H(Y)}_{-\left(\frac{6}{8}\log_2\left(\frac{6}{8}\right)+\frac{2}{8}\log_2\left(\frac{6}{8}\right)\right)} - H(Y|X_1) = 0 + 0 + -\left(\frac{1}{3}\log_2\left(\frac{1}{3}\right) + \frac{2}{3}\log_2\left(\frac{2}{3}\right)\right)$ $-\frac{3}{8}\cdot$
     0.467
   - What is $I(Y; X_2)$? 0.061
   - What is $I(Y; X_3)$? 0.311
   - What feature should be split on at the root node? X_1 → largest mutual information

2. Calculate what the next split should be.

   X_3

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? What are the inputs and outputs?

1) train the model →inputs: data, root node, max depth
2) predict the model
3) Calculate error →outputs: tree

basecase: we achieve stoping criteria
1) maximum mutual information is zero
2) data is pure (in label)
3) node depth is greater than max depth
4) store .majority vote

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?

Class node → self.depth → self.left
→ self.data → self.right
→ self.majority vote → self.attribute

Helper functions
↳ calc mutual info (x,y)

build out the tree: do this if we dont stop
1) find feature to split on: IB → highest MI
2) split dataset
3) recurse! train(data.left, node.left, max_depth)
train(data.right, node.right, max_depth)

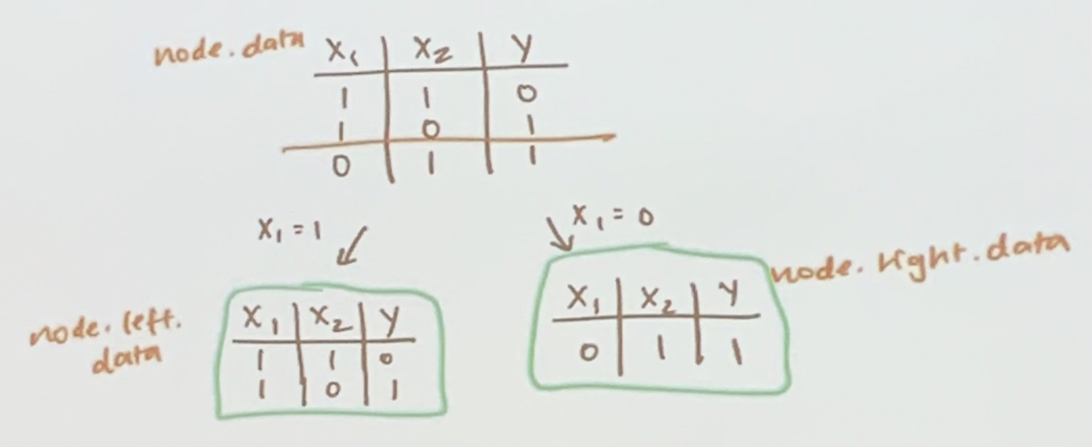
4. What do we need to do at training time?

5. What happens if max depth is 0?

6. What happens if max depth is greater than the number of attributes?

2) Predict
→ inputs: data, tree
→ predict 1 point at a time
→ basecase: return majority vote
↳ node.left & node.right = None
→ recursive case
1) check attribute
2) if attribute = 1: predict(data, node.left)
else: predict(data, node.right)
→ save all predictions to an array

3) calculate error rate!



X_1	X_2	Y
1	1	0
1	0	1
0	1	1

X_1	X_2	Y
1	1	0
1	0	1

X_1	X_2	Y
0	1	1

4 Programming: Debugging w/ Trees

pdb and common commands

- `import pdb` then `pdb.set_trace()`
- n (next)
- ENTER (repeat previous)
- q (quit)
- p variable (print value)
- c (continue)
- b (breakpoint)
- l (list where you are)
- s (step into subroutine)
- r (continue until the end of the subroutine)
- ! python command

Real Practice

- In this (extremely contrived) example, we will reversing a 2d list in python.

Buggy Code

- add pdb.set_trace() before the line that is causing the error

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

    new = [[0]*cols]*rows

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

a = [[1,2],
    [3,4],
    [5,6]]

print(reverse(a))
```

Buggy Code

```
import numpy as np

Mat = [[1,0,0,0],
       [0,1,1,0],
       [1,0,0,0],
       [0,1,-1,1],
       [0,0,1,0]]

#biggestCol takes a binary - 2d array without headers and returns
#the index of the column with the most non-zero values
def biggestCol(Mat):

    #get the number of columns and initialize variables
    numCol = len(Mat[0])
    maxValue = -1
    maxIndex = -1

    #iterate over the columns of the matrix
    for col in range(numCol):

        #counts the number of nonzero values
        count = np.count_nonzero(Mat[:,col])

        #change max if needed
        if count > maxValue:
            maxValue = count
            maxIndex = col

    return maxIndex

#helper
def getCount(Mat,col):
    numRow = len(Mat)
    count = 0

    for row in range(numRow):
        count+= Mat[row][col] == 1

    return count

#correct answer is column index 2!
print("column index %d has the most non-zero values" % biggestCol(Mat))
```