Recitation 2
Decision Trees

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

02/11/2021

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 and height of a node examples
2. In class coding and explanation of Depth First Traversal in Python.

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

# A function to do preorder tree traversal
def printPreorder(root):

    if root is not None:
```

```
        # First print the data of node
        print(root.val, "\t",end="")

        # Then recurse on left child
        printPreorder(root.left)

        # Finally recurse on right child
        printPreorder(root.right)


# A function to do inorder tree traversal
def printInorder(root):

    if root is not None:

        # First recur on left child
        printInorder(root.left)

        # then print the data of node
        print(root.val, "\t",end="")

        # now recur on right child
        printInorder(root.right)



# A function to do postorder tree traversal
def printPostorder(root):

    if root is not None:
        # First recurse on left child
        printPostorder(root.left)

        # then recurse on right child
        printPostorder(root.right)

        # now print the data of node
        print(root.val, "\t",end="")

# Main body of the program
root = Node(1)
root.left     = Node(2)
root.right    = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)

print("\n")
```

```
input("press any key to display Preorder traversal")

print ("Preorder traversal of binary tree is: ")
printPreorder(root)

print("\n")

input("press any key to display Inorder traversal")

print ("Inorder traversal of binary tree is")
printInorder(root)

print("\n")

input("press any key to display Postorder traversal")
print ("Postorder traversal of binary tree is")
printPostorder(root)

print("\n")
```

Code Output

```
Preorder traversal of binary tree is:


Inorder traversal of binary tree is


Postorder traversal of 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

- Calculate the entropy of tossing a fair coin.

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

- Calculate the entropy of a fair dice roll.

- 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.

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? What are the inputs and outputs?

2. How should we represent our decision tree? With which data structures?

3. At each node of the tree, what do we need to store?

4. At each node of the tree, what do we need to do?
5. What are some edge cases we need to think about?

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

```
#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))
```