Recitation 2
Decision Trees

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

09/11/2020

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
```

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 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.
  This is the average surprisal from each flip.

  $H(X) = -p(heads) \log_2(p(heads)) - p(tails) \log_2(p(tails))$
  $= -\frac{1}{2} \log_2(\frac{1}{2}) - \frac{1}{2} \log_2(\frac{1}{2}) = 1$

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

  $= -0 * \log_2(0) - 1 \log_2(1) = 0$
  In other words we are never surprised by any flip. It's always tails.

- Calculate the entropy of a fair dice roll.
  $H(X) = -\sum_{x=1}^{6} (\frac{1}{6}) \log_2(\frac{1}{6}) = \log_2(6)$

- When is the mutual information I(X;Y) = 0?
  $I(X;Y) = H(X) - H(X \mid Y)$
  $I(X;Y)$ is 0 if and only if X and Y are independent.
  Mathematically, $H(Y \mid X) = H(Y)$ making $I(X;Y)$ go to 0.
  Intuitively, this is because if X and Y are independent, knowing one tells you nothing about the other and vice versa, so their mutual information is 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.

   H(Y) = -$\frac{6}{8} * log(\frac{6}{8}) - \frac{2}{8} * log_2(\frac{2}{8}) \approx 0.811$

   For attribute X_1,

   - $H(Y \mid X_1 = sunny) = -[\frac{1}{3} * log_2(\frac{1}{3}) + \frac{2}{3} * log_2(\frac{2}{3})] \approx 0.918$
   - $H(Y \mid X_1 = rain) = 0$
   - $H(Y \mid X_1 = overcast) = 0$

   $\implies H(Y \mid X_1) = [\frac{3}{8} * 0.918 + \frac{3}{8} * 0 + \frac{2}{8} * 0] \approx 0.344$
   $\implies I(Y; X_1) \approx 0.811 - 0.344 = 0.467$

   For attribute X_2,

   - $H(Y \mid X_2 = hot) = -[\frac{1}{3} * log_2(\frac{1}{3}) + \frac{2}{3} * log_2(\frac{2}{3})] \approx 0.918$
   - $H(Y \mid X_2 = cool) = 0$
   - $H(Y \mid X_2 = mild) = -[\frac{3}{4} * log_2(\frac{3}{4}) + \frac{1}{4} * log_2(\frac{1}{4})] \approx 0.811$

   $\implies H(Y \mid X_2) = [\frac{3}{8} * 0.918 + \frac{1}{8} * 0 + \frac{4}{8} * 0.811] \approx 0.75$
   $\implies I(Y; X_2) \approx 0.811 - 0.75 = 0.061$

   For attribute X_3,

   - $H(Y \mid X_3 = high) = -[\frac{1}{2} * log_2(\frac{1}{2}) + \frac{1}{2} * log_2(\frac{1}{2})] = 1$
   - $H(Y \mid X_2 = normal) = 0$

   $\implies H(Y \mid X_3) = [\frac{4}{8} * 1.0 + \frac{4}{8} * 0] = 0.5$
   $\implies I(Y; X_3) \approx 0.811 - 0.5 = 0.311$

   Since splitting on attribute X_1 gives the highest mutual information, the root node is X_1.

2. Calculate what the next split should be.
From the above part, as we can see that the sub-datasets $\mathcal{D}_{(X_1=rain)}$ and $\mathcal{D}_{(X_1=overcast)}$ are pure, there will be no further splitting on those and we will place a leaf node with label assignment decided by majority vote classifier. So, we need to split only on the sub-dataset $\mathcal{D}_{(X_1=sunny)}$. Now, we will use only $\mathcal{D}_{(X_1=sunny)}$ to estimate the probabilities for the next split.

$H(Y) = -\frac{1}{3} * log_2(\frac{1}{3}) - \frac{2}{3} * log_2(\frac{2}{3}) \approx 0.918$

For attribute X_2,

- $H(Y \mid X_2 = hot) = 0$
- $H(Y \mid X_2 = cool) = 0$
- $H(Y \mid X_2 = mild) = -[\frac{1}{2} * log_2(\frac{1}{2}) + \frac{1}{2} * log_2(\frac{1}{2})] = 1$

$\implies H(Y \mid X_2) = [\frac{2}{3} * 1.0 + \frac{1}{3} * 0] \approx 0.67$
$\implies I(Y; X_2) \approx 0.918 - 0.67 \approx 0.25$

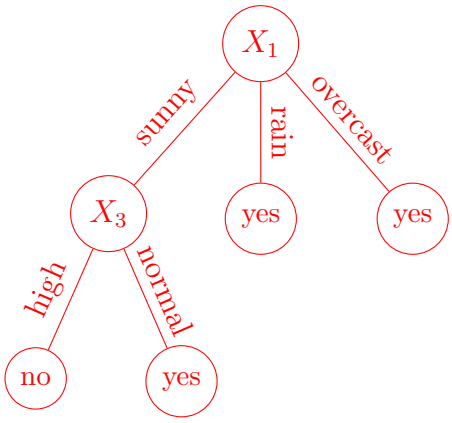
For attribute X_3,

- $H(Y \mid X_3 = high) = 0$
- $H(Y \mid X_3 = normal) = 0$

$\implies H(Y \mid X_3) = [\frac{2}{3} * 0 + \frac{1}{3} * 0] = 0$
$\implies I(Y; X_3) \approx 0.918$

We split using attribute X_3 as it gives the highest mutual information.

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?
   The task: Given a set of train data, test data, and max depth of a tree, we want to learn a decision tree classifier.
2. How should we represent our decision tree? With which data structures?
   There are different ways one can represent the decision tree. Arguably the easiest" or "most intuitive" would be in a tree data structure introduced in section 1.
3. At each node of the tree, what do we need to store?
   Some of the most basic things we want to store:
   - The attribute to split at the node
   - Attributes that have been used
   - The input data at that node
   - The left and right child nodes
   - Node depth

   Note that this list (and the list on the next question) is not exhaustive. One might want to store other items that can aid the implementation.
4. At each node of the tree, what do we need to do?
   - Check "stopping criteria." One being the node depth reaching the max depth (What other stopping criteria is there?). What happens when we reach the stopping criteria?
   - Calculate entropy and mutual information for the non-used attributes and select the best attribute to split
   - Split the data based on the selected attributes
5. What are some edge cases we need to think about?
   - What happens if max depth equals 0?
   - What happens if max depth is greater than the number of attribute?

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

```
#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))
```

Solution: There are two errors:

1. oppositeRow should be set to rows-i-1 as it will be out of bounds otherwise
2. Creating a 2d list with $new = [[0 * cols] * rows$ will result in aliasing.

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

    new = [ ([0] * cols) for row in range(rows) ]

    for i in range(rows):
        for j in range(cols):
            oppositeRow = rows-i-1
            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))
```

Solution: There are two errors:

1. we should be calling getCount instead of np.count_nonzero
2. getCount should be checking if the cell is not equal to 0

```
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 = getCount(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] != 0

    return count

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