

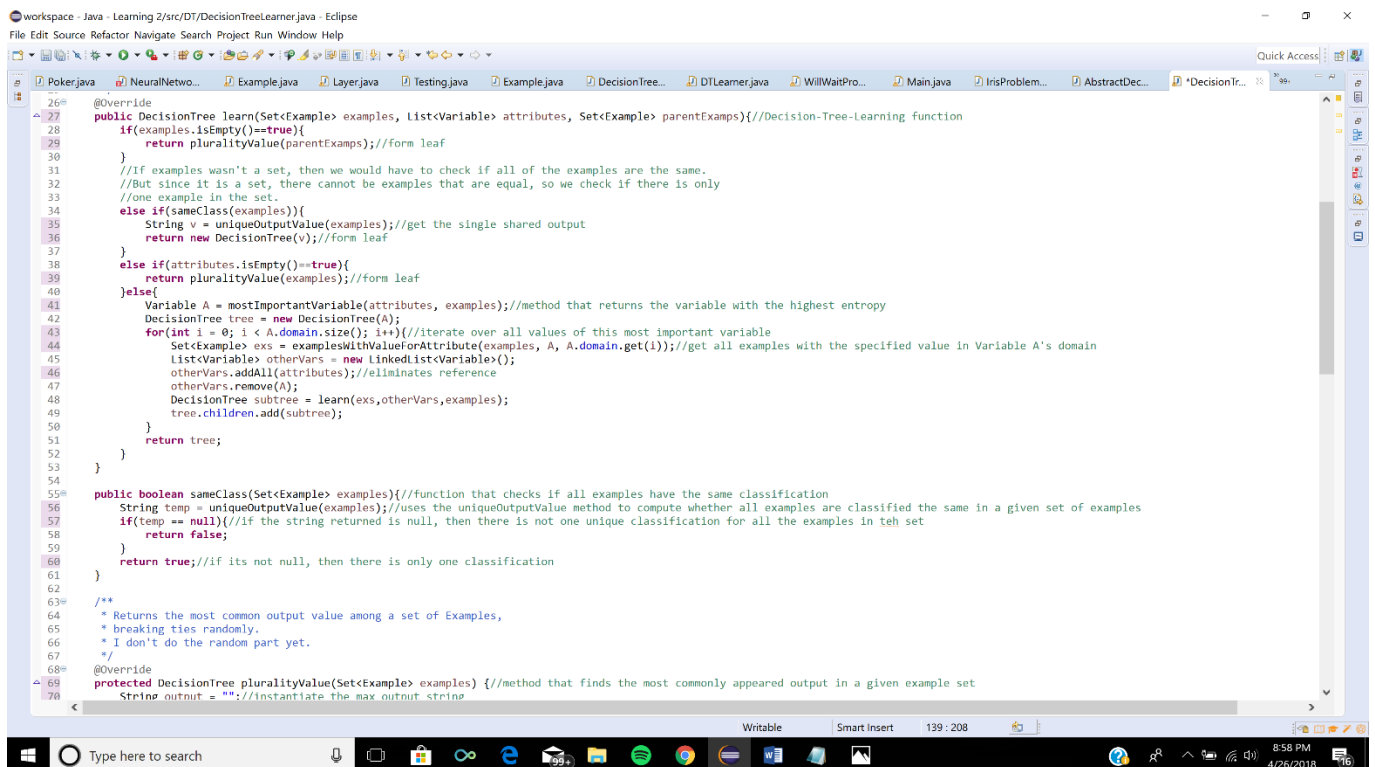
Thomas Burt

CSC 242

## Project 4 Writeup

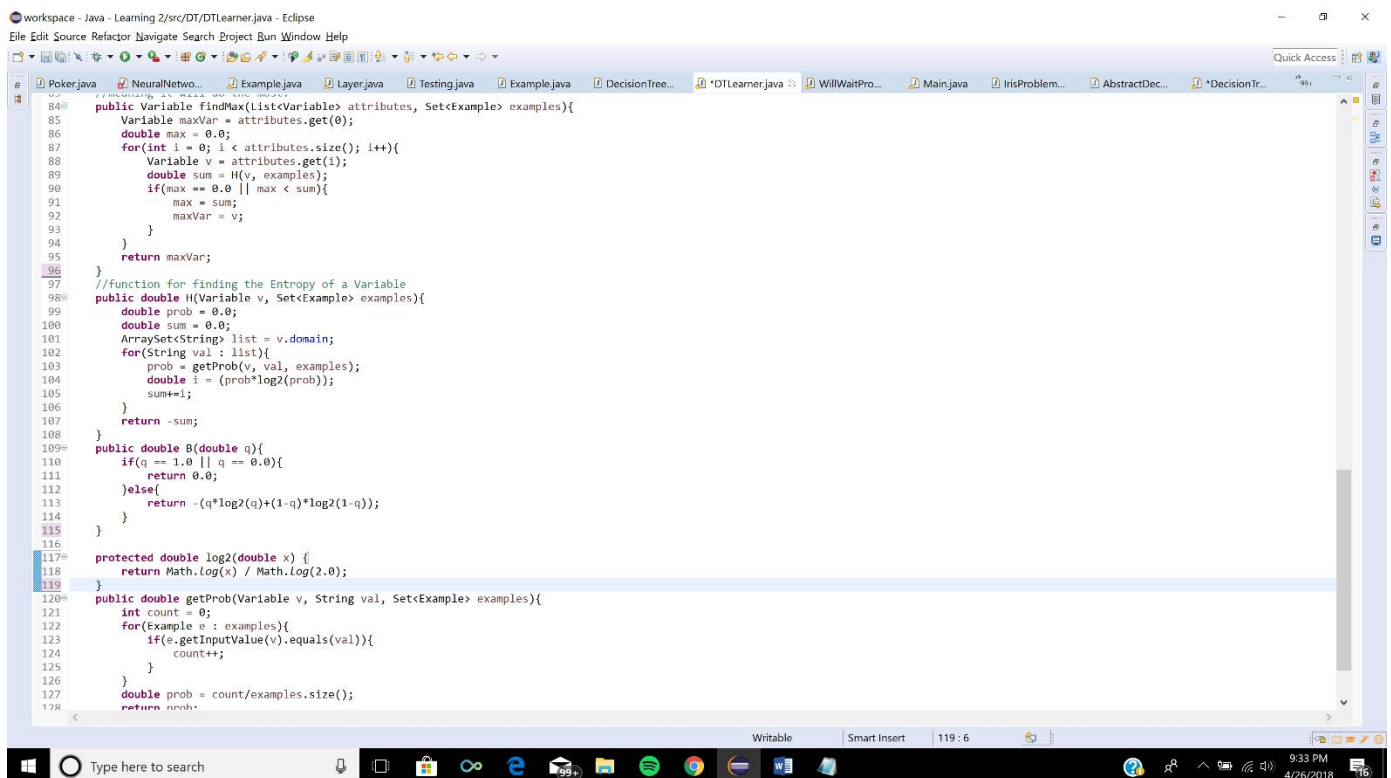
To start off for this project, I had to first decide what data structure I wanted to implement for my project. I was thinking neural networks because they are the most useful out of all the learning methods we were taught. However, I decided to start off with the decision tree and make my way to trying to the other two methods in the future; when I have more time.

The first steps I took at coding up my decision tree was figuring out the representation of it. I started out on my own because I agree with professor Ferguson that you will learn more by doing it on your own. However, this became overwhelming due to various domain sizes for each problem and had a hard time figuring out how to efficiently represent a decision tree on java. So, I decided to use a big portion Professor Ferguson's code for help on the representation. I used Professor Ferguson's shell code in order to better understand how to represent that data model cleanly. I didn't use all of his classes at first, excluding out his DecisionTreeLearning class and AbstractDecisionTreeLearner class. At first, I ran into usual issues regarding referencing when using recursion. This occurred specifically in the learn algorithm in my own DTLearning class where when I eliminated attribute A out of attributes when calling learn again.



```
26 @Override
27 public DecisionTree learn(Set<Example> examples, List<Variable> attributes, Set<Example> parentExmps){//Decision-Tree-Learning function
28     if(examples.isEmpty()==true){
29         return pluralityValue(parentExmps);//form leaf
30     }
31     //If examples wasn't a set, then we would have to check if all of the examples are the same.
32     //But since it is a set, there cannot be examples that are equal, so we check if there is only
33     //one example in the set.
34     else if(sameClass(examples)){
35         String v = uniqueOutputValue(examples);//get the single shared output
36         return new DecisionTree(v);//form leaf
37     }
38     else if(attributes.isEmpty()==true){
39         return pluralityValue(examples);//form leaf
40     }
41     }else{
42         Variable A = mostImportantVariable(attributes, examples);//method that returns the variable with the highest entropy
43         DecisionTree tree = new DecisionTree(A);
44         for(int i = 0; i < A.domain.size(); i++){//iterate over all values of this most important variable
45             Set<Example> exs = examplesWithValueForAttribute(examples, A, A.domain.get(i));//get all examples with the specified value in Variable A's domain
46             List<Variable> otherVars = new LinkedList<Variable>();
47             otherVars.addAll(attributes);//eliminates reference
48             otherVars.remove(A);
49             DecisionTree subtree = learn(exs, otherVars, examples);
50             tree.children.add(subtree);
51         }
52         return tree;
53     }
54 }
55 public boolean sameClass(Set<Example> examples){//function that checks if all examples have the same classification
56     String temp = uniqueOutputValue(examples);//uses the uniqueOutputValue method to compute whether all examples are classified the same in a given set of examples
57     if(temp == null){//if the string returned is null, then there is not one unique classification for all the examples in teh set
58         return false;
59     }
60     return true;//if its not null, then there is only one classification
61 }
62 /**
63  * Returns the most common output value among a set of Examples,
64  * breaking ties randomly.
65  * I don't do the random part yet.
66  */
67 //
68 @Override
69 protected DecisionTree pluralityValue(Set<Example> examples) {//method that finds the most commonly appeared output in a given example set
70     String output = "";//instantiate the max output string
```

The Variables would get messed up during the combination of the iteration of the most important variables domain and recursion. It would eliminate variables at points where these variables were not suppose to be eliminated. I would run into NullPointerException errors. However, I fixed this quite fast considering I always end up running into this problem when coding my projects. The fix was a three line addition which can be seen in the above picture from lines 45 to 47, where I make a temporary copy of the list of attributes minus the most important variable chosen before. Thus the The second challenge was the entropy function in my own DTLearning class. I thought my importance and H functions correctly, but when I tested the data I would get correction proportions at values like 30%. The picture below is how I tried to code the entropy function of the algorithm.



```

84 public Variable findMax(List<Variable> attributes, Set<Example> examples){
85     Variable maxVar = attributes.get(0);
86     double max = 0.0;
87     for(int i = 0; i < attributes.size(); i++){
88         Variable v = attributes.get(i);
89         double sum = H(v, examples);
90         if(max == 0.0 || max < sum){
91             max = sum;
92             maxVar = v;
93         }
94     }
95     return maxVar;
96 }
97 //function for finding the Entropy of a Variable
98 public double H(Variable v, Set<Example> examples){
99     double prob = 0.0;
100     double sum = 0.0;
101     ArraySet<String> list = v.domain;
102     for(String val : list){
103         prob = getProb(v, val, examples);
104         double i = (prob*log2(prob));
105         sum+=i;
106     }
107     return -sum;
108 }
109 public double B(double q){
110     if(q == 1.0 || q == 0.0){
111         return 0.0;
112     }else{
113         return -(q*log2(q)+(1-q)*log2(1-q));
114     }
115 }
116
117 protected double log2(double x) {
118     return Math.log(x) / Math.Log(2.0);
119 }
120 public double getProb(Variable v, String val, Set<Example> examples){
121     int count = 0;
122     for(Example e : examples){
123         if(e.getInputValue(v).equals(val)){
124             count++;
125         }
126     }
127     double prob = count/examples.size();
128     return prob;

```

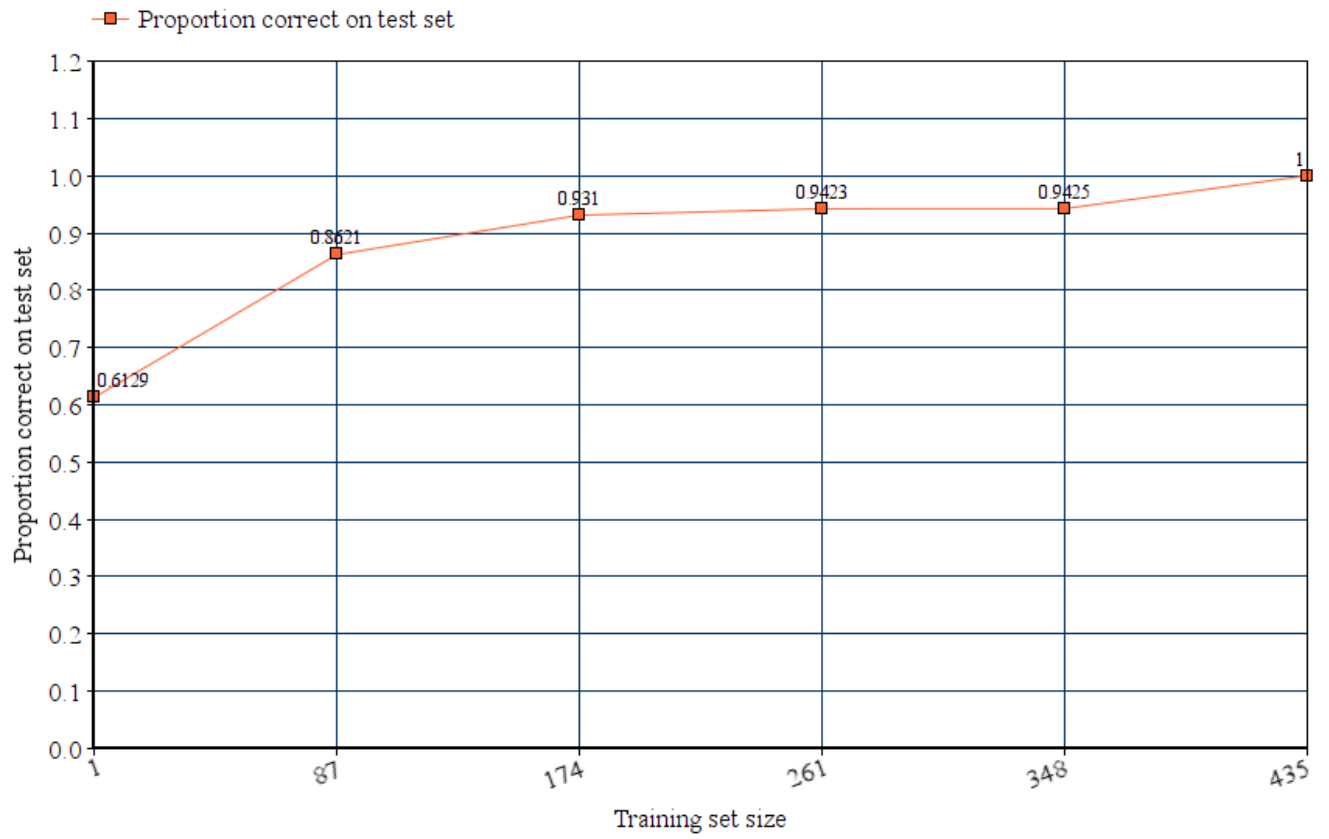
I still cannot figure out what I did. I looked back on the Entropy equation, checked my H function and everything, but it still wouldn't work. After a while, I realized that my code was getting too messy and I wasn't quite going anywhere at this point in the project. At this point, I decided to use Professor Ferguson's DecisionTreeLearner class and AbstractDecisionTreeLearner class. Once I put the shells in for these, I transferred over my code for the plurality function, learn function and utilized Professor Ferguson's method, mostImportantVariable in my project, the function worked. I don't know what I was doing wrong, but clearly something was up with that. Either way the problem was fixed and thus the representation was complete and my coding had no errors.

However, I had been testing the data with the WillWait-data example set that is only of size 12. Since the example set is so small, it is hard to really see the potential of this algorithm completely

When I solved all these issues with the code, I used the small WillWait-data problem example set. This was easy to use for rooting out issues because it is a small example set. This was a very good tool for rooting out my errors, especially the reference error. However, when trying to find how accurate my Decision tree was, I then implemented a second problem. The second problem I used was the Iris.data.discrete.txt file. I renamed it to Iris-data-discrete.txt because I didn't like the '.'. The second problem is an example set of size 150. When I started on this problem, I used the whole example set for training and then for testing as well. When I did this for the Iris problem, the ratio I got correct was only 95.33 percent of the time. When this happened I thought it was suppose to be one hundred percent, which is what happened what I did the same thing for the WillWait-data Problem. However, it came to me that this might be because of overfitting. I then decided I was going to perform analysis on the accuracy of the program for this problem using a similar method as figure 18.7 does in AIMA. I decided to test 11 different training set sizes for the problem. For each size on the x axis, I ran the program 20 times and summed up correctness ratio for the 20 times and then divided that sum by 20 to get the average. Each of these 20 times, the program picks all unique random examples for the testing set. What I was shocked to find out was that the proportion of correctness given each training set size relatively stayed the same from training set sizes 15-150. There is only roughly a 5 percent accuracy advantage of using 150 versus 15. The overall conclusion as to why it wasn't 100 percent accuracy when using all the example set to train the data and then use all of it to test it as well. So the underlying issue was that the Decision Tree would overfit the Iris-data-discrete example set. Thus, it will never be 100 percent where as it would if it didn't overfit.

The third problem I did was to add a problem that didn't overfit. There is a graph below showing that the house-votes-84.data.mod problem took more training data to get good accuracy, or at least as good as the Iris problem. However, this problem ends up being 100 percent accurate when the training set and test set are the same at max example size. To conclude the project, the Decision tree is a solid basic form of machine learning. However, the more variables a problem has, the more of a chance of overfitting and thus not generalizing data well.

# House-Votes Problem Analysis



Iris-data-discrete Problem Analysis

