William Zay

Michael Knapp

Leonard Museau

Decision Tree Model for Mushroom Data Set

Our group had found a mushroom data set to use to build our decision tree. We implemented a generalized ID3 decision tree algorithm based on what we learned in class. We built it in Python 3 using the math, csv, and pprint libraries. We built our decision tree as a recursive function that builds nested dictionaries. Most keys in a dictionary mapped to another dictionary whose keys mapped either to another dictionary or a Boolean value. First, we checked a set of conditions to ensure we had enough data, and if not, we returned a default value. Then we determined the best split by calculating the entropy of every attribute in the data set.

There were several problems we ran into while designing the program architecture. We had to build several versions of this first functions of the program starting with a hard-coded, non-dynamic version that processed the sample training data. This was a very complicated program and it was hard to get a “birds eye view” of the of steps of logic involved in building it correctly. So, the best way for us was to attack each function with a lot of conditionals until we were able to hack out the complicated steps in getting the correct output of the DT. I think maybe some more pre-planning and building some type of flow diagram would help us in the future save time on coding different routes/versions of solving the same logical problem.

Another problem we found, we overlooked calculating the information gain on each attribute and just did entropy. This of course gave us inaccurate results. After fixing that, our algorithm will then select the attributes that has the largest information gain value. We had to set a limit to this though, as some values in the mushroom dataset had so many values it skewed results. Then, the data set was split by those selected attributes to create subsets or branches of the data. Each attribute then created its own decision tree node. This algorithm will continue to repeat on each subset only considering attributes that haven’t been chosen and at the end of each branch is a Boolean value.

Upon reflection it would have been better to make a class-based implementation. This would have mode building the tree and pruning it easier. I would have like to use recursion to find the optimal decision tree. As it is we had to use a knowledge of the datasets to prune our trees. For the hiring data set this just required cutting out the language portion of the branches. For the mushroom dataset we have no concerns of overfitting, as we were able to have a branch depth of one. This was possible because the odor trait alone correctly identifies 98.5% of the poisonous mushrooms. For the odors that don’t automatically map to poisonous 97% of them are edible and the most common value of all sub branches is edible, or True. This method may not lead to the same accuracy as recursive pruning, but it is very good, and has a much lower time complexity.

The main way we thought to improve this implementation on the mushroom set is to trim other attributes, so the loop doesn’t go through all the keys since odor is the only one that matters. To test this, I ran it both ways five times and averaged the times. Both times averaged to about 19.5 seconds when I realized that most of the time was IO related so I removed the print statements. Both took less than a second, but the odor only one consistently returned 0.0 so it may have been a little faster.