Recitation 1: Decision Trees

10-301/10-601 Introduction to Machine Learning (Summer 2024)
http://www.cs.cmu.edu/~hchai2/courses/10601
Released: May 16th, 2023
HW Due Date: May 23rd, 2023
TAs: Alex, Doris, Zhifei, Zoe, and Neural the Narwhal

1 Decision Trees

1.1 Information Theory and Tree Terminology

1. Calculate the entropy of tossing a fair coin.

$H(X)$	Work
1	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:* $0 \cdot \log_2(0) = 0$.

$H(X)$	Work
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. In your own words, what is the definition of the depth of a tree?

Answer

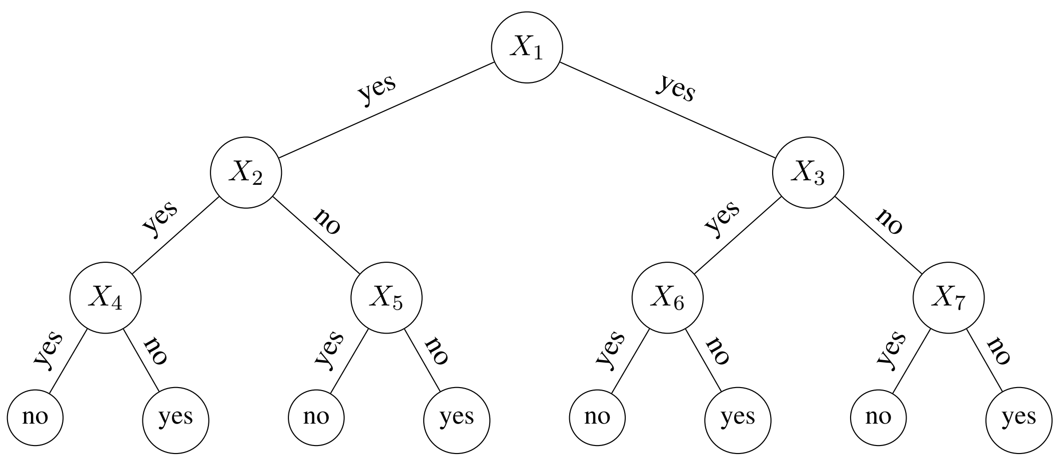
The depth of a tree is the length (number of edges) of the longest path from a root to a leaf.

4. In your own words, what is the definition of the depth of a node?

Answer

The depth of a node is the number of edges between the root and the given node.

5. What is the depth of the tree below? What is the depth of node X_4 in the tree below?



Answer

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

1.2 Example Tree

The following dataset D consists of 8 examples, each with 3 attributes, (A, B, C), and a label, Y.

A	B	C	Y
1	2	0	1
0	1	0	0
0	0	1	0
0	2	0	1
1	1	0	1
1	0	1	0
1	2	1	0
1	1	0	1

Use the data above to answer the following questions.

- *All calculations should be done without rounding!* After you have finished all of your calculations, write your rounded solutions in the boxes below.
- Unless otherwise noted, numeric solutions should include 4 digits of precision (e.g. 0.1234).
- Note that the dataset contains duplicate rows; treat each of these as their own example, do not remove duplicate rows.

1. What is the entropy of Y in bits, $H(Y)$? In this and subsequent questions, when we request the units in *bits*, this simply means that you need to use $\log$ base 2 in your calculations.[1] (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$H(Y)$
1

Work
$H(Y) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$

[1] If instead you used $\log$ base e, the units would be *nats*; $\log$ base 10 gives *bats*.

2. What is the mutual information of Y and A in bits, $I(Y;A)$? (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$I(Y;A)$	Work
0.0488	$I(Y;A) = H(Y) - H(Y\|A)$ $= H(Y) - [\frac{3}{8}(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}) + \frac{5}{8}(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5})]$ $= 1 - 0.951205 = 0.048795 \approx 0.0488$

3. What is the mutual information of Y and B in bits, $I(Y;B)$? (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$I(Y;B)$	Work
0.3113	$I(Y;B) = H(Y) - H(Y\|B)$ $= H(Y) - [\frac{2}{8}(-1\log_2 1) + \frac{3}{8}(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}) + \frac{3}{8}(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3})]$ $= 1 - 0.688722 = 0.311278 \approx 0.3113$

4. What is the mutual information of Y and C in bits, $I(Y;C)$? (Please include one number rounded to the fourth decimal place, e.g. 0.1234)

$I(Y;C)$	Work
0.5488	$I(Y;C) = H(Y) - H(Y\|C)$ $= H(Y) - [\frac{5}{8}(-\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}) + \frac{3}{8}(-1\log_2 1)]$ $= 1 - 0.451205 = 0.548795 \approx 0.5488$

5. **Select one:** Consider the dataset given above. Which attribute (A, B, or C) would a decision tree algorithm pick first to branch on, if its splitting criterion is mutual information?

   ◯ A

   ◯ B

   ◯ C

   C, because this has the highest mutual information.

6. **Select one:** Consider the dataset given above. After making the first split, which attribute would the algorithm pick to branch on next, if the splitting criterion is mutual information? (*Hint:* Notice that this question correctly presupposes that there is *exactly one* second attribute.)

   ◯ A

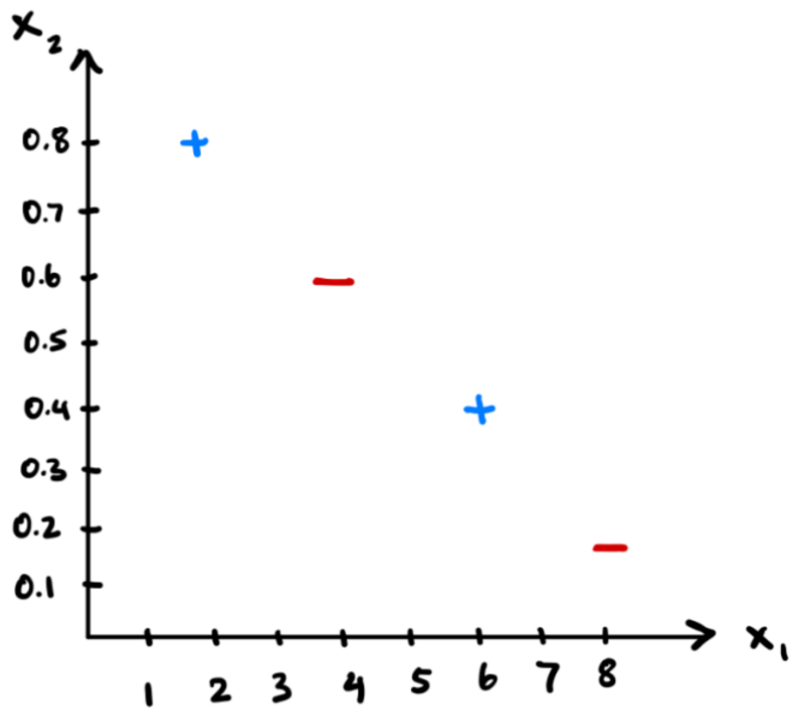
   ◯ B

   ◯ C

   A, because continuing the calculations for the decision tree shows that A has higher mutual information with Y when compared to B.

1.3 Real-Valued Trees

Consider the following training data. The red '−' marks represent $Y = 0$ and the blue '+' marks represent $Y = 1$.



1. What is the entropy of Y in bits?

Answer
1

2. What is the mutual information if we are splitting on $X_1 < 5$? Show all of your work in the provided space. (Please include one number rounded to the third decimal place, e.g. 0.123)

Answer	Work
0	$1-(\frac{2}{4}*(-\frac{1}{2}*\log_2(\frac{1}{2})-\frac{1}{2}*\log_2(\frac{1}{2}))+\frac{2}{4}*(-\frac{1}{2}*\log_2(\frac{1}{2})-\frac{1}{2}*\log_2(\frac{1}{2}))) = 0$

3. **True or False:** It is possible to have a decision tree with zero training error for this dataset. Assume only binary splits and attributes selected with replacement.

◯ True

◯ False

True

2 Programming

1. In-class coding and explanation of Depth First Traversal in Python.
   Link to the code:
   https://colab.research.google.com/drive/1dFNwBk_Ddr08DbNCeDGQEz1hT_QLGKc8?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)
```

```python
    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:

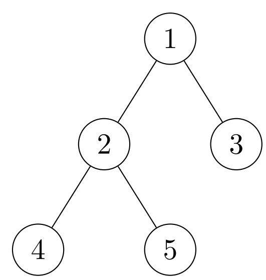
- traversal1() is
- traversal2() is
- traversal3() is

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
```

2.1 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/1dFNwBk_Ddr08DbNCeDGQEz1hT_QLGKc8?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

```python
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

```python
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
```