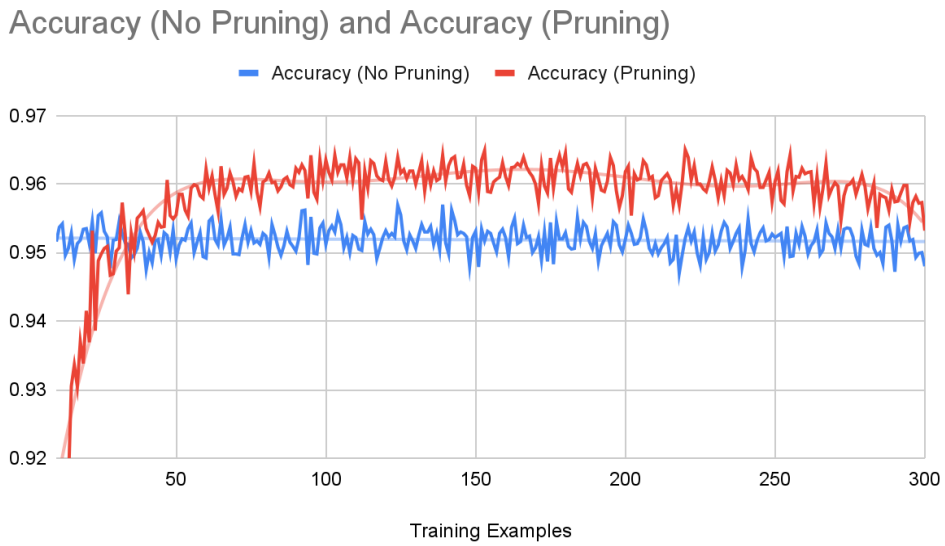
1. **Did you alter the Node data structure? If so, how and why?**
   Yes. The children are now in a hashmap (Python dictionary) where every key is one value of the node's attribute (the branch) and every value is the node corresponding to that specific value of that attribute. We did this because there exists no real branch from one node to another. For example, if we're currently on a node with label `'Patrons'`, we would only have its children—`True`, `False`, and another node with label `'WaitEstimate'`—with no way of getting to each child based on the attribute value of `'Patrons'`. With key-value pairs, we can know look up the next node based on an attribute's value. All other functionality is in helper functions residing in `ID3.py`.

2. **How did you handle missing attributes, and why did you choose this strategy?**
   We handled missing attributes in two ways. For missing attributes in the training data, we cleaned the data before we ever trained on it. For every data point, we replaced each missing attribute value with the most common attribute value for the class that the data point belongs to. For example, if we had a data point with a missing `'Patrons'` attribute, we would simply give it the most common attribute value of the class `'Class'`. This was done in order to maximize the accuracy of the model while minimizing the impact of the majority attribute values. The data point d with a missing attribute value v belonging to a class c is statistically more likely to have the majority attribute value belonging to class c. Not only that, but doing so would skew the dataset the least. If there are 3 positive attribute value, 4 negative attribute values, and 100 missing attribute values, then our algorithm wouldn't just assign the 100 missing values the negative attribute value. Instead it would look at each data point with a missing value's class, and assign it the majority attribute value of that class. That way, our data can be more balanced.

3. **How did you perform pruning, and why did you choose this strategy?**
   We chose reduced error pruning algorithm mainly because it is a simple algorithm to follow and because of its time complexity. At each non-leaf node of our tree, we calculate the accuracy if the node we're instead replaced with a leaf node. This way, we traverse the tree exactly once. This option was the one with the least search space, which was the most practical for us given we are running this program locally. We replaced each non-leaf node with the common class of that node. For example, if a non-leaf node classifies an example `True` 3 times and `False` 2 times, then we would assign the node a value of `True` and remove its children. If this lead to a worst accuracy then before we replaced the node, then we simply revert the node back to what it was and recurse on its children.

4. **Now you will try your learner on the house_votes_84.data, and plot learning curves. Specifically, you should experiment under two settings: with pruning, and**

without pruning. Use training set sizes ranging between 10 and 300 examples. For each training size you choose, perform 100 random runs, for each run testing on all examples not used for training (see testPruningOnHouseData from unit_tests.py for one example of this). Plot the average accuracy of the 100 runs as one point on a learning curve (x-axis = number of training examples, y-axis = accuracy on test data). Connect the points to show one line representing accuracy with pruning, the other without. Include your plot in your pdf, and answer two questions



What is the general trend of both lines as training set size increases, and why does this make sense?

On average, the accuracy marginally decreases as the training set size increases. This makes sense because as we give the model more training data, it gets better at classifying the training data. However, it learns the training data up to the point where it doesn't generalize to the test data. We overfit as we get more training data.

How does the advantage of pruning change as the dataset size increases? Does this make sense, and why or why not?

As the training data size increases, we get better accuracy from our pruned model. In fact, the increase from 10 to 50 training examples greatly improves the accuracy of the pruned model. However, as our training data size increases after this, we get diminishing returns. In fact, after around 200 examples, the accuracy only goes downward. This makes sense because as we first give it more examples, the model gets better at making predictions for the training data, then it gets fined tuned to improve the accuracy. However afterwards, there is only so much we can prune to improve the accuracy since the model is already very representative of the training data examples.