ML HW 1 Group - 13

Q1. Did you alter the Node data structure? If so, how and why?

Yes, we altered the Node data structure, we added the ‘attribute’ variable. The attribute variable shows the attributes on which the decision tree was split.

Q2. How did you handle missing attributes, and why did you choose this strategy?

We added a ‘impute_missing_value’ function which replaces the missing value with the most probable value of that attribute. We did this because of the following reasons:

- Number of rows with missing values were very high, specifically there were less than 50 rows where there weren’t any missing values in the row. Removing them would have significantly reduced the dataset size. Hence replacing the missing value was the better alternative.
- Replacing the missing values with most probable values would not change the probability distribution of individual attribute and keep the data drift at minimum.

```
def impute_missing_value(dataset):
    attributes = list(dataset[0].keys())
    attributes.remove("Class")

    for attribute in attributes:
        # Get the values for the attribute.
        attribute_values = [example[attribute] for example in dataset]
        # Find the mode for non-missing values.
        non_missing_values = [value for value in attribute_values if value is not None and value != "?"]
        mode = max(set(non_missing_values), key=non_missing_values.count)

        # Update missing values in the dataset with the mode.
        for example in dataset:
            if example[attribute] is None or example[attribute] == "?":
                example[attribute] = mode

    return dataset
```

Q3. How did you perform pruning, and why did you choose this strategy?

We performed ***critical value pruning***. We considered two things while building a decision tree:

- Accuracy on the test dataset
- Complexity of the tree

So, although accuracy is directly proportional to the complexity of the tree, more complex a tree is, more it tends to overfit on the dataset. Secondly, complexity of the tree also increases the

computation time for making a decision. We focused on gain in accuracy with respect to marginal increase in complexity.
Keeping these things in mind, we decided to eliminate the sub-tress which showed accuracies lower than a chosen threshold. Since critical value pruning achieved this we decided to choose this strategy.

```python
def prune(node, examples, critical_value=0.5):

  """
  Takes in a trained tree and a validation set of examples. Prunes nodes in order
  to improve accuracy on the validation data; the precise pruning strategy is up to you.

  Args:
    node: The tree node to prune.
    examples: The validation set of examples.
    critical_value: The critical value used to prune nodes.

  Returns:
    The pruned tree node.
  """

  # Calculate the accuracy of the current node on the validation data.
  accuracy = test(node, examples)

  # Prune the node if its accuracy is below the critical value and it is not a leaf node.
  if accuracy < critical_value and node.children:
    node.label = None
    for attVal,child in node.children.items():
      # Need to pass on the specific node wise dataset for accuracy prediction
      prune(child, utility.getDataWithAttValue(examples, node.attribute, attVal), critical_value)

  # Return the pruned node.
  return node
```

Q4. Explaining the general trends with increased dataset and results of accuracies with and without pruning.

As expected, the ***training accuracy increased with the increase in sample size.*** This happened because with more datapoints the decision tree will have less tendency to overfit on data resulting in increased accuracy. More datapoints also have the benefit of better generalization, because with less number of datapoints, there is a possibility that the tree will memorize the specific examples rather than making generalized decisions.

For pruning we choose critical value pruning method. Below are our observations on accuracies of our decision tree before and after pruning:

Accuracies before pruning:

For our house dataset we generally had accuracies over 94 for a good enough sample size. In the case of unit test we had accuracy around 95.

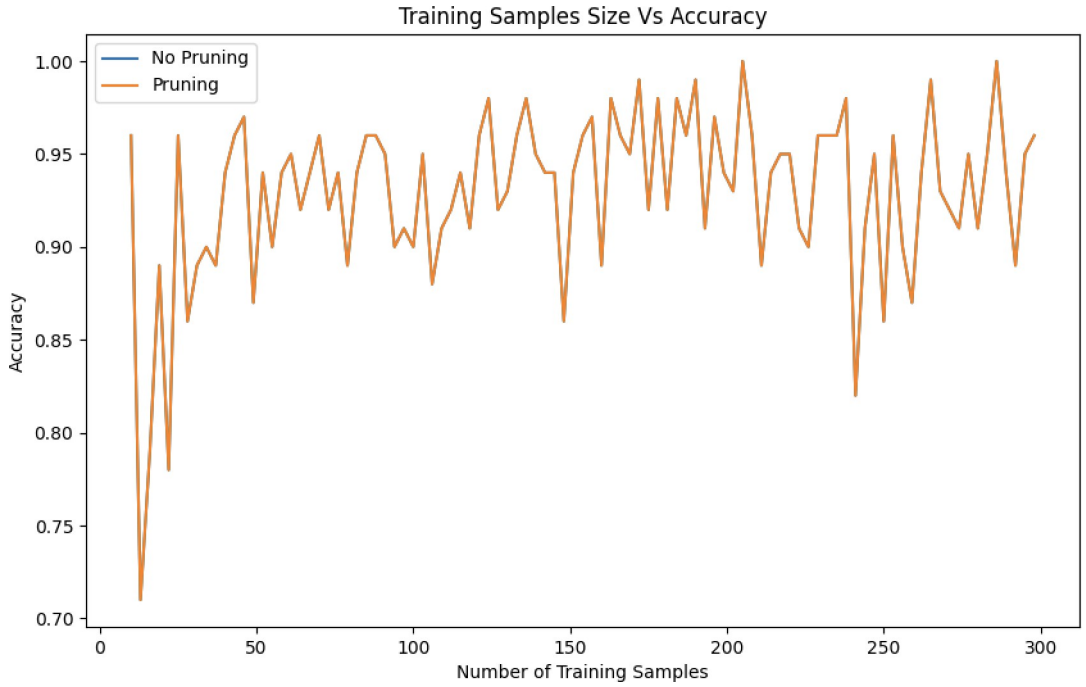
Accuracies after pruning:

Although, For house dataset we observed increase in the accuracies in the sample sizes, it was marginal. This happened because we already had a very high accuracy before pruning itself. In the case of unit tests we were able to observe noticeable increment of 4-5% in accuracies after pruning, hence, showing the effectiveness of our pruning algorithm.

Advantage of pruning:

Fundamentally pruning algorithm are used to reduce overfitting, and overfitting is more prominent with small data sets. So, In our algorithm, as data size increased we observed less incremental effect of pruning because the model itself din't showed overfitting with sufficiently big sample size.

Graph showing accuracies with and without pruning over an incremental training dataset:



Q5. Random forest implementation and results

Two critical parameters in designing a random forest are:

- Training Size
- Number of trees

We decide on a Training size on which we want all our trees in random forest to be trained on. For example, let's say we have a dataset of 100 data points. If we decide that our training size is 80, this means that all our trees will be individually trained on 80 randomly selected data points from the entire set. The importance of this parameter being that we can introduce randomness in the training phase and allow different trees to discover unique slices of data that exist.

Too small of a Training size would lead to under fitting of trees and as a result, the entire Random forest algorithm. Too big of a Training size would mean that essentially all the trees are similarly trained and there is not enough variance between them.

Again to test this parameter, we experimented with incremental values of sample sizes starting from 10 to length of dataset.

Further we have to decide the number of trees in a random forest. For deciding this parameter, we went ahead and tested the random forests with different numbers of trees from 1 to 10. We decided to experiment with the random forest that provided the most gain in terms of accuracy on the test set.

Overall, we chose the random forest with the highest accuracy on the dataset. Practically, we concluded that RF performed best on the Candy Dataset when Training Size was 10 and number of trees were 2. This particular RF performed with 88.88% Accuracy which was 33% better than the regular Tree’s accuracy.