Recitation 2
Decision Trees

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

09/05/2025

1 Programming: Tree Structures and Algorithms

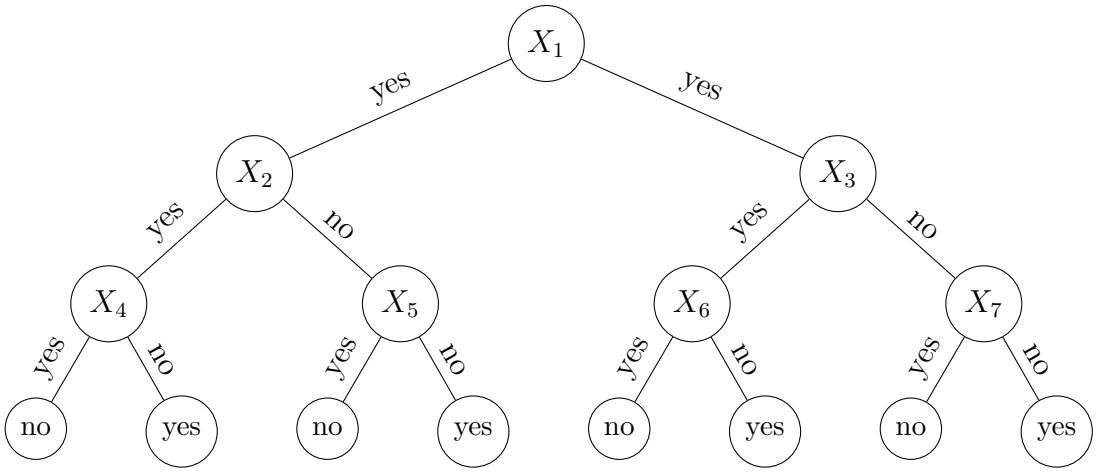
Topics Covered:

- 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:

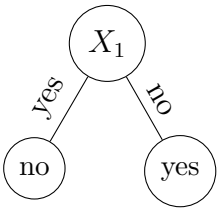
1. Depth of a tree definition
   The depth of a tree is the length (number of edges) of the longest path from a root to a leaf.
2. Depth of a node definition
   The depth of a node is the number of edges between the root and the given node.

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



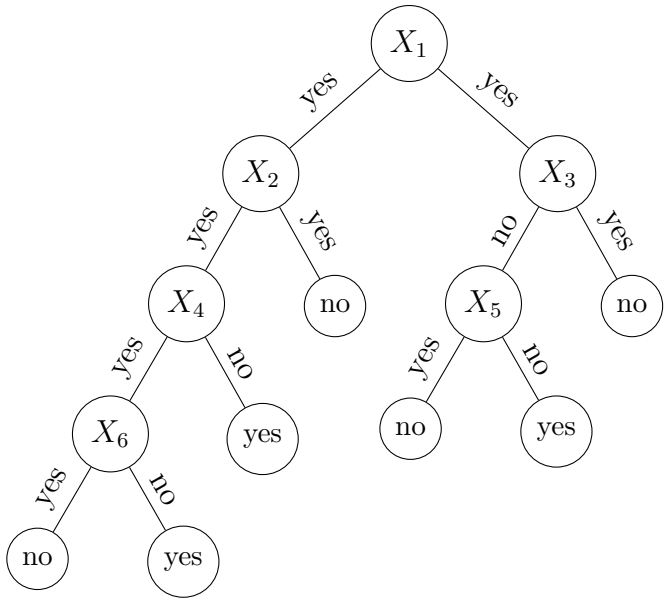
The depth of tree A is 3 and the depth of node X_4 is 2.

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/1VvNZUQ4ZikQXcvWL-EY1OPnGdye2P024?usp=sharing

   DFS Tree Traversals and Printing

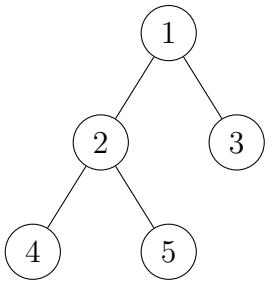
```
# This class represents an individual node
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

def traversal1(root):
    if root is not None:
        # First print the data of node
        print(root.val, end='\t')
        # Then recurse on left child
        traversal1(root.left)
        # Finally recurse on right child
        traversal1(root.right)

def traversal2(root, __[a]__):
    if root is not None:
        # First recurse on left child
        traversal2(root.left, __[b]__)
        # Then print the data of node
        print(f'({root.val}, {__[c]__})')
        # Now recurse on right child
        traversal2(root.right, __[d]__)

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, 0)
```



First identify which traversal function is pre-order, in-order, or post-order DFS:

- traversal1() is
- traversal2() is

Fill-in-the-Blanks: Next, fill in the code for `traversal2`.

Code Output:

```
traversal1 of the binary tree is:


traversal2 of the binary tree is:
```

Identifying traversals:

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

```
def traversal2(root, __depth__):
    if root is not None:
        # First recurse on left child
        traversal2(root.left, __depth+1__)
        # then print the data of node
        print(f'({root.val}, {__depth__})')
        # now recurse on right child
        traversal2(root.right, __depth+1__)
```

traversal1 of the binary tree is:
1 2 4 5 3

traversal2 of the binary tree is:
(4, 2)
(2, 1)
(5, 2)
(1, 0)
(3, 1)

2 The Need For Speed: Vectorization and Numpy

Performing mathematical operations on vectors and matrices is ubiquitous in most machine learning algorithms. Whether it's a simple similarity measure that works by calculating the dot product between two vectors, or deep neural networks, they all involve repeated matrix operations. This makes it imperative that our underlying code design to perform matrix operations is efficient.

2.1 The Perils of Python

While Python is widely the language of choice for machine learning researchers across the globe (thanks to the speed of development and code readability it offers and the support it enjoys from the open-source community), Python as a high-level language on average is much slower than a lower level language like C++. To combat this, libraries like numpy and scipy implement most of the back-end operations they perform in C/C++, while providing wrappers in Python to be able to call underlying C code seamlessly from a Python script.

2.2 Speed Comparison: Numpy and Python

We highly recommend you to use *numpy* extensively in this course, it will be difficult to pass the programming portion of Homework 4 without writing most of your matrix operations in numpy. In this section, we'll see why.

Consider you have two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. To see how similar they are, as measured by the cosine angle between them, you want to compute their dot product. This translates to the following operation:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + ... + a_n b_n$$

When translated to code, notice how the dot product in NumPy is a whopping 100x faster than the native Python!

```
from timeit import timeit
import numpy as np
import array

VECTOR_SIZE = int(1e8)

# NumPy arrays
a = np.random.rand(VECTOR_SIZE)
b = np.random.rand(VECTOR_SIZE)

# Python arrays
aArr = array.array('d', a)
bArr = array.array('d', b)
```

```
def test_np():
    return np.dot(a, b)

# faster than multiprocessing, python lists, or numpy arrays with
   python loops
# faster than using a range and indexing
def test_py_arr():
    return sum(x * y for x, y in zip(aArr, bArr))

def time_dot_product(f):
    return timeit(f, setup=f, number=5) / 5



if __name__ == "__main__":
    print(f"NumPy = {time_dot_product(test_np):.2f}")  # 0.05s
    print(f"Python on an array = {time_dot_product(test_py_arr):.2f}")
        # 5.45s
```

2.3 Useful Numpy Operations

Some operations in numpy that you will find really useful in your assignments are:

- np.matmul: Matrix multiplication of two matrices
- np.unique: Returns unique elements along an axis.
- np.hstack: Stack two arrays horizontally (column-wise)
- np.expand_dims: Convert a row vector of size n into a matrix of size $n * 1$ or $1 * n$
- np.log, np.sum, np.exp, @, .T, and so on...

You can read C vs. Python for more details, and you can also read these two tutorials (beginner, intermediate) from the official numpy website. For instance, understanding broadcasting is recommended. It will help you debug the shape errors you might face in all future homeworks.

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 do we need to do at testing time?
   - Use the classifier trained using the training data. Do NOT train on test data.
   - Predict labels on the train data and test data using the trained classifer.
   - Write predictions and metrics to output files as necessary
6. What happens if max depth is 0?

   Majority Vote
7. What happens if max depth is greater than the number of attributes?

   Stop growing the tree when all attributes are used.

4 ML Concepts: Mutual Information

Information Theory Definitions:

- $H(Y) = -\sum_{y \in values(Y)} P(Y = y) \log_2 P(Y = y)$
- $H(Y \mid X = x) = -\sum_{y \in values(Y)} P(Y = y|X = x) \log_2 P(Y = y|X = x)$
- $H(Y \mid X) = \sum_{x \in values(X)} P(X = x) H(Y \mid X = x)$
- $I(Y; X) = H(Y) - H(Y \mid X) = H(X) - H(X \mid Y)$

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 $I(Y; X) = 0$?

$$I(Y;X) = H(Y) - H(Y \mid X) = H(X) - H(X \mid Y)$$

$I(Y;X)$ is 0 if and only if X and Y are independent.
Mathematically, $H(Y \mid X) = H(Y)$ making $I(Y;X)$ 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.