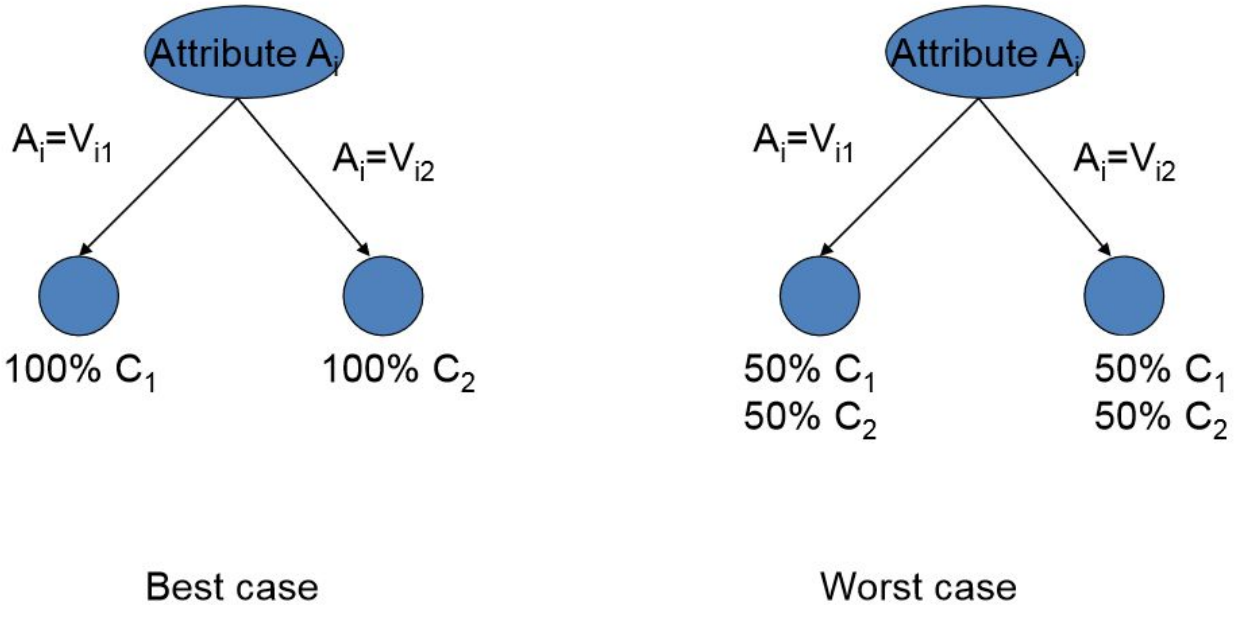
Hunter Esler
Dr. Phillips
CSCI 4350
14 November 2019

ID3 Tree Project

Decision tree classification is a method of classification that uses attributes associated with nodes in a tree and splits based on the values given, eventually getting to a terminal node to classify an object. Decision tree learning is the method of coming up with such trees in an efficient or logical manner. To first understand a tree, we must first know what a node is. A node is a "root" of a branch that evaluates a decision based on the object we are trying to classify. Once it decides what direction to go, it goes down the tree and gives the object to the next node that makes the next decision. Depending on how many splits there are, this structure eventually can look like a "Tree". The way these nodes make decisions is with attributes. Each object will have associated attributes, such as length of hair, number of freckles (for a human), and the node then splits these into decisions to bring down the tree. A terminal node is the end of a branch of the tree that has decided it knows what the object is and classifies it.

The ID3 Method is the method that we use to determine how to split the tree in order to make decisions. Essentially what ID3 does is it goes through each potential split point for each and every attribute, and calculates the maximum information gain for splitting at that spot. The more information gain we can achieve, the faster we will be able to classify an object resulting in smaller trees.



Best case Worst case

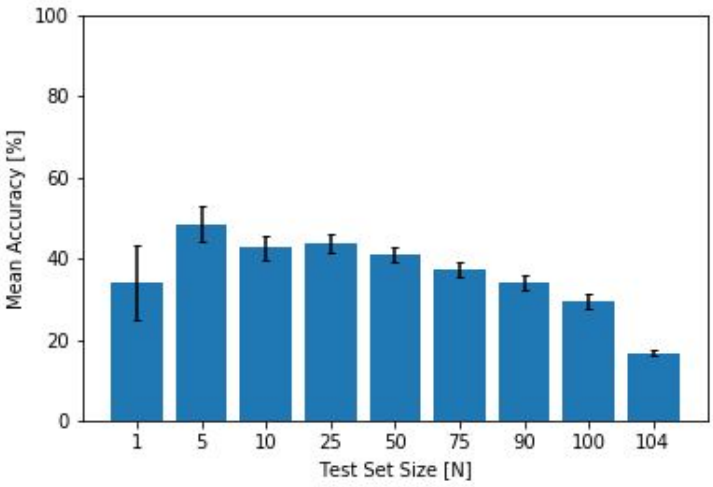
With this figure, we can see that if we split a certain way, like the left side, we have really good information gain as we generate two terminal nodes. However, if we split poorly we can

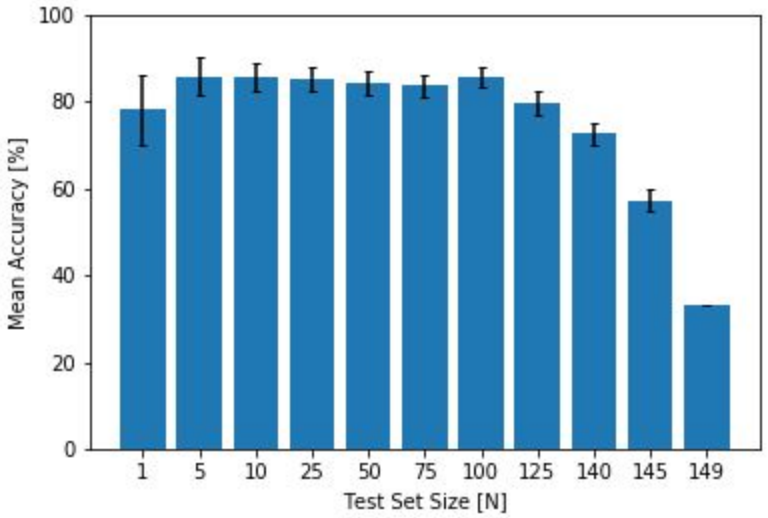
end up like the right side which will have two different possibilities of classification which will result in more tests.

To calculate the information gain, we must first determine how much information we have. This is calculated by summing the probabilities (P) as -P(n)*log_2(P(n)) for each attribute n. Then, to find the information gain, we then get the entropy which is the sum of the information of the splits multiplied by the probability of that split. (50% and 50% as above figure). This entropy, E, is then subtracted from I as I-E to get the gain. This process is repeated for each and every potential split point, until we find the maximum gain and decide to split there. Then we repeat the process until all terminal nodes are reached.

To translate this algorithm into code, I created 3 separate functions for calculating the probability of an attribute, the information of a set, and calculating the gain. This program also is only able to determine split points depending on whether the attribute is less than a split point or greater than or equal to a split point. This means that as I iterate through each attribute trying to find the maximum gain, I determine split points based on a sorted version of the list for each attribute. Each split point is the average of a change in values. To create an easy sorted list I used numpy's argsort which is an indexed sort which allows us to have N sorted arrays in one data structure instead of having N data structures. For the tree structure I have a class called node which contains the attribute being split, the split point, whether it is a terminal node, and the classification if it is a terminal node. The node also contains pointers to the less than and greater than nodes if it isn't a terminal node to further test. The program takes two arguments, a file for the training data to create the decision tree, and another file for testing the decision tree created.

To test my id3 program, I had two different data sets, one for iris data, another for cancer data. Each one was tested by randomly splitting the data into different training/test sets. To evaluate the difference in training data size, each test started with few test samples and more training samples one hundred times, and eventually increased to many test samples but fewer training samples. This was done to determine what balance would be needed for iris and cancer data. The following charts show the % accuracy depending on the amount of test and training data used for each type of classification. The left is Cancer Data, the right is Iris data.





For Iris data, our tree generated was decently accurate. From test sizes 5 through 100 the results were about 80-90% accurate! One problem I see with this data is the test sizes of 125 and above. Since there were only three different types of classes, the tree has a decently high chance of getting the class right anyway. This means that the results on that side can't be too reliable. You can see this with test size 149. Since we trained with only one class label, it was only able to get one third of the classes correctly with zero standard deviation. That proves that it picked the same class regardless of what the input was. The one test size has higher deviation because it only has one chance to get it right and won't average out like the ones with higher test data.

Cancer data didn't do as well as I would have hoped. This might be because the data is more complicated and thus harder to predict. With only 50% accuracy with a test set of 5 it's not a very reliable test to see if you have cancer. Though, with cancer 50% likely-hood of correctly diagnosing is better than nothing considering the effects of being wrong.

With both of these results in mind, it might be wise to increase the training data a lot in order to get much more accurate results. With only 105 training samples and some of them being used to test, it isn't too much of a wonder why our accuracy it lacking. In the end though, it was fun trying to figure out how to code this project and get some meaningful data with my very lacking knowledge in python.