CS 3430: S24: Scientific Computing
Assignment 11
Entropy, Information Gain, Decision Trees, Binary ID3

Vladimir Kulyukin
Department of Computer Science
Utah State University

April 13, 2024

1 Learning Objectives

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

Introduction

In this assignment, we will implement the Binary ID3 algorithm that learns decision trees for a binary target attribute from a set of examples, each of which is a set of attribute-value pairs. All values are discrete, i.e., they can be labeled with symbols or with natural numbers. You will save your solutions to Problems 1 and 2 in `bin_id3.py` and submit it in Canvas.

The file `bin_id3_uts.py` contains a couple dozen unit tests I wrote for this assignment. The objective of the initial unit tests, i.e., unit tests $0 - 3$, is to make you comfortable with the decision tree data structures and methods. You do not have to implement anything for these tests. Simply run them while going through the Coding Lab section below and see how it all fits together. The other unit tests are for Problems 1 and 2.

Optional Problem 0 (0 points)

Review your notes/slides for Lectures 24 and 25 in Canvas to become comfortable with the concepts of entropy, information gain, decision tree, attribute, value, example/sample (i.e., a set of attribute-value pairs), and target attribute. As I write this assignment on April 13, 2024, the Lecture 24 PDF is in Canvas. I plan to post the Lecture 25 PDF on 04/15/2024 after we are done with the F2F lecture. You can also read the PDF scan of Chapter 3 from Tom Mitchell's book "Machine Learning" (included in the zip for this assignment). I based a few slides in Lecture 24 on this chapter. If you are interested in standard machine learning methods, I recommend Dr. Mitchell's text. In my opinion, he does an excellent job presenting various standard machine learning methods and algorithms and gives many references and pointers to interesting publications and projects. You do not have to read this PDF to complete the assignment. It is stricly FYI.

Coding Lab (0 points)

Decision Tree Nodes

We need to get comfortable with the `id3_node` class in `bin_id3.py`, i.e., the data structure out of which we will build decision trees.

```
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

As we discussed in Lecture 24, in many domains of scientific computing where we need to classify the data, decsion trees are applied to datasets of examples saved in CSV files. The zip for this homework contains the file `play_tennis.csv` that we will use to learn the decsion tree for the `PlayTennis` concept automatically. This is Dr. Quinlan's original dataset we worked with in Lecture 24 (cf. Slide 14). 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 the file path `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 method also returns the column names (i.e., the attributes) specified on the first line of the CSV file, which is the case with many datasets from which decistion trees are built.

Let us run `bin_id3_uts.test_id3_ut00(self, tn=0)` to see how this method works. We will 1) get the list of examples and column names, 2) get the set of attributes, i.e., the column names, 3) will make sure that we have read in all 14 examples, and 4) will print the attributs and examples.

```
## 1)
examples, colnames = bin_id3.parse_csv_file_into_examples('play_tennis.csv')
## 2)
attribs    = set(colnames[1:])
## 3)
assert len(examples) == 14
## 4)
print('attribs = {}'.format(attribs))
print('\nColumn/attribute names:')
for i, cn in enumerate(colnames):
    print('{}) {}'.format(i+1, cn))
print('Examples:\n')
for i, ex in enumerate(examples):
    print('{}) {}'.format(i+1, ex))
```

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 reasonable margins of my LaTeXdocument.

```
============ UT 0 ======================
attribs = {'PlayTennis', 'Humidity', 'Wind', 'Outlook', 'Temperature'}

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

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'}
```

Make sure that you are comfortable with the output above. Each example prints as a set of attribute-value pairs. On to `test_id3_ut01(self, tn=1)`. This unit test constructs a string-to-set-of-strings dictionary from a list of examples. The dictionary maps each attribute 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 we run it you should see the following output.

```
Attribute --> Values:

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

We can now build the decision tree on Slide 32 in Lecture 24 manually. We will later build it automatically with BinID3. This is the same tree we built step by step in Lecture 24 by computing the information gain of each attribute, choosing the one with the largest gain, removing it from the list of attributes, and recursing on the remaining attributes. The code is in `test_id3_ut02()`. All the classes and methods for this unit test are implemented in `bin_id3.py`. I wrote this test to show you how you can set labels of individual notes and link child nodes to parent nodes on specific attribute values.

The label of each node, except for the leaf nodes, is an attribute. Recall BinID3 learns decision trees for binary concepts, e.g., `PlayTennis` (aka, target attributes) that have two possible values – `Yes` and `No`. In other words, each example can fall into one of the two possible classes with respect to the target attribute `PlayTennis` – `Yes` and `No`. Thus, the leaf

nodes, i.e., the classes, in the final decision tree have the labels `'Yes'` or `'No'`. So, we start from the leaves nodes and build up. The constants `PLUS` ('Yes') and `MINUS` ('No') are defined at the beginning of `bin_id3.py`. Although the string values are arbitrary (e.g., we can set them to 0 or 1 instead), you should not change them change them for consistency's sake.

```
plus_node = id3_node(PLUS)
assert plus_node.get_label() == PLUS
minus_node = id3_node(MINUS)
assert minus_node.get_label() == MINUS
```

Now we build the node `Humidity`. We create the node with the label `'Humidity'` and then connect/link two children to it on the two possible 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', minus_node)
humidity_node.add_child('Normal', plus_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 will see the following output.

```
Humidity
        High
                No
        Normal
                Yes
```

Let me reiterate this point again to drive it home – the internal nodes of any decision tree built (or *learned* if we use the standard machine learning terminology) with ID3 are attributes connected to their children via their values. Now we move on to constructing the attribute node `Wind`.

```
wind_node = id3_node('Wind')
assert wind_node.get_label() == 'Wind'
wind_node.add_child('Strong', minus_node)
wind_node.add_child('Weak', plus_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 value links: `'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', plus_node)
assert outlook_node.get_child('Overcast').get_label() == PLUS
outlook_node.add_child('Rain', wind_node)
assert outlook_node.get_child('Rain').get_label() == 'Wind'
assert len(outlook_node.get_children()) == 3
```

At the end of `bin_id3_uts.test_id3_ut02()` the call `bin_id3.display_id3_node(outlook_node, '')` recursively displays the decision tree as follows.

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

Thus, the node `Outlook` is connected to the node `Wind` by the link `Rain`, to the leaf node `Yes` by the link `Overcast`, and to the node `Humidity` by the link `Sunny`, and so on.

The test `test_id3_ut03()` shows you how to use the method `find_examples_given_attrib_val()` of `bin_id3` to find all examples, in each of which a given attribute has a given value. Specifically, this unit test finds all examples with `Outlook=Sunny` and prints them out. Then, in the examples with `Outlook=Sunny`, the method finds all the examples with with `PlayTennis=Yes` and, following that, the examples with `PlayTennis=No`. Here is my output.

```
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'}

Examples with Outlook=Sunny and PlayTennis=Yes:
1) {'Day': 'D9', 'Outlook': 'Sunny', 'Temperature': 'Cool',
    'Humidity': 'Normal', 'Wind': 'Weak', 'PlayTennis': 'Yes'}
2) {'Day': 'D11', 'Outlook': 'Sunny', 'Temperature': 'Mild',
    'Humidity': 'Normal', 'Wind': 'Strong', 'PlayTennis': 'Yes'}

Examples with Outlook=Sunny and PlayTennis=No:
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'}
```

Problem 1 (2 points): Proportion, Entropy, Information Gain

We have the tools and data structures in place 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 formula on Slide 16 in Lecture 24, `proportion` computes p_i. It is an estimate of the probability of an example with `attrib=val` occuring in a given dataset of examples.

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 is 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`.

Once we can compute 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 18 in Lecture 24.

You can test your implementations of `proportion`, `entropy`, and `gain` with the unit tests 04 – 22. You values should be close to these.

```
========== UT 22 ====================
Information gains are as follows:
Humidity: 0.15183550136234136
Wind: 0.04812703040826927
Outlook: 0.2467498197744391
Temperature: 0.029222565658954647
Best Attribute = Outlook
Best Gain      = 0.2467498197744391
```

Problem 2 (3 points): BinID3

Fitting

Everything is in place now to implement `BinID3` (cf. Slides 33–36, Lecture 24). 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.

A call to `fit()` returns the `id3_node` object that is the root of a decision tree fit on a given dataset of examples. You do not have to use the `dbg` argument. If you do not 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 decision tree construction is evolving. But, everybody's debugging tricks and preferences are different and IDE-specific. So, I leave the decision to use this flag up to you.

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

In machine learning, applying a learned model to classify an example is referred to as *predicting*. What's being predicted in our case? A given example's class. In the `PlayTennis` dataset, we are given a day and we would 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` to compute `RAV` and `id3_node.get_child` to get `CH` connected to the root on `RAV`.

Run unit tests 23, 24, 25 to test your implementation of `BinID3`. The multi-line 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%.

When you get to this point, cherish the moment for a few minutes. We have learned a tree from just 14 examples and used it to accurately classify 10,000 examples. What a great gift Dr. J. Ross Quinlan gave to the scientifc community by discovering ID3! Alas, such simple and elegant datasets and accuracy results, as some of you may already know, do not happen too often. Missing attributes or attribute values due to data entry errors or sensor failures are the norm in many real world datasets, which push the classification accuracy down. As the saying goes, you are as good as your data.

What To Submit

Submit `bin_id3.py` in Canvas.

Happy Hacking!