Viet Duong
CSC 242
April 27th, 2017

Project 04 Write-Up
Learning

A. Program Design

My design consists of 3 packages: core, example and util.

- 'core' package: an outline for the implementation of the decision tree learning algorithm covered in AIMA Section 18.3 provided by Professor Ferguson.
    - Variable: a variable that represents an attribute that has a name and a Domain of possible values.
    - Domain: a set of possible String values of a Variable (attribute) .
    - YesNoDomain: a subclass of Domain that consists of two values "Yes" and "No" that corresponds to boolean classes of the Variables.
    - Example: a value for each input Variable and a single output value (or classification).
    - Problem: a list of input Variables with their Domains and a single output with its Domain. The inputs are stored as a List.
    - Decision Tree: a recursive dynamic tree structure. There are two types of Decision Tree: if its label is a String, it is a result (leaf) node, otherwise its label is a Variable. This class includes a method to evaluate a learned tree, compute the results on the given Examples and print results and summary statistics (percentage of correctly predicted results and the $L_{1/0}$ loss function)
    - Abstract Decision Tree Learner: a skeletal implementation of the decision tree learning algorithm in AIMA Figure 18.5 with the abstract methods required in the Decision Tree Learner subclass to complete the implementation.
- 'util' package: an ArraySet implementation backed by an ArrayList.
- 'example' package: includes datasets for the restaurant WillWait problem described in AIMA and a discrete Iris dataset that are used for testing the algorithm. I also use the Iris dataset to do further evaluation of my machine learning algorithm with several parameters such as the percentage Learning curve, the Empirical Loss function and the Entropy function.

B. Decision Tree Learning

For this part of the project, I implemented directly the pseudocode for "ID3 decision-tree learning" algorithm described in AIMA Figure 18.5, as shown below:

```
function DECISION-TREE-LEARNING(examples, attributes, parent_examples) returns
a tree

  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
      A ← argmax_{a ∈ attributes} IMPORTANCE(a, examples)
      tree ← a new decision tree with root test A
      for each value v_k of A do
          exs ← {e : e ∈ examples and e.A = v_k}
          subtree ← DECISION-TREE-LEARNING(exs, attributes − A, examples)
          add a branch to tree with label (A = v_k) and subtree subtree
      return tree
```

The implement is fairly straight-forward, given the the well-designed structured of the code provided by Professor Ferguson. The only extra piece of work is to implement the PLURALITY-VALUE and IMPORTANCE methods. To obtain the most common output value among a set of Examples, I create a HashMap table that maps the frequency to each of the output values and find the Key with the maximum frequency.

I do appreciate the extra help with the methods to compute the entropy values of the output variables given sets of Examples. Entropy is a measure of uncertainty of a random variable, acquisition of additional information corresponds to a reduction in entropy. Thus the function $argmax_{a \in attributes} IMPORTANCE(a,examples)$ that returns the prioritized attribute with maximum gain on information can be computed by minimizing the remaining entropy after testing such attribute. As opposed the boolean case described in AIMA, this implementation uses the full definition of entropy and information gain in order to do classification for more than two classes (like in the Iris Dataset).

I implement a method to check if all of the classifications of a set of Examples are the same and return that unique classification, and null otherwise. Also, as discussed in the algorithm design, the result nodes, labeled with a String, are implicitly incorporated with the recursive calls in my implementation.

C. Testing

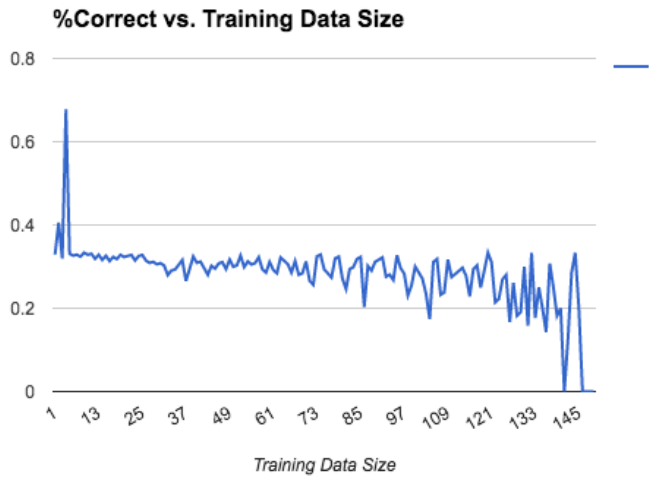
As required by the assignment, I test my algorithm on the restaurant example and and the discrete Iris dataset. Both tests produced desirable outputs. The results predicted by the algorithm matched the results for the restaurant WillWait example perfectly throughout multiple runs. For the Iris dataset, the results matches 84.67% of the true output values and the $L_{0/1}$ loss function is 23, which means the predictor missed out on 23/150 of the examples. In my opinion, this accuracy is quite impressive given the fact that this dataset is not entirely realistic as the attributes Sepal Length, Sepal Width, Petal Length and Petal Width are obtained by running classifications on the true measurements into four classes *{"L","S","ML","MS"}.* Also, I print out all the predictions for all the input values in the and a rough representation of the learned tree in the command line.

D. Evaluation

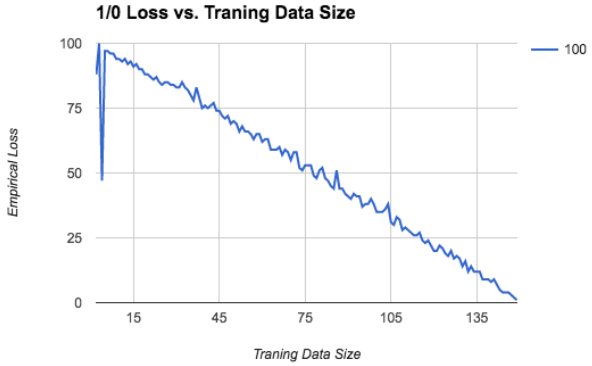
As suggested by the percentage of correct prediction by the algorithm given the input values in the Iris data set, 23 examples were not classified correctly or could not be classified uniquely. This means that these examples have the same input values of the attributes but different classifications due to these reasons:

- There are errors or noise in the data, which is highly probable because the discrete dataset is obtained through classification of the numeric dataset.
- The domain is truly non-deterministic, for example, there exists an unknown class of Iris flower, in which case no function can predict the classification.
- There are some attributes that could distinguish the examples but we cannot observe them.

Of all those reasons, the one that I suspect the most is the noise in the data. In order to make clear of this, as there are 150 examples in the dataset, I split the dataset into a training set and a testing set, starting with a randomly selected set of training examples of size 1 up to size 149. Then I test the accuracy of the model by plotting the learning curve with the percentages of correct prediction, as shown below:

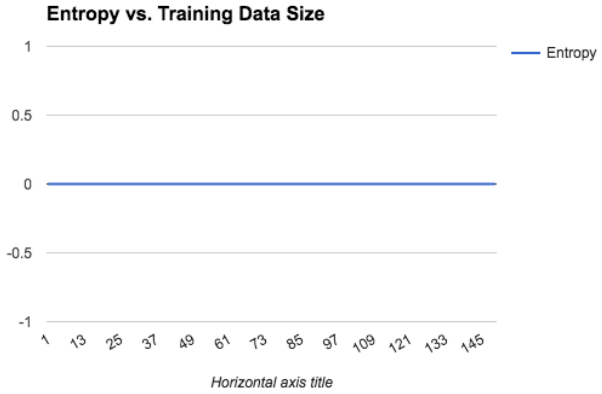


Observing the plot, the prediction results are very poor when testing on outside of the training data. In addition, when the size of the training data is 149, the learned decision tree cannot correctly predict the only result left in the testing set. This means the model fits the training data well yet does poorly on unseen data points, which is a symptom of overfitting. The learned tree maybe accurate on the training data, but it does not generalize to new examples. Another evidence of this phenomenon is the plot of the Empirical Loss function. Although the $L_{1/0}$ does approach 0 as the size of the training set increases, this does not show much in terms of accuracy because the size of the testing set decreases as the result. As stated in the textbook, the generalization loss of a predictor can be estimated by the empirical loss on the set of testing examples. The plots of $L_{1/0}$ loss function and the Empirical Loss function are shown below:



As the loss of generalization approaches one in the Empirical Loss function plot, it is clear that the learned decision tree fails to generalize to examples outside of its training set.

Finally, to show the noisiness of the dataset, I plot the Entropy function as discussed in the algorithm. Generally, we want the Entropy function to approximate 0 but remain larger than 0. However, in this case, all the Entropy values are strictly 0, which means that the output values of the examples provide no additional information on the importance of the attributes incorporated in the decision tree algorithm. This means the decision tree can possibly be a “switch” statement that matches input values to output values instead of estimating the hypothesis.



E. Conclusion

Despite its drawbacks in accuracy, my implementation of the decision-tree learning algorithm does a fairly good job at illustrating a simple and consistent machine learning lesson.

Some improvements can also be made to make my program such as implementing a more complex machine learning algorithm such as linear classifier to estimate the numeric continuous data or obtaining more accurate and larger dataset than the discrete Iris dataset to better train the learning tree.

Instructions on how to run the code and sample outputs are also included.