Recitation 2
Decision Trees

10-601: Introduction to Machine Learning

01/25/2019 (Updated on 01/27/2019)

1 Programming: Tree Structures and Algorithms

Topics Covered:

- Linear and Non linear data structures
- 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)

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 inorder tree traversal
def printInorder(root):
```

```
    if root:

        # 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:

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

        # the recur on right child
        printPostorder(root.right)

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


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

    if root:

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

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

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


# 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")

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

print("\n")

print ("\nInorder traversal of binary tree is")
printInorder(root)

print("\n")

print ("\nPostorder traversal of binary tree is")
printPostorder(root)

print("\n")
```

Code Output

```
Preorder traversal of binary tree is:
1  2  4  5  3


Inorder traversal of binary tree is
4  2  5  1  3


Postorder traversal of binary tree is
4  5  2  3  1
```

2 Programming: Debugging w/ Trees

ipdb and common commands

- `import ipdb` then `ipdb.set_trace()`
- n (next)
- ENTER (repeat previous)
- q (quit)

- p variable (print value)
- c (continue)
- l (list where you are)
- s (step into subroutine)
- r (continue until the end of the subroutine)
- ! python command

Real Practice

- Now that we've seen the basics of traversing a tree, let's try to recursively build one (pre-order)
- In this (extremely contrived) example, we will be building part of the Pokemon ancestry tree (see Chalkboard)
- We will use the following list to denote the structure of the tree:

```
mother = "Arceus"
all_pokemon = ["Lugia", "Articuno", "Stop", "Zapdos", "Stop", "Ho-oh"]
```

- We will try to debug the following (pseudo) implementation of a recursive build. (Note to students: you can assume that "Arceus" has been set as the "root" value and that the function below is called with "root" as your input for "node")

Buggy Code

```
def buildTree(node,new_data):
    if new_data is "Stop":
        delete first element of new_data
        return

    else:
        if node.right is None:
            node.right becomes a new node with val = new_data[0]
            delete first element of new_data
            buildTree(node.right,new_data)

        if node.left is None:
            node.left becomes a new node with val = new_data[0]
            delete first element of new_data
            buildTree(node.left,new_data)

        else: return
```

- The output of this code will be:

```
Traceback (most recent call last):
  File "tree_traversal_all_bugs.py", line 82, in <module>
    buildTree(root, all_pokemon)
  File "tree_traversal_all_bugs.py", line 67, in buildTree
    buildTree(node.right,new_data)
  File "tree_traversal_all_bugs.py", line 67, in buildTree
    buildTree(node.right,new_data)
  File "tree_traversal_all_bugs.py", line 67, in buildTree
    buildTree(node.right,new_data)
  [Previous line repeated 4 more times]
  File "tree_traversal_all_bugs.py", line 65, in buildTree
    node.right = Node(new_data[0])
IndexError: list index out of range
```

1. What causes the error message?
2. Where do we think this is a problem in our code?
3. How can we verify our suspicion?
4. Where should we set our ipdb trace (i.e. breakpoint) or a print statement?

After we fix the first bug, our next output is:

```
"Preorder traversal of binary tree is:"
"Arceus  Ho-oh  Lugia   Zapdos  Articuno"

"Correct tree preorder:"
"Arceus  Lugia  Articuno     Zapdos  Ho-oh"
```

1. Why is our tree order wrong?
2. Where do we think this is a problem in our code?
3. How can we verify our suspicion?
4. Where should we set our ipdb trace (i.e. breakpoint) or a print statement?

3 ML Concepts: Information Theory

Definitions:

- $H(Y) = -\sum_{i=1}^{n} P(Y=i) \log_2 P(Y=i)$
- $H(Y|X=v) = -\sum_{i=1}^{n} P(Y=i|X=v) \log_2 P(Y=i|X=v)$
- $H(Y|X) = \sum_{v \in values(X)} P(X=v) H(Y|X=v)$

- $I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$

Warm Up Exercise:

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

Information Theory 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 information gain 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.

4 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?