Recitation 2
Decision Trees

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

01/27/2023

1 Programming: Tree Structures and Algorithms

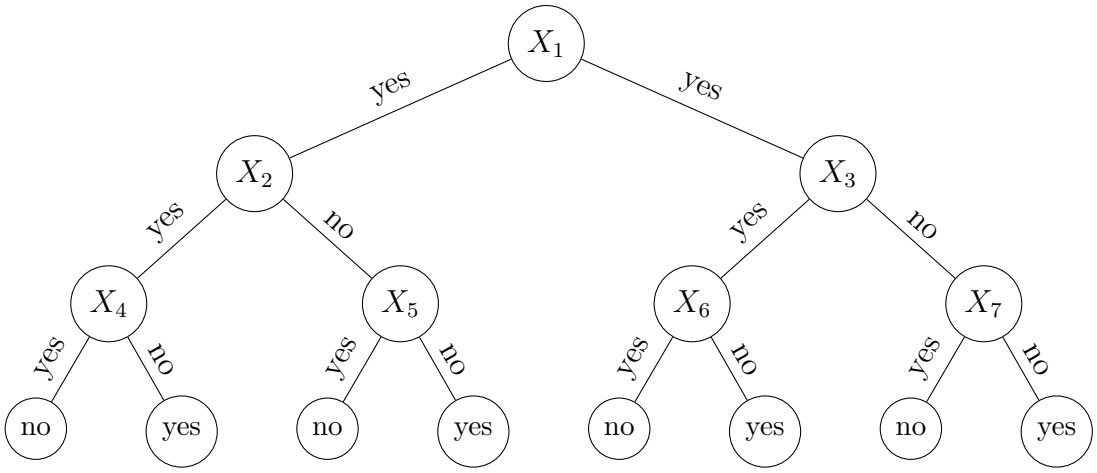
Topics Covered:

- 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:

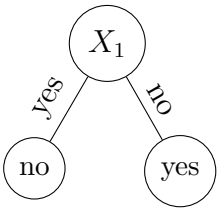
1. Depth of a tree definition
   The depth of a tree is the length (number of edges) of the longest path from a root to a leaf.
2. Depth of a node definition
   The depth of a node is the number of edges between the root and the given node.

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



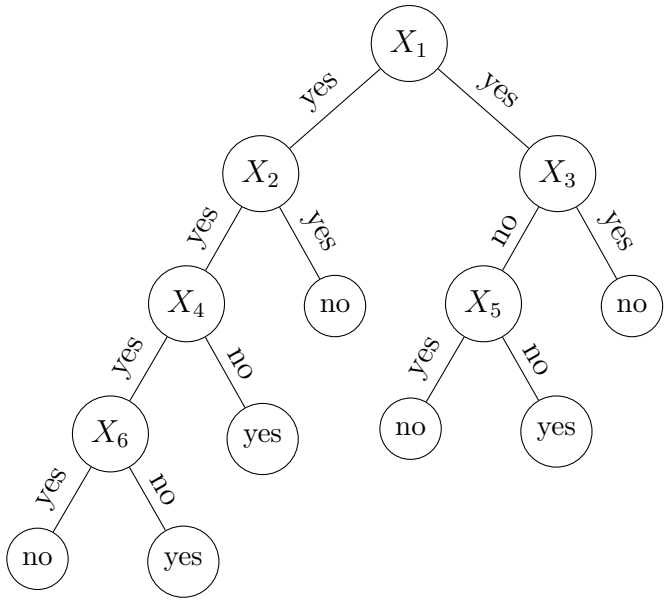
The depth of tree A is 3 and the depth of node X_4 is 2.

4. What is the depth of tree B?



The depth of tree B is 1 (decision stump).

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



The depth of tree C is 4. The depth of node X_1 is 0 and the depth of X_5 is 2.

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:

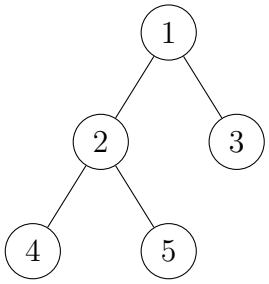
- traversal1() is
- traversal2() is
- traversal3() is

Traversal1 of binary tree is 4 5 2 3 1

Traversal2 of binary tree is: 1 2 4 5 3

Traversal3 of binary tree is 4 2 5 1 3

- traversal1() is Post-Order.
- traversal2() is Pre-Order.
- traversal3() is In-Order.



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.
   This is the average surprisal from each flip.

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

2. 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.

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

4. 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.
   - 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?

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

- $I(Y; X_1) = 0.467$

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

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

- $I(Y; X_2) = 0.061$

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

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

- $I(Y; X_3) = 0.311$

  For attribute X_3,

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

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

- Split on X_1 at the root node

  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,

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

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

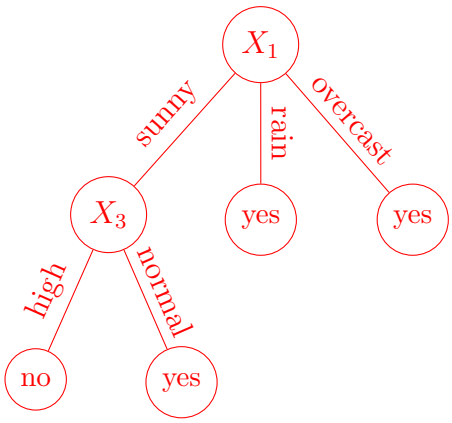
   For attribute X_3,

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

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

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?
   The task: Given a set of train data, test data, and max depth of a tree, we want to do the following:
   1. Use the train data to learn a decision tree classifier.
   2. Use our trained classifier to predict the labels of both the train and the test data
   3. Calculate the error rates for our classifier on the train and test data
2. What are the inputs and outputs at training time? At testing time?
   For training inputs:
   - The max-depth of the tree
   - The training data

   For training outputs:
   - A fully trained decision tree

   For testing inputs:
   - A new dataset in the same format as the training data

   For testing outputs:
   - A prediction for every input row of the dataset given
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
   - The subset of data at a given 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. What do we need to do at training time?
   - Check "stopping criteria" (e.g. if max depth has been reached, or if the node is pure). If either are true, run majority vote at the node.
   - 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 happens if max depth is 0?

   Majority Vote

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

   Stop growing the tree when all attributes are used.

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))
```

Solution: There are two errors:

1. `new_index` 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_rows(original):
    rows = len(original)
    cols = len(original[0])

    new = [[0 for _ in cols] for _ in rows]

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

Solution: There are two errors:

1. we should be calling `get_count` instead of `np.count_nonzero` (or use an `np.array`)
2. `get_count` should be checking if the cell is not equal to 0

3. `count >= max_value` will pick the largest index

```
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 = get_count(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
```