CS 3430: SciComp with Py
Assignment 10
Decision Trees, Entropy, Information Gain, Binary ID3

Vladimir Kulyukin
Department of Computer Science
Utah State University

March 29, 2020

1 Learning Objectives

1. Decision Trees
2. Entropy
3. Information Gain
4. Binary ID3

Introduction

In this assignment, we'll implement the Binary ID3 algorithm that learns decision trees from a set of examples, each of which is a set of attribute-value pairs. You'll save your solutions to Problem 2 in `bin_id3.py` and submit it in Canvas. The file `bin_id3_uts.py` contains my unit tests for this assignment.

The slide pdfs and screencasts for this assignment are available on Canvas under the announcements titled "CS 3430: S20: Lecture 20: Parts 02 and 03" and "CS 3430: S20: Lecture 20: Part 04."

Problem 1 (0 points)

Watch the three screencasts for Lecture 20 on Canvas: `CS3430_S20_Lec20_Part02.mp4`, `CS3430_S20_Lec20_Part03.mp4`, and `CS3430_S20_Lec20_Part04.mp4`. I know that some of you have left USU, which is quite understandable under the circumstances. When I come to my office every morning, I notice that our CS Department on the 4th floor of Old Main looks more and more like a ghost town. So, if you are not on campus and don't have access to broadband (or learn better from reading), read the pdf scan of Chapter 3 from Tom Mitchell's excellent book "Machine Learning" (included in the zip for this assignment) then read through `CS3430_S20_Decision_Trees_Part01.pdf` and `CS3430_S20_Decision_Trees_Part02.pdf`. You may also choose to read through the slides first and then go through Mitchell's text. **Don't feel like you have to do both.** If you read the slides and feel like you have a good grip on the material, go on to Problem 2. But, whatever materials you choose to read or watch, make sure that you're comfortable with the decision tree terminology and such concepts as entropy and information gain before proceeding to Problem 2.

I'm not sure what we'll do if they restrict campus access for the faculty before the end of the semester. The internet access at my house won't be able to handle screencast uploads. We may have to confine ourselves to text and pdf. But, we'll think of how to handle it if and when we'll get there.

By the way, if you're interested in classical machine learning methods, I highly recommend Dr. Mitchell's text. He does an excellent job presenting various machine learning methods and algorithms and gives lots of references and pointers for further research and individual projects alike.

I also recommend that you work on Problem 2 not in the order of the unit tests in `bin_id3_uts.py` but in the order of the unit tests specified in the next section. If necessary, put the unit tests in a separate file, then copy and paste them into `bin_id3_uts.py` one by one as you work on them.

The first bunch of unit tests in the next section makes you comfortable with the decision tree data structure. You don't have to implement anything for these tests. Just run them and see how it all works. The second bunch of unit tests asks you to implement proportion, entropy, and information gain. The final bunch of unit tests aims at the implementation of the Binary ID3 algorithm.

Problem 2 (4 points)

As you read through the sections below, keep in mind that our end objective is to implement the Binary ID3 algorithm on slides 4 and 5 of `CS3430_S20_Decision_Trees_Part02.pdf`.

Decision Tree Nodes

Fist, we need to get comfortable with the `id3_node` class in `bin_id3.py`, because decision trees are built out of `id3_node` objects

```
class id3_node(object):
    def __init__(self, lbl):
        self.__label = lbl
        self.__children = {}

    def set_label(self, lbl):
        self.__label = lbl

    def add_child(self, attrib_val, node):
        self.__children[attrib_val] = node

    def get_label(self):
        return self.__label

    def get_children(self):
        return self.__children

    def get_child(self, attrib_val):
        assert attrib_val in self.__children
        return self.__children[attrib_val]
```

Let's build the decision tree on slide 3 in `CS3430_S20_Decision_Trees_Part02.pdf` manually. We'll later build it automatically with the Binary ID3 algorithm. The code is in `bin_id3_uts.test_id3_ut02(self, tn=2)`. You don't have to implement anything to run this unit test. All the classes and methods for this unit test are implemented in `bin_id3.py`.

The label of each node, except for the leaf nodes, is an attribute. Recall that the Binary ID3 algorithm learns decision trees for binary concepts (e.g., `PlayTennis`) that have two possible values – `Yes` and `No`. Thus, the leaf nodes (i.e., the classes) have the labels 'Yes' or 'No.' Let's build two leaf nodes. The constants `PLUS` ('Yes') and `MINUS` ('No') are defined at the beginning of `bin_id3.py`.

```
yes_node = id3_node(PLUS)
assert yes_node.get_label() == PLUS
no_node = id3_node(PLUS)
assert no_node.get_label() == PLUS
```

Let's build the `Humidity` node. We create the node with the label 'Humidity' and then connect two children to it on the two attribute values of the attribute 'Humidity': 'High' and 'Normal.' After the children are connected to their parent, we call the method `bin_id3.display_id3_node()` on the parent node.

```
humidity_node = id3_node('Humidity')
assert humidity_node.get_label() == 'Humidity'
humidity_node.add_child('High', no_node)
humidity_node.add_child('Normal', yes_node)
assert humidity_node.get_child('High').get_label() == MINUS
assert humidity_node.get_child('Normal').get_label() == PLUS
assert len(humidity_node.get_children()) == 2
bin_id3.display_id3_node(humidity_node, '')
```

When we run this portion of `bin_id3_uts.test_id3_ut02(self, tn=2)`, we'll see the following output.

```
Humidity
        High
                No
        Normal
                Yes
```

It's straightforward to interpret this output. The `Humidity` node has two child nodes - `No` and `Yes`. The `No` node is connected to its parent (i.e., the `Humidity` node) via the link `High` whereas the `Yes` node is connected to its parent via the link `Normal`. Let me reiterate this point again to drive it home – the internal nodes of any decision tree built (or learned if you prefer the standard machine learning terminology) with the ID3 algorithm are attributes connected to their children via their attribute values. Of course, instead of using strings for attributes and their values, we can map them all to numbers. But, this is just an inconsequential implementational detail that we'll ignore in this assignment. In other words, we'll assume that attributes and their values are strings. The recursive structure of decision trees allows us to use these trees to classifiy new examples. More on this later. For now, let's build the `Wind` node and display it.

```
wind_node = id3_node('Wind')
assert wind_node.get_label() == 'Wind'
wind_node.add_child('Strong', no_node)
wind_node.add_child('Weak', yes_node)
assert wind_node.get_child('Strong').get_label() == MINUS
assert wind_node.get_child('Weak').get_label() == PLUS
assert len(wind_node.get_children()) == 2
bin_id3.display_id3_node(wind_node, '')
```

Running this portion of `bin_id3_uts.test_id3_ut02(self, tn=2)` produces the following output.

```
Wind
        Strong
                No
        Weak
                Yes
```

We finish building the decision tree on slide 3 in `CS3430_S20_Decision_Trees_Part02.pdf` by creating the root node `Outlook` and connecting it to its three child nodes via the appropriate attribute value links for the attribute 'Outlook': 'Sunny', 'Overcast', and 'Rain'.

```
outlook_node = id3_node('Outlook')
assert outlook_node.get_label() == 'Outlook'
outlook_node.add_child('Sunny', humidity_node)
assert outlook_node.get_child('Sunny').get_label() == 'Humidity'
outlook_node.add_child('Overcast', yes_node)
assert outlook_node.get_child('Overcast').get_label() == 'Yes'
outlook_node.add_child('Rain', wind_node)
assert outlook_node.get_child('Rain').get_label() == 'Wind'
assert len(outlook_node.get_children()) == 3
bin_id3.display_id3_node(outlook_node, '')
```

Running this portion of `bin_id3_uts.test_id3_ut02(self, tn=2)` produces the following output.

```
Outlook
        Rain
                Wind
                        Weak
                                Yes
                        Strong
                                No
        Overcast
                Yes
        Sunny
                Humidity
                        High
                                No
                        Normal
                                Yes
```

Before moving on, I'd like to point out in passing that our manual construction of the decision tree follows very closely what the Binary ID3 algorithm on slides 4 and 5 in `CS3430_S20_Decision_Trees_Part02.pdf` does automatically.

Reading Examples from CSV Files

In many domains where decsion trees are applied, examples come from CSV files. The archive for this homework contains the file `play_tennis.csv` that we'll use to learn the decsion tree for the `PlayTennis` concept automatically. Let's take a look at it.

```
Day,Outlook,Temperature,Humidity,Wind,PlayTennis
D1,Sunny,Hot,High,Weak,No
D2,Sunny,Hot,High,Strong,No
D3,Overcast,Hot,High,Weak,Yes
D4,Rain,Mild,High,Weak,Yes
D5,Rain,Cool,Normal,Weak,Yes
D6,Rain,Cool,Normal,Strong,No
D7,Overcast,Cool,Normal,Strong,Yes
D8,Sunny,Mild,High,Weak,No
D9,Sunny,Cool,Normal,Weak,Yes
D10,Rain,Mild,Normal,Weak,Yes
D11,Sunny,Mild,Normal,Strong,Yes
D12,Overcast,Mild,High,Strong,Yes
D13,Overcast,Hot,Normal,Weak,Yes
D14,Rain,Mild,High,Strong,No
```

We can use the method `bin_id3.parse_csv_file_into_examples(csv_fp)` in `bin_id3.py` to read the CSV file specified by `csv_fp` and convert it into example objects. Each example is a Python dictionary mapping attributes to values. Recall that attributes and their values are strings. This method also returns the column names (i.e., the attributes) specified on the first line of the file.

Let's run `bin_id3_uts.test_id3_ut00(self, tn=0)` to see how this method works. We get the list of examples and column names, get the set of attributes from the column names, make sure that we've read in 14 examples, and then print both examples and column names.

```
examples, colnames = bin_id3.parse_csv_file_into_examples('play_tennis.csv')
attribs    = set(colnames[1:])
print('attribs = {}'.format(attribs))
assert len(examples) == 14
print('Examples:\n')
for i, ex in enumerate(examples):
    print('{}) {}'.format(i+1, ex))
print('\nColumn names:')
for i, cn in enumerate(colnames):
    print('{}) {}'.format(i+1, cn))
```

Running this test produces the following output. When you run the unit test, each example prints on a separate line. I introduced new lines below to keep the margins of my LaTeX document in place.

```
attribs = {'PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind'}
Examples:
1) {'Day': 'D1', 'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'No'}
2) {'Day': 'D2', 'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Strong', 'PlayTennis': 'No'}
3) {'Day': 'D3', 'Outlook': 'Overcast', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'Yes'}
4) {'Day': 'D4', 'Outlook': 'Rain', 'Temperature': 'Mild', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'Yes'}
5) {'Day': 'D5', 'Outlook': 'Rain', 'Temperature': 'Cool', 'Humidity': 'Normal',
    'Wind': 'Weak', 'PlayTennis': 'Yes'}
6) {'Day': 'D6', 'Outlook': 'Rain', 'Temperature': 'Cool', 'Humidity': 'Normal',
    'Wind': 'Strong', 'PlayTennis': 'No'}
7) {'Day': 'D7', 'Outlook': 'Overcast', 'Temperature': 'Cool', 'Humidity': 'Normal',
    'Wind': 'Strong', 'PlayTennis': 'Yes'}
8) {'Day': 'D8', 'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'No'}
9) {'Day': 'D9', 'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'Normal',
    'Wind': 'Weak', 'PlayTennis': 'Yes'}
```

```
10) {'Day': 'D10', 'Outlook': 'Rain', 'Temperature': 'Mild', 'Humidity': 'Normal',
     'Wind': 'Weak', 'PlayTennis': 'Yes'}
11) {'Day': 'D11', 'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'Normal',
     'Wind': 'Strong', 'PlayTennis': 'Yes'}
12) {'Day': 'D12', 'Outlook': 'Overcast', 'Temperature': 'Mild', 'Humidity': 'High',
     'Wind': 'Strong', 'PlayTennis': 'Yes'}
13) {'Day': 'D13', 'Outlook': 'Overcast', 'Temperature': 'Hot', 'Humidity': 'Normal',
     'Wind': 'Weak', 'PlayTennis': 'Yes'}
14) {'Day': 'D14', 'Outlook': 'Rain', 'Temperature': 'Mild', 'Humidity': 'High',
     'Wind': 'Strong', 'PlayTennis': 'No'}

Column names:
1) Day
2) Outlook
3) Temperature
4) Humidity
5) Wind
6) PlayTennis
```

Let's run a couple more unit tests and use the tool methods that we'll need to implement the Binary ID3 algorithm. Let's start with `bin_id3_u.test_id3_ut05(self, tn=5)`.

```
examples, colnames = bin_id3.parse_csv_file_into_examples('play_tennis.csv')
outlook_sunny_examples = bin_id3.find_examples_given_attrib_val(examples, 'Outlook', 'Sunny')
print('Examples with Outlook=Sunny:\n')
for i, ex in enumerate(outlook_sunny_examples):
    print('{}) {}'.format(i+1, ex))
assert len(bin_id3.find_examples_given_attrib_val(outlook_sunny_examples, 'PlayTennis', 'Yes')) \
       == 2
assert len(bin_id3.find_examples_given_attrib_val(outlook_sunny_examples, 'PlayTennis', 'No')) \
       == 3
```

This test uses the method `bin_id3.find_examples_given_attrib_val()` to find all the examples with `Outlook=Sunny`, prints them out and then uses the same method to find the examples with `Outlook=Sunny` that also have `PlayTennis=Yes` and, following that, to find the examples with `Outlook=Sunny` that also have `PlayTennis=No`. Of the 14 examples in `play_tennis.csv`, there are 5 examples (returned by `find_examples_given_attrib_val()`) with `Outlook=Sunny`.

```
Examples with Outlook=Sunny:

1) {'Day': 'D1', 'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'No'}
2) {'Day': 'D2', 'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Strong', 'PlayTennis': 'No'}
3) {'Day': 'D8', 'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'No'}
4) {'Day': 'D9', 'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'Normal',
    'Wind': 'Weak', 'PlayTennis': 'Yes'}
5) {'Day': 'D11', 'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'Normal',
    'Wind': 'Strong', 'PlayTennis': 'Yes'}
```

The unit test `test_id3_ut20(self, tn=20)` shows how to use the method `find_most_common_attrib_val()` to find the most common (i.e., the most frequent) value of several attributes in the examples with `Outlook=Sunny`. For example, in the following three lines of this unit test we return the most common value of the attribute `Humidity`, confirm that it is equal to `'High'` and occurs in 3 examples.

```
atv, cnt = bin_id3.find_most_common_attrib_val(outlook_sunny_examples, 'Humidity', avt)
assert atv == 'High'
assert cnt == 3
```

Proportion, Entropy, Information Gain

Let's implement the three pillars of the Binary ID3 algoirhm: `proporition`, `entropy`, and `gain`.

Implement the method `proportion(examples, attrib, val)` that takes a list of examples, an attribute, and a value of that attribute (both `attrib` and `val` are strings) and returns the proporition of examples with `attrib=val`. In the entropy formula on slide 16 in `CS3430_S20_Decision_Trees_Part01.pdf`, `proportion` computes p_i. It's an estimate of the probability of an example with `attrib=val` occuring in the population of all examples. Remember that each example is a Python dictionary mapping attributes to their values. You can test your implementation of `proportion()` with `test_id3_ut03(self, tn=3)` and `test_id3_ut04(self, tn=4)`. Running these tests generates the following output.

```
============ ID3 UT 3 =======================
proportion(PlayTennis, Yes) = 0.6428571428571429
============ ID3 UT 3 passed ================
============ ID3 UT 4 =======================
proportion(Humidity, High) = 0.5
============ ID3 UT 4 passed ================
```

The next logical step after `proportion` is to implement `entropy(examples, target_attrib, avt)`. This method takes a list of examples, a target attribute (`target_attrib`) and a dictionary where each attribute is mapped to a list of all its possible values in the examples. This table is constructed by the method `construct_attrib_values_from_examples()` implemented for you in `bin_id3.py`.

An important thing to remember when computing entropy is that it's always computed with respect to the target attribute (`PlayTennis` in our case) and a given list of examples. Thus, the entropy of all examples for `PlayTennis` is different than the entropy of the examples with `Outlook=Sunny` for `PlayTennis`. The unit tests `tests test_id3_ut06a(self, tn=6)` and `test_id3_ut06b(self, tn=19)` illustrate this important difference. The unit test `test_id3_ut06a(self, tn=6)` computes the entropy of all examples with respect to `PlayTennis`. Runnning this unit test generates the following output.

```
============ ID3 UT 6a =======================
Entropy(S)=0.9402859586706309
============ ID3 UT 6a passed ================
```

Slide 14 in `CS3430_S20_Decision_Trees_Part01.pdf` shows you how this value is computed. Note that I rounded the value to 0.94 on the slide. Let's run `test_id3_ut06b(self, tn=19)` now. This unit test computes the entropy of the examples with `Outlook=Sunny` with respect to `PlayTennis`. In this context, our list of examples is as follows.

```
1) {'Day': 'D1',  'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'No'}
2) {'Day': 'D2',  'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
    'Wind': 'Strong', 'PlayTennis': 'No'}
3) {'Day': 'D8',  'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'High',
    'Wind': 'Weak', 'PlayTennis': 'No'}
4) {'Day': 'D9',  'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'Normal',
    'Wind': 'Weak', 'PlayTennis': 'Yes'}
5) {'Day': 'D11', 'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'Normal',
    'Wind': 'Strong', 'PlayTennis': 'Yes'}
```

Take a careful note that the proportions of positive and negative examples in this list are different than in the context of `test_id3_ut06a(self, tn=6)` that we just ran. Specifically, we have 3 negative (`PlayTennis=No`) examples and 2 positive (`PlayTennis=Yes`) examples. Thus, the value of entropy is different than the value computed in `test_id3_ut06a(self, tn=6)`, as corraborated by the following output.

```
============ ID3 UT 6b =======================
Entropy(S)=0.9709505944546686
============ ID3 UT 6b pass ==================
```

Once we have entropy, we can compute the information gain of an attribute. Implement the method `gain(examples, target_attrib, attrib, avt)` that computes the formula on slide 20 in `CS3430_S20_Decision_Trees_Part01.pdf`. Slides 22, 23 in the same pdf show how to compute the information gains of `Humidty` and `Wind`. The result values are rounded on the slides.

Run the unit tests `test_id3_ut07(self, tn=7)` and `test_id3_ut11(self, tn=11)` to test your implementation. Your gains should be as follows (or very, very close to the values below).

```
============ ID3 UT 7 =======================
gain(Wind)=0.04812703040826927
============ ID3 UT 7 passed ================
============ ID3 UT 11 =======================
gain(Humidity)=0.9709505944546686
============ ID3 UT 11 passed ================
```

The final piece before we can put together the Binary ID3 puzzle is the method `bin_id3.find_best_attribute(examples, target_attrib, attribs, avt))`. This method finds the best attribute (i.e., the attribute with the highest info gain) in the examples. The ties are broken arbitrarily.

Run `test_id3_ut21(self, tn=21)` to test your implementation. You should see the the following output. The information gains for all attributes are displayed with the method `bin_id3.display_info_gains(gains)`. It's a useful debugging tool.

```
============ ID3 UT 21 =======================
Information gains are as follows:
        Temperature: 0.029222565658954647
        Wind: 0.04812703040826927
        Humidity: 0.15183550136234136
        Outlook: 0.2467498197744391
Best Attribute = Outlook
Best Gain      = 0.2467498197744391
============ ID3 UT 21 passed ================
```

Binary ID3 Algorithm

Everything's in place now to implement the Binary ID3 algorithm on slides 4, 5 in `CS3430_S20_Decision_Trees_Part02.pdf`. In machine learning, applying an algorithm (e.g., Binary ID3) to data to learn a model (e.g., a decision tree) is called *fitting*. Implement the method `bin_id3.fit(examples, target_attrib, attribs, avt, dbg)`. The arguments of this method are:

1. `examples` is a list of examples, each of which is a Python dictionary;
2. `target_attrib` is a string (e.g., 'PlayTennis');
3. `attribs` is a list of attributes (strings);
4. `avt` is a dictionary constructed by `construct_attrib_values_from_examples()`;
5. `dbg` is a debug `True/False` flag.

You don't have to use the `dbg` argument. If you decide not to use it, keep it there to be compliant with the unit tests. In my implementation, when the debug flag is true, the diagnostic messages are printed out as the algorithm builds the decision tree. For example, in my implementation, I have code segments like

```
## if all examples are positive, then return the root node whose label is PLUS.
if len(SV) == len(examples):
   if dbg == True:
      print('All examples positive...')
      print('Setting label of root to {}'.format(PLUS))
      root.set_label(PLUS)
   return root
```

These messages help me see how the algorithm is working, especially when I apply it to more complex data sets. But, everybody's debugging tricks are different. So, I leave the decision to use this flag up to you.

Run `test_id3_ut22(self, tn=22)` to test your implementation. Your implementation of `fit()` should return the `id3_node` object that is the root of the decision tree shown on slide 3 of `CS3430_S20_Decision_Trees_Part02.pdf`.

Remember to remove best attributes from the list of attributes (i.e., `attribs`) in your recursive calls. Don't use a single global list of attributes. Each recursive call should have its own copy of attributes. You can use the method `copy.copy` from the `copy` package to make shallow copies of `attribs`. Here's a quick example of how I remove the best attribute from the list of attributes in my implementation of `fit` before making a recursive call.

```
if dbg == True:
   print('Removing {} from attributes...'.format(best_attrib))
copy_attribs = copy.copy(attribs)
copy_attribs.remove(best_attrib)
if dbg == True:
   print('\nComputing decision tree for examples where {}={}...'.format(best_attrib, bav))
child_node = bin_id3.fit(new_examples, target_attrib, copy_attribs, avt, dbg)
```

Here's the output I get when I run `test_id3_ut22(self, tn=22)` on my implementation with the debug flag set to `True`.

```
============ ID3 UT 22 ======================
Looking for best attribute among Temperature, Wind, Humidity, Outlook...
Information gains are as follows:
        Temperature: 0.029222565658954647
        Wind: 0.04812703040826927
        Humidity: 0.15183550136234136
        Outlook: 0.2467498197744391
Best attrib is Outlook with a gain of 0.2467498197744391...
Looking for examples where Outlook=Rain...
Found some examples...
Removing Outlook from attributes...

Computing decision tree for examples where Outlook=Rain...
Looking for best attribute among Wind, Temperature, Humidity...
Information gains are as follows:
        Wind: 0.9709505944546686
        Temperature: 0.01997309402197489
        Humidity: 0.01997309402197489
Best attrib is Wind with a gain of 0.9709505944546686...
Looking for examples where Wind=Weak...
Found some examples...
Removing Wind from attributes...

Computing decision tree for examples where Wind=Weak...
All examples positive...
Setting label of root to Yes
Adding child node Yes to root Wind on link Weak...
Looking for examples where Wind=Strong...
Found some examples...
Removing Wind from attributes...

Computing decision tree for examples where Wind=Strong...
All examples negative
Setting label of root to No
Adding child node No to root Wind on link Strong...
Adding child node Wind to root Outlook on link Rain...
Looking for examples where Outlook=Overcast...
Found some examples...
Removing Outlook from attributes...

Computing decision tree for examples where Outlook=Overcast...
All examples positive...
Setting label of root to Yes
Adding child node Yes to root Outlook on link Overcast...
Looking for examples where Outlook=Sunny...
Found some examples...
Removing Outlook from attributes...

Computing decision tree for examples where Outlook=Sunny...
Looking for best attribute among Wind, Temperature, Humidity...
Information gains are as follows:
        Wind: 0.01997309402197489
        Temperature: 0.5709505944546686
        Humidity: 0.9709505944546686
Best attrib is Humidity with a gain of 0.9709505944546686...
Looking for examples where Humidity=High...
Found some examples...
Removing Humidity from attributes...

Computing decision tree for examples where Humidity=High...
```

```
All examples negative
Setting label of root to No
Adding child node No to root Humidity on link High...
Looking for examples where Humidity=Normal...
Found some examples...
Removing Humidity from attributes...

Computing decision tree for examples where Humidity=Normal...
All examples positive...
Setting label of root to Yes
Adding child node Yes to root Humidity on link Normal...
Adding child node Humidity to root Outlook on link Sunny...

Outlook
        Rain
                Wind
                        Weak
                                Yes
                        Strong
                                No
        Overcast
                Yes
        Sunny
                Humidity
                        High
                                No
                        Normal
                                Yes
============ ID3 UT 22 passed ================
```

Prediction

This is the final cut! In machine learning, applying a learned model to classify an example is referred to as *predicting*. What's being predicted? A given example's class. In the `PlayTennis` dataset, we're given a day and we'd like to predict whether the value of `PlayTennis`, our target/concept attribute, is `True` or `False`.

Implement the method `bin_id3.predict(root, example)` that takes the root of the tree returned by `bin_id3.fit()` and an example and returns `PLUS` or `MINUS` (recall that these constants are defined at the beginning of `bin_id3.py`). This method implements the following recursive algorithm. If you get the recursion right, your implementation should be no longer than 7-8 lines of code.

```
predict(root, example)
  IF the root's label is PLUS
     THEN return PLUS
  IF the root's label is MINUS
     THEN return MINUS
  Let RAT be the root's label.
  Let RAV be the value of RAT in example.
  Let CH be the root's child connected to the root on the link RAV.
  Return predict(CH, example)
```

You can use `bin_id3.get_example_attrib_val(example, attrib)` to compute `RAV` and `id3_node.get_child(self, attrib_val)` to get `CH` connected to the root on `RAV`.

Run the unit tests `test_id3_ut23(self, tn=23)` and `test_id3_ut24(self, tn=24)` to test your implementation of `predict()`. Unit test 23 is easy. It uses the examples in `play_tennis_unlbl.csv`. These examples are just the original 14 examples in `play_tennis.csv` with their classes (i.e., the values of `PlayTennis`) removed. Essentially, we're testing the learned decision tree on the examples on which the tree was learned in the first place. Not a great testing practice, but a great unit test to make sure that everything's working.

The unit test `test_id3_ut24(self, tn=24)` runs the decision tree on 10,000 examples of PlayTennis data not involved in learning the decision tree. If your decision tree is like the tree on slide 3 of `CS3430_S20_Decision_Trees_Part02.pdf`, your accuracy will be 100%!

When you get to this point, cherish the moment for a few minutes. We've learned a tree from just 14 examples that accurately classifies 10,000 examples. What a great gift Dr. J. Ross Quinlan gave to the scientifc community by discovering the ID3 algorithm! Alas, such simple and elegant datasets and accuracy results, as some of you know, don't happen often in machine learning. In the next assignment, we'll deal with datasets for which this type of accuracy is but a dream.

What To Submit

Submit `bin_id3.py` in Canvas. Remember to work on one unit test at a time.

Stay healthy and have fun hacking this assignment!