Background

In the field of machine learning, more specifically supervised learning, there exists a need to classify data using a data structure known as a decision tree. Decision trees can take on many purposes, but often they are used to make educated guesses about data. Different values of data determine which way, if any, that the tree needs to be traversed. Terminal nodes represent decisions while the others "ask questions" that lead to more informed decisions. As one could expect, this is a very useful data structure if it is properly created with enough good quality data, as informed predictions can be made on entirely new data. However, a key question that needs to be answered about these are how they are to be constructed. The goal of this project was using what is known as the ID3 algorithm to create a decision tree.

Iterative Dichotomiser 3 (ID3) Algorithm

The iterative dichotomiser version three algorithm (ID3) is a method for building a decision tree that focuses on calculating the information gain of splitting a dataset relative to every attribute and then partitioning the data such that the split that yielded the highest information gain is selected-yielding subsets that have less uncertainty. This is done on the subsets until all elements in the subset belong to the same categorization or there are no more potential split points that can be selected. In the end, a traversable decision tree is created where the terminal nodes reached by navigating according to values of attributes yield categorizations. An example of a possible decision tree created by the id3 algorithm can be found in figure one. Notice how each non-terminal node makes an inquiry about the data and then performs a traversal to the node that matches the results of the inquiry. For example, if the Patrons attribute is equal to Full, a move will be made in the tree using the edge that matches the value and then the WaitEstimate attribute will then be questioned. This process of questioning and traversing will continue until a terminal node is reached.

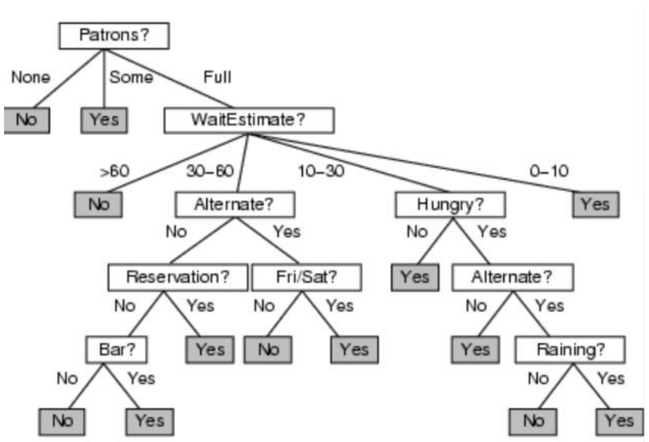

Figure 1 Decision tree created by ID3 algorithm (16, Supervised Learning, Phillips)

Approach

The task in open lab three was to implement the id3 algorithm in python. id3.py takes in two arguments: a filename representing the training data, and a filename representing the validation data. It puts both datasets into numpy arrays and then performs the entropy calculations required to determine the best splits. The formula used to calculate the information gain is demonstrated in figure two:

$$\sum_{k=1}^{K} P(C_k)(-\log_2 P(C_k)) + \sum_{j=1}^{J} P(A_i = V_{ij}) \sum_{k=1}^{k} P(C_K|A = V_{ij}) (\log_2 P(C_K|A = V_{ij}))$$

Figure 2 Information gain equation (32 Supervised Learning, Phillips)

Open lab three used a binary split approach. All the data is sorted by attribute and then potential split points are found: any time the attribute increases, a value directly between the two values is considered and then the resulting entropies are calculated to determine any information gain. As a result of this binary split approach, all the split nodes end up just "asking" if a particular attribute is less than or greater than a given split value. All non-terminal nodes only have a left and a right pointer meaning that the decision making is quite simple.

After the decision tree is created, the validation data from earlier is checked against the tree and the total number of correct categorizations is printed to standard out. For this OLA, a Monte Carlo cross-validation method was utilized for creating the validation data. The provided split.bash script randomly shuffles the data in a given file and then pulls n validation sets from it. Cross validation allows for easily checking the performance of the decision tree as the validation data will be randomly sampled every single time the program is ran, meaning that with a relatively small sample size one can get a good idea of how robustly the tree performs.

Results

To test the program, one hundred different training and validation sets of size m-v and v, respectively, were tested where m=the total number of examples in the entire data set and v is the desired number of validation examples. There were two data sets that were used: the iris data set, and the cancer data set. The first is a dataset describing various species of irises, and the second is a dataset describing various types of growths on the breast-some of which are cancerous. The v sizes used in both data sets are shown in figure three.

Iris's v=[1, 5, 10, 25, 50, 75, 100, 125, 140, 145, 149]

Cancer's v=[1, 5, 10, 25, 50, 75, 90, 100, 104

Figure 3 V values for iris and cancer datasets, (top to bottom)

The results of the one hundred different runs of each case were placed into a files, and then a short jupyter notebook was used to parse the data and then plot the mean and standard error of the mean of the percentage of testing examples corrected classified by the decision trees. The two plots generated are in figure 4 and figure 5.

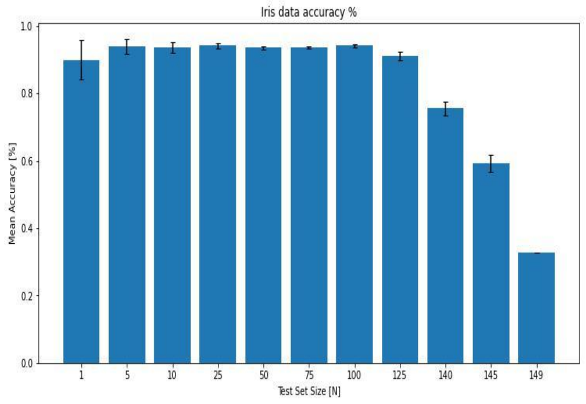

Figure 4 Iris data accuracy, error shown with lines

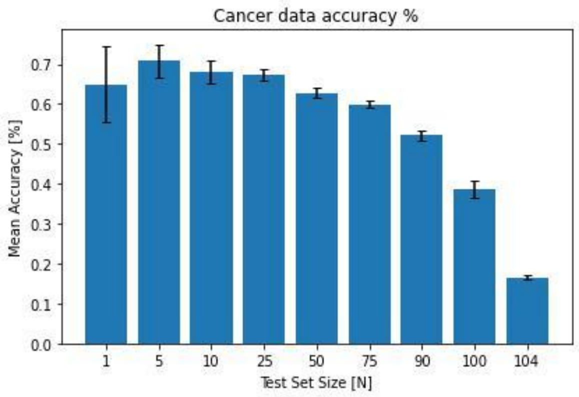

Figure 5 Cancer data accuracy, error shown with lines

The arguably simpler dataset was the iris data set as it had three categorizations and each categorization had equal quantities. The tree had a substantial amount of accuracy as eight of the eleven cases had greater than eighty percent accuracy. The lowest accuracy was on the largest test size which makes sense as having less data to train with is going to inherently cause it to not be accurate. There was also the highest degree of uncertainty for the case with only one validation data point which also makes sense because if one is only testing one data point, it isn't really a good indication of accuracy. This uncertainty levels out as the dataset reaches the test sizes that are close in size to the training sizes, but then starts to increase again. The large test size had little to no uncertainty as the runs of that case put out identical answers every time. It's unclear if that is an implementation error or simply a result of underfitting.

Upon glancing at the cancer data, one can see that there was a substantial drop off in accuracy as the test set size increased which as stated before makes sense, but what is worth noting is there was a *very* high degree of uncertainty for the lower test sizes and a very low accuracy for the higher test sizes. The cancer data was more complex than the iris data and had more categorizations. This more than likely made the tree less accurate because the splits were not able to reduce entropy as much because of the increase in number of categories.

Both results seem to mostly make logical sense, although the uncertainty being near zero on both cases' highest test set sizes is a little puzzling. Although both used a shuffle command that will yield different permutations every time it could not have just been a fluke as the chances of that happening for 100 runs on two different data sets are astronomical.

Conclusion/Considerations

When implementing the id3 algorithm, there was a two-to-three-day period where the results were incorrect from the expected. After a large amount of time debugging, it was discovered that the original test function written was logically incorrect and yielding bad traversals. This severely impacted the development flow of the open lab, but once this was fixed the output was what was expected. It would be very interesting to run this program against a very large dataset to see how versatile it is. There aren't any hacks that would reduce its robustness, and it uses a significant amount of numpy methods when possible so it won't be too slow.

One slight drawback to the project is that it uses Monte Carlo subsampling which means that each time it is ran, the training and validation sets will be different. To counter this, the specific case was run one hundred times. This may cause some issues as with certain cases, the same configuration could on paper appear twice and skew results. It also made debugging difficult at first. One way to get around this is to manually pass the training data in as the exact same as the validation data i.e. running the command *./id3.py datafilename datafilename*. This was very useful while initially debugging because the printed result of this should equal the length of the training data itself.

Overall, the results were pretty close to what was expected, and the project was a good lesson on tree traversal in python as the lesson learned from doing it wrong will not be forgotten. Some future work could be to implement a random forest program or to do more research into the C4.5 algorithm as it was Quinlan's extension his original id3 algorithm (Russell).

References

Phillips, Joshua Dr. (2022 October 26). Open Lab 3: Supervised Learning. Department of Computer Science. Middle Tennessee State University. https://jupyterhub.cs.mtsu.edu/azuread/services/csci4350-materials/private/OLA3-4350.pdf

Phillips, Joshua Dr. (2022, October 24). Supervised Learning. [32]. Department of Computer Science. Middle Tennessee State University. https://jupyterhub.cs.mtsu.edu/azuread/services/csci4350-materials/private/2022-10-24_Chapter_18a.pdf

Russell, Stuart J. (Stuart Jonathan). (2020). Artificial intelligence : a modern approach. Upper Saddle River, N.J. :Prentice Hall, Chapter 19 Bibliographical and Historical Notes.