Decision Trees Pt. 1: Maximizing Information Gain

Eckel, TJHSST AI2, Spring 2022

Background & Explanation

It is time to construct decision trees! But not just any decision trees: we want decision trees that are as *small* as possible. How do we do this? We maximize the *entropy lost*, or in other words, maximize the *information gained* from each choice in the tree.

Let's do an example. Consider the PlayTennis training set on the right.

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

The measure of the entropy of this data set's outcomes is the negated sum over each possible outcome of the probability of that outcome times the information content of that outcome, which is the log base 2 of the probability of that outcome.

In other words, the starting entropy of this data set is:

$$-1\left(\frac{5}{14}\cdot log_2\frac{5}{14}+\frac{9}{14}\cdot log_2\frac{9}{14}\right) = .9403$$

…since "No" has a 5/14 chance of happening and "Yes" has a 9/14 chance of happening.

(The derivation and explanation for this formula is in the Entropy activity's textbook reading; I highly recommend revisiting that reading and the example problems & solutions from our deep dive until the above calculation is clear!)

So what we want is to reduce the entropy as much as possible by making the best choice of feature each time. (Recall that any data set with only one outcome has an entropy of zero, so we want to minimize entropy as we go.)

So: how can we measure this? What does it mean to maximize information gained?

The basic idea here is that we want to be able to calculate the *expected entropy* if we know the value of a particular feature. Think of this like *expected value* in a probability problem – the odds of each outcome * the value of that outcome. We will find the odds of each value * the entropy of the corresponding data set.

Note: we must make a particular assumption here, which is that the chances of each value of a feature are approximately equal in reality to their proportion in the training set, but given this quite reasonable assumption, the calculation is straightforward.

To find the *expected entropy* of knowing the value of, say, **Outlook**, we would consider each possible value of **Outlook**, consider only the feature vectors in the table with that particular value for **Outlook**, find the entropy of *that* data set, and multiply it by the likelihood of that value. For instance, we would look at only the rows of the table with a "Sunny" outlook, find their entropy, and multiply by the odds of a "Sunny" outlook occurring.

In the case of **Outlook**, considering "Sunny", "Overcast", and "Rainy" in that order, the calculation is:

$$\frac{5}{14}\cdot .9710+\frac{4}{14}\cdot 0+\frac{5}{14}\cdot .9710 = .6936$$

You should verify each entropy calculation I made here; make sure you understand where these numbers come from. This math **MUST** make sense for this assignment to work; don't turn the page until you're clear on this calculation!

Warm-Up Exercise: Make sure you can correctly calculate information gain

On the previous page, you'll note that .6936 is a much smaller value than .9403. Does that make it a good choice for the first node in our tree? Well, that depends on the expected entropy we'd get from knowing the value of each other feature…

By choosing **Outlook**, we would have an *information gain* of .9403 - .6936 = .2467.

Either by hand or using code, verify the information gained from each choice of feature in the PlayTennis dataset.

You should get:

Outlook	.2467
Temperature	.0292
Humidity	.1518
Windy	.0481

So: which feature should comprise our first node, and why? Outlook, for sure – it gains us the most information!

Essential Task: Generate ideal decision trees

If you understand how to use the above process to decide on the *first* node of the tree, then the entire algorithm should now be clear: simply a recursive application of this process.

The recursive algorithm should:

1. Consider each feature in turn. Find the information gain from differentiating based on that feature.
2. Choose the feature with the highest information gain.
3. Split the dataset into smaller datasets based on the possible values of the chosen feature.
4. If any of the resulting datasets has an entropy of 0, then it is a leaf.
5. Otherwise, recur on any smaller dataset with an entropy > 0.

The goal is to turn the PlayTennis dataset on the front into *precisely* the tree below, output in *precisely* the format shown below. (**Note**: the *ordering* of the options may be different, but the same *flow* must be present. So, **Outlook** must be the first choice, a rainy outlook must then split on **Wind**, etc. I've sorted the options within each feature alphabetically; if you do the same, it may be easier to verify.) On the right, my tree also prints out the information gain and entropy at each step for you to use in debugging & verification.

```
* Outlook?
  * Overcast --> Yes
  * Rain
    * Wind?
      * Strong --> No
      * Weak --> Yes
  * Sunny
    * Humidity?
      * High --> No
      * Normal --> Yes
```

```
* Starting Entropy: 0.9402859586706311
* Outlook? (information gain: 0.24674981977443933)
  * Overcast --> Yes
  * Rain (with current entropy 0.9709505944546686)
    * Wind? (information gain: 0.9709505944546686)
      * Strong --> No
      * Weak --> Yes
  * Sunny (with current entropy 0.9709505944546686)
    * Humidity? (information gain: 0.9709505944546686)
      * High --> No
      * Normal --> Yes
```

Don't move on until you can verify this tree and those numbers!

An important note: notice in the run on the right that the entropy at "Rain" is actually *higher* than the beginning entropy. Why is this ok? Well, since it's a leaf, the entropy at "Overcast" is zero! So even though two possibilities for **Outlook** *increase* the entropy of the resulting data set, that one option lowers the *overall* or *average* expected entropy across all three options quite a lot!

So, your job in this assignment is to generate these trees and print them like I've shown above. But it's worth mentioning that in the next assignment, we'll need to use this tree to classify new observations; that is, you will have a previously generated decision tree and a new data point that you haven't used to build that tree, and you'll need to follow the branches of the tree based on the new data point until you get to a classification for that new input. Keep that in mind as well as you decide how to do this! If you'd like some hints for how to write this, I have implementation advice on the next page; if you'd prefer the challenge of figuring it out yourself, though, feel free to ignore it.

On the page where you downloaded this assignment, you'll find several data sets. I'd like you to build trees based on three of them. For further data verification, this is the tree generated from the WillWait dataset:

```
* Patrons?
  * Full
    * Hungry?
      * No --> No
      * Yes
        * Type?
          * Burger --> Yes
          * Italian --> No
          * Thai
            * Friday/Saturday?
              * No --> No
              * Yes --> Yes
  * None --> No
  * Some --> Yes
```

```
* Starting Entropy: 1.0
* Patrons? (information gain: 0.5408520829727552)
  * Full (with current entropy 0.9182958340544896)
    * Hungry? (information gain: 0.2516291673878229)
      * No --> No
      * Yes (with current entropy 1.0)
        * Type? (information gain: 0.5)
          * Burger --> Yes
          * Italian --> No
          * Thai (with current entropy 1.0)
            * Friday/Saturday? (information gain: 1.0)
              * No --> No
              * Yes --> Yes
  * None --> No
  * Some --> Yes
```

Your specific task:

1) Verify that the PlayTennis & WillWait datasets produce precisely the trees shown above.
2) Generate a tree using the ToxicMushroom dataset (which answers the question "should I eat this mushroom?"). This is a famous problem in early machine learning research – the Audubon Field Guide, from which this data comes, has certain rules for plants to avoid (ie, "leaves of three, let it be" for poison ivy) but says that there is no 100% accurate way to decide which mushrooms are poisonous or not. So, someone put all the data into a data set, and decided to find out if that was true.
3) Find another student who has also completed this assignment and verify that you both produce the same decision tree**. Examine carefully; be sure. Then think about this in the context of the original question – is this tree a useful guide for people finding mushrooms in the wild? Should Audubon publish it? What do you think? **Send me a message on Mattermost** that says "I checked with _____ and our mushroom trees are the same", and then your opinion on the usefulness of your output. **Each person** should **individually** send me a message. You can check more than one person's answer against your own; in that case, send me a message **each time you do so** verifying the check, but you only have to send me your opinion of the output once.

   **Remember that the *ordering* of the options within a particular feature doesn't have to be the same, but the *flow* should be identical. If you'd like to make checking easier, you can sort the options within a particular feature in alphabetical / numerical order as I have done in my examples above.

Implementation advice

If you want a challenge, you don't need to read any of this – everything you need to know to complete this assignment is in the sections above. But if you'd like, some FAQs / good advice:

- **Don't worry about efficiency.** The data sets we'll use, even the biggest ones, aren't big enough for speed to matter. Any solution you write will almost certainly be fine.
- **Split this into several helper functions and test them one by one.** In addition to the core recursive function, you probably want a function just to find the entropy of the outcomes of a particular data set, a function to reduce a data set based on a particular value of a particular feature, and a function to choose the feature on the data set with the greatest information gain.
- **Put some thought into how you want to store your tree.** Generating the tree, and getting the output given above, isn't trivial. You can code up a node class like we did in APCS (though note that the nodes only need to point to children; you'll never have to traverse *up* the tree), or you can use something like nested dictionaries (dictionaries of dictionaries), or any other way that makes sense to you. Again, remember that in the next assignment, we'll need to use this tree to classify new observations; that is, you will have a previously generated decision tree and a new data point that you haven't used to build that tree, and you'll need to follow the branches of the tree based on the new data point until you get to a classification for that new input. So you need to think about how to build the structure, how to print it, and also how to use it. One thing I do recommend is storing a "depth" value at each question and answer; as you print, at each node you can just print a string of two spaces multiplied by depth + your actual output for that node and get the indenting in your output for free!
- **Don't worry about removing features from the vectors as you use them.** Once you've split the data set on a feature, choosing that feature *again* will result in information gain of… zero! So it will never be chosen.

Specification for Decision Trees Part 1

Once you've messaged me on Mattermost, read the specification below carefully. Make sure your code follows it, and submit **a single python script** to the link on the course website.

This assignment is **complete** if:

- The "Name" field on the Dropbox submission form contains your **class period**, then your **last name**, then your **first name**, in that order.
- Your code accepts one command line argument – a .csv file like the ones provided for this assignment.
- Your code creates a text file with the ***exact file name*** `"treeout.txt"` where it saves the decision tree resulting from its run.
- Runtime should be less than 30 seconds (though I will be shocked if it is not!)

Fun Optional Thing: Draw the Mushroom Tree Nicely

Finally, this has nothing to do with improving your tree, just visualizing it – the pygraphviz package will draw lovely visualizations of trees for you. (“Graph” in this case refers to “graph theory”, **not** graphs like in algebra; “graph theory” is a mathematical branch dealing with nodes and connections, including trees.)

Pygraphviz doesn’t seem to be working on school laptops, so this isn’t an assignment for credit this year; it wouldn’t be fair. Just for fun. You may need to download and install additional software to get this to work; ask me about this if google doesn’t come through for you.

Here’s some minimal code to try, which should be instructive. Using this as an example, create a visualization of the deadly mushroom tree:

```python
import pygraphviz as pgv
G = pgv.AGraph(directed=True)
G.add_node(1, label="Age?")
G.add_node(2, label="Accept")
G.add_node(3, label="Reject")
G.add_edge(1,2, label = ">25")
G.add_edge(1,3, label="<= 25")
G.layout(prog="dot")
G.draw("minimal-tree.png")
```

Can you make something nicely formatted enough for, say, the Audubon Field Guide to use? Post your image in Town Square on Mattermost – let’s see what this looks like!