Anita Slater
CSI 431 -- Data Mining
HW2

1a. Computing Information Gain relies on computing the entropy for the original data set, the dataset of the D_y and the entropy of D_n. Because entropy seemed to be a function that would be used several times, I decided to make it it's own method to modularize my code. That way I could compute information gain as :

```
(entropy(D)-((len(yy)/len(y)*entropy(Dy))
           +(len(yn)/len(y)*entropy(Dn))))
```

Where *yy* and *yn* are the lists of all of the classes of the entries in D_y and D_n,respectively. And *len(y)* is the number of classes in the original data set. This makes the code not only less complex, but it also looks similar to the given formula:

$$IG = H(D) - H(D_Y, D_N) = H(D) - [\frac{n_Y}{n} H(D_Y) + \frac{n_N}{n} H(D_N)]$$

Along with entropy, I also included 2 other helper functions: *split()* and *prob()*. *split()* takes a data set, an index, and a value and returns the D_y and D_N as a result of the split. For a point to be classified into D_y, the attribute value must be less than or equal to the parameter value. If this condition is not met, the instance gets classified into D_N.

prob() takes a list of classes, *y*, and returns a tuple of the number of instances of each class. As I was writing this code, I found myself writing this for loop for every classifier, so I just threw it into its own function to be more efficient.

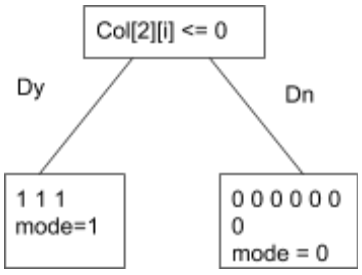
1b. Similar to what I did in 1a, I sought to break down the formula into little pieces and then solve it step by step. However, when I did this the first time, I actually became lost in all my variables, and I wasn't sure what did what anymore. So I renamed some things and added a lot of comments. Then I just broke down the formula and arrived at an answer.

1c. The CART classifier was the one that I had the most fun with. Even though it was seemingly the most daunting formula, it was, again, a relatively easy one to implement, even though I had to include this block comment in my code to remind me of all of my variables:

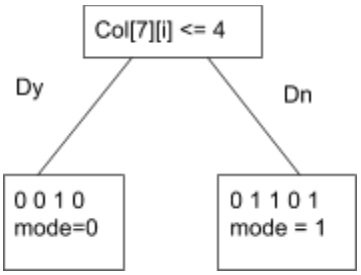
```
#yc1: The number of c1(1)s in yes tree
#yc2: The number of c2(0)s in yes tree
#nc1: The number of c1(1)s in no  tree
#nc2: The number of c2(0)s in no  tree
#yy: contains the classes for each entry in yes tree
#yn: contains the classes for each entry in no  tree
```

1d-e. Implementing this made it very easy to see how different data affect each of these measurements. Before the final product, I had each classifier be printed to the console. Instead of splitting into each criteria and having 1 of 3 for loops execute, I decided to save ~100 lines of code to have the loop evaluate the criteria each iteration, which is less efficient computation-wise. Something that also resulted in ugle code, is the fact that the *best* value had to be changed if the GINI classifier was being used, since the best GINI value is one that's been minimized. Also because it's been minimized, instead of taking the largest value, the smallest value is the one that will be passed along, which was another condition to be solved inside the for loop.

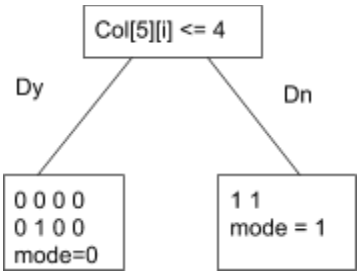
1f. **Decision Tree for CART:** index=2, value=0



Decision Tree for IG: index=7, value=4



Decision Tree for GINI: index=5, value=4



1g. Initially, I was really confused about this problem. I wrongfully assumed that to predict the value, all you had to do was classify the testing data based on the classifier, and then once you got the split value, anything in D_y would be in class 1 and anything in D_N would be in class 0. This made sense with the CART classification, but the other 2 seemed backwards. Then I read a little further and saw how prediction was actually done.

1. Train the data
    a. Determine the classifier
    b. Determine the best split
2. Classify the Test data
3. With the new split, find the mode of each subtree
4. Rexamine test data, with each split, assign a class based on the subtree the data ends up in

In order to easily visualize the different predictions of each classifier,I have the main method print out the following:

```
(0.53, 2, 0)[0, 0, 0, 0, 1, 1, 0, 1, 0, 0] [CART]
(0.31, 7, 4)[0, 0, 0, 1, 1, 1, 1, 1, 0, 0] [IG]
(0.25, 5, 4)[0, 0, 0, 0, 1, 1, 0, 0, 0, 0] [GINI]
            [0, 0, 0, 0, 1, 1, 0, 1, 0, 0] [ACTUAL]
```

The portion in the parenthesis is the output of *bestSplit()* when given the dataset *D* and the criteria which is all the way to the right. The portion in the brackets is the result of each *classify* function. The final row is just the output of the actual classes of the testing set, for comparison.

The CART classifier actually made the fewest mistakes and was a perfect classifier of the testing data. The IG classifier made 3 mistakes and was the worst classifier. Finally, the GINI classifier only one mistake when predicting the classes of the testing data.

Something interesting that I noticed accidently, is that if each classifier is trained with the testing set, they will all say that the best split is obtained via index 2 and value 0.