CS 3430: Scientific Computing
Assignment 11
Entropy, Information Gain, Decision Trees, Binary ID3

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 09, 2022

1 Learning Objectives

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

Introduction

In this assignment, we'll implement the Binary ID3 algorithm that learns decision trees (DTs) for a binary target attribute from a set of examples, each of which is a set of attribute-value pairs. All attribute values are discrete. You'll save your solutions to Problem 2 in `bin_id3.py` and submit it in Canvas.

The file `bin_id3_uts.py` contains 26 unit tests I wrote for this assignment. You may want to place 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 or comment them all out initially in `bin_id_uts.py` and uncomment them one by one as you work on them.

The objective of the initial unit tests is to make you comfortable with the decision tree data structures and methods. You don't have to implement anything for these tests. Simply run them and see how it all works and fits together. The middle unit tests ask you to implement proportion, entropy, and information gain. The final bunch of unit tests tests your implementation of the Binary ID3 algorithm.

Problem 1 (0 points)

Review your notes or the slides for Lectures 22 and 23 in Canvas or your class notes to become comfortable with entropy, information gain, DTs, examples (i.e., sets of attribute-value pairs), and target attributes. You may also want to read the PDF scan of Chapter 3 from Tom Mitchell's excellent book "Machine Learning" (included in the zip). You don't have to read this chapter. It's strictly FYI. If you're interested in classical machine learning methods, I recommend Dr. Mitchell's text. I think he does an excellent job presenting various machine learning methods and algorithms and gives many references and pointers to interesting publications and projects.

Problem 2 (5 points)

As you read through the sections below, keep in mind that our end objective is to implement the Binary ID3 algorithm summarized in pseudocode on Slides 25–27 in Lecture 23.

Decision Tree Nodes

Fist, we need to get comfortable with the `id3_node` class in `bin_id3.py`. That's the data structure out of which we'll build DTs.

```
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]
```

Reading Examples from CSV Files

In many domains where DTs 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. This is Dr. Quinlan's original dataset we worked with in the last two lectures. The CSV file looks as follows.

```
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. In our implementation, attributes and their values are strings (this is the most generic solution). 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.

```
>>> from bin_id3 import bin_id3
>>> examples, colnames = bin_id3.parse_csv_file_into_examples('play_tennis.csv')
>>> print(examples[0])
{'Day': 'D1', 'Outlook': 'Sunny', 'Temperature': 'Hot',
'Humidity': 'High', 'Wind': 'Weak', 'PlayTennis': 'No'}
### This is how we can convert attribs to a set.
>>> attribs    = set(colnames[1:])
>>> print('attribs = {}'.format(attribs))
attribs={'PlayTennis', 'Wind', 'Humidity', 'Temperature', 'Outlook'}
>>> assert len(examples) == 14
>>> for i, ex in enumerate(examples):
        print('{}) {}'.format(i+1, ex))
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'}
>>> for i, cn in enumerate(colnames):
        print('{}) {}'.format(i+1, cn))
1) Day
2) Outlook
3) Temperature
4) Humidity
5) Wind
6) PlayTennis
```

Let's run `test_id3_ut01(self, tn=1)`. Here it is.

```
def test_id3_ut01(self, tn=1):
    """
    Tests bin_id3.construct_attrib_values_from_examples().
    """
    print('\n============ ID3 UT {} ======================='.format(tn))
    examples, colnames = bin_id3.parse_csv_file_into_examples('play_tennis.csv')
    avt = bin_id3.construct_attrib_values_from_examples(examples, colnames[1:])
    print('Attribute --> Values:\n')
    for k, v in avt.items():
        print('{} --> {}'.format(k, v))
    print('\n============ ID3 UT {} passed ================'.format(tn))
```

This unit test constructs a dictionary from a list of examples where each attribute is mapped to a list of all its possible values in examples. The method `construct_attrib_values_from_examples()` that does this construction is in `bin_id3.py`. When you run it you should see the following output.

```
============ ID3 UT 1 =======================
Attribute --> Values:

Outlook --> {'Rain', 'Sunny', 'Overcast'}
Temperature --> {'Mild', 'Hot', 'Cool'}
Humidity --> {'High', 'Normal'}
Wind --> {'Weak', 'Strong'}
PlayTennis --> {'No', 'Yes'}

============ ID3 UT 1 passed ================
```

Let's build the decision tree on Slide 6 in Lecture 22 manually. We'll later build it automatically with the Binary ID3 algorithm. The code is in `test_id3_ut02()`. 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) in the final decision tree 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`. Although the string values are arbitrary, you shouldn't change them for consistency's sake.

```
>>> from bin_id3 import *
>>> PLUS
'Yes'
>>> MINUS
'No'
>>> yes_node = id3_node(PLUS)
>>> assert yes_node.get_label() == PLUS
>>> no_node = id3_node(MINUS)
>>> assert no_node.get_label() == MINUS
```

Let's build the node `Humidity`. 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.

```
>>> from bin_id3 import *
>>> 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()`, 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` and 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 DT built (or *learned* if we use 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 an inconsequential implementational detail that we'll ignore in this assignment and assume that attributes and their values are strings. The recursive structure of DTs will allow us to use these links to classify 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()` produces the following output.

```
Wind
        Strong
                No
        Weak
                Yes
```

We finish building the decision tree 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()` produces the following output. And, we're done with the manual construction of the tree.

```
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 closely the operation of the Binary ID3 algorithm we developed in Lecture 23.

Let's run `test_id3_ut03(self, tn=3)` that shows you how to use the method `find_examples_given_attrib_val()` of the `bin_id3` class to find all examples where a given attribute has a given value. Specifically, this unit test shows you how to find all examples where `Outlook=Sunny`. 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 that have `Outlook=Sunny` and also have `PlayTennis=Yes` and, following that, to find the examples that `Outlook=Sunny` and have `PlayTennis=No`. Of the 14 examples in `play_tennis.csv`, there are 5 examples with `Outlook=Sunny` (returned by `find_examples_given_attrib_val()`).

```
============ ID3 UT 3 ======================
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'}

============ ID3 UT 3 passed ================
```

The class `bin_id3` has the method `find_most_common_attrib_val()` we can use to find the most common (i.e., the most frequent) value of a given attribute in a given set of examples. 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.

```
>>> from bin_id3 import *
>>> examples, colnames = bin_id3.parse_csv_file_into_examples('play_tennis.csv')
>>> colnames
['Day', 'Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis']
### Let's get all examples where Outlook=Sunny.
>>> outlook_sunny_examples = bin_id3.find_examples_given_attrib_val(examples,
                                                            'Outlook', 'Sunny')
### Let's construct the table (avt) that maps each attribute to
### its values; we use colnames[1:], because we don't need 'Day'.
>>> avt = bin_id3.construct_attrib_values_from_examples(examples, colnames[1:])
>>> avt
{'Outlook': {'Overcast', 'Sunny', 'Rain'},
'Temperature': {'Cool', 'Hot', 'Mild'},
'Humidity': {'High', 'Normal'},
'Wind': {'Strong', 'Weak'},
'PlayTennis': {'Yes', 'No'}}
### Let's find the most frequent attribute value of Humidity and its count
### in avt (attribute value table) computed in the previous step.
>>> atv, cnt = bin_id3.find_most_common_attrib_val(outlook_sunny_examples,
                                                    'Humidity', avt)
>>> assert atv == 'High'
>>> assert cnt == 3
```

Proportion, Entropy, Information Gain

We now have the tools and data structures to 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 formulas on Slides 17, 18 in Lecture 22, `proportion` computes p_i. It's an estimate of the probability of an example with `attrib=val` occuring in the population of all examples. You can test your implementation of `proportion()` with `test_id3_ut04()` and `test_id3_ut05()`. Running these tests generates the following output.

```
============ ID3 UT 4 ======================
proportion(PlayTennis, Yes) = 0.6428571428571429
============ ID3 UT 4 passed ================
============ ID3 UT 5 ======================
proportion(Humidity, High) = 0.5
============ ID3 UT 5 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 its possible values found in the examples. This table is constructed by `construct_attrib_values_from_examples()` in `bin_id3.py`.

An important thing to remember when computing entropy is that it's computed with respect to a given target attribute (i.e., `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` with respect to `PlayTennis`. The unit tests `tests test_id3_ut06()` and `test_id3_ut07()` illustrate this difference. Here's my output.

```
============ ID3 UT 6 ======================
Entropy(S)=0.9402859586706309
============ ID3 UT 6 passed ================
============ ID3 UT 7 ======================
Entropy(S)=0.9709505944546686
============ ID3 UT 7 pass ==================
```

Once we have entropy, we can compute the information gain of an attribute. Toward that end, implement the method `gain(examples, target_attrib, attrib, avt)` that computes the formula on Slide 19 in Lecture 22. Run the unit tests `test_id3_ut08()` and `test_id3_ut09()`, and `test_id3_ut10()` to test your implementation. Your gains should be as follows (or sufficiently close to the values below).

```
============ ID3 UT 8 ======================
gain(Wind)=0.04812703040826927
============ ID3 UT 8 passed ================
```

```
============ ID3 UT 9 =======================
gain(Humidity)=0.15183550136234136
============ ID3 UT 9 passed ================
============ ID3 UT 10 =======================
gain(Outlook)=0.2467498197744391
============ ID3 UT 10 passed ================
```

Another important method we need before we implement the Binary ID3 algorithm is the `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_ut22()` 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 22 =======================
Information gains are as follows:
Outlook: 0.2467498197744391
Humidity: 0.15183550136234136
Wind: 0.04812703040826927
Temperature: 0.029222565658954647
Best Attribute = Outlook
Best Gain      = 0.2467498197744391

============ ID3 UT 22 passed ================
```

Binary ID3 Algorithm

Everything's in place now for us to implement the Binary ID3 algorithm. The algorithm is summarized on Slides 25, 26 in Lecture 23. 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 don't to use it, please 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 and large data sets with missing attributes or attribute values. But, everybody's debugging tricks and preferences are different. So, I leave the decision to use this flag up to you.

Run `test_id3_ut23()` to test your implementation of the algorithm. The multi-line comment right before this unit tests gives my output. A call to `fit()` returns the `id3_node` object that is the root of the decision tree. My output to this test is in `test_id3_ut23_output.txt`. You output doesn't have to be exactly the same but the final DT should look be the one printed at the end of the test.

Remember to remove best attributes from the list of attributes (i.e., `attribs`) in your recursive calls. I recommend against using 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 you can remove the best attribute from the list of attributes in `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)
child_node = bin_id3.fit(new_examples, target_attrib, copy_attribs, avt, dbg)
```

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 `Yes` or `No` (i.e., `PLUS` or `MINUS`).

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 (10 maximum if you use local variables for clarity).

```
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_ut24()` and `test_id3_ut25()` to test your implementation of `predict()`. Unit test 24 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, of course, but a great unit test to make sure that everything's working.

The unit test `test_id3_ut25()` runs the fit decision tree on 10,000 examples of PlayTennis data not involved in learning the decision tree. If your decision passes unit test 24, then its accuracy for unit test 25 should be 100%! This is an easy domain.

When you get to this point, cherish the moment for a few minutes. We've learned a tree from just 14 examples and used it to accurately classify 10,000 examples. What a great gift Dr. Quinlan gave to the scientifc community by discovering the ID3 algorithm! Of course, such simple and elegant datasets and accuracy results, as some of you may already know, don't happen too often in machine learning. Missing attributes or attribute values due to data entry errors or sensor failures and inseparable categories are the norm in real world datasets, which push the classification accuracy down.

What To Submit

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

Have fun hacking this assignment!