HW1 binary decision tree
Yanbing Wang
9/16/19

## A summary for learning a decision tree classifier

The decision tree algorithm takes in a node class, which contains training data as an argument. `TreeGrowth(node)` recursively finds the optimal split between the dataset according to either weighted Gini index or entropy ratio. In each recursive step, the two subsets from the best split are then assigned as the children to that node.

***Pseudocode:***

```
TreeGrowth(node):
    if stopping_cond(node) == true
        leaf = createNode()
        leaf.label = Classify(node)
        return leaf
    else
        left, right = find_best_split(node)
        make left and right children of node
        left = TreeGrowth(node)
        right = TreeGrowth(node)
    end if
  return node
```

***Helper functions:***

`nodeClass(NodeMixin):`

has features name, dataframe, split_feature, split_threshold, parent, children,label. Name, parent and children are the built-in features of "node" class from anytree. All the other features are customized.

`split(node, split_feature, split_value):`

split a node by the split_feature and split_value, return two nodes to be potentially assigned as children of node.

`gini_index(node):`

return the gini index of node

`total_gini(node1, node2):`

compute the weighted gini index by the number of training examples in each node,

`entropy(node):`

return the entropy

`total_entropy(node1, node2):`

compute the weighted entropy by the number of training examples in each node,

`stopping_cond(node,gain_measure):`

according to either gini index or entropy as the gain_measure, return true if the node is pure or only 1 piece of training example is left. Otherwise return false.

`get_split_values(data,column):`

return a list of all the potential split values obtained from averaging two adjacent unique values of each feature column.

`computeOptimalSplit(node, gain_measure):`

iterate through all the features and all the values in each feature to get potential splits. For each potential split, calculate either gini or entropy according to the desired gain_measure, and return the best feature and best threshold that contribute to the lowest entropy or gini (purest).

`classify(leaf):`

classify a leaf node as 0 or 1 according to the maximum probability of observing 0 or 1.

`predict(example, node):`

pass example to the root (node) of a tree, traverse that example according to the split condition of that node until reaching the leaf node. Return the label of that leaf node.

`get_predict_list(dataset, node):`

pass a dataset where each row is an example. Apply predict() to that example iteratively.

`calc_accuracy(test_data, node):`

node as the root of a tree. Return the accuracy of test_data on the trained decision tree.

## A summary for pruning a tree

*This write-up only applies to removing one single node from the tree.*

***Helper functions:***

`possible_pruned_trees(base_tree):`

Iterate through all the nodes in the unpruned decision tree. For each node, remove all the descendants of it to form a possible pruned tree. Put all the possible pruned trees in a list.

`count_nodes(tree):`

return number of nodes in the tree.

`calc_f1score(test_data, node):`

return f1_score of test_data on the tree.

`pruneSingleGreedyNode(validation_set, base_tree):`

iterate through all the possible trees obtained from possible_pruned_trees(). Calculate the accuracy of all the possible trees tested on the validation_set. Get the best tree that produces the largest increase or smallest decrease in accuracy. If multiple best trees are reported, choose the one that as the least number of nodes.

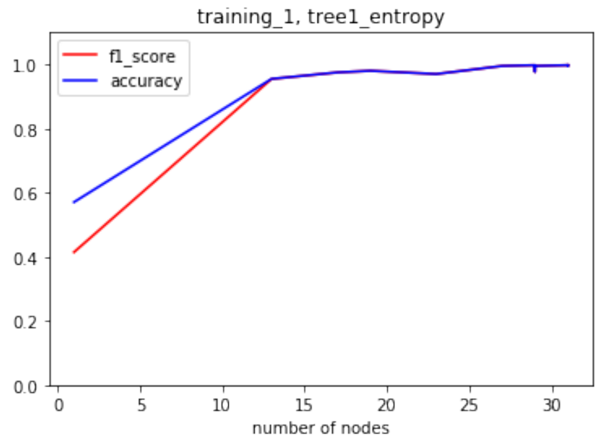
## Data analysis

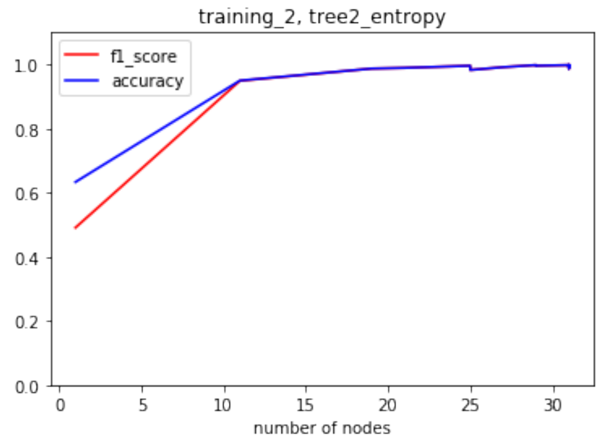
*Table 1: Accuracy table for unpruned trees. Tested with testing tests.*

| Gain measure | Train : test | # training set | # nodes | # leaves | # accuracy % |
|---|---|---|---|---|---|
| Entropy ratio | 8:2 | Tree 1 | 33 | 17 | 90.35 |
| | 9:1 | Tree 2 | 33 | 17 | 91.23 |
| Gini index | 8:2 | Tree 1 | 33 | 17 | 90.35 |
| | 9:1 | Tree 2 | 33 | 17 | 91.23 |

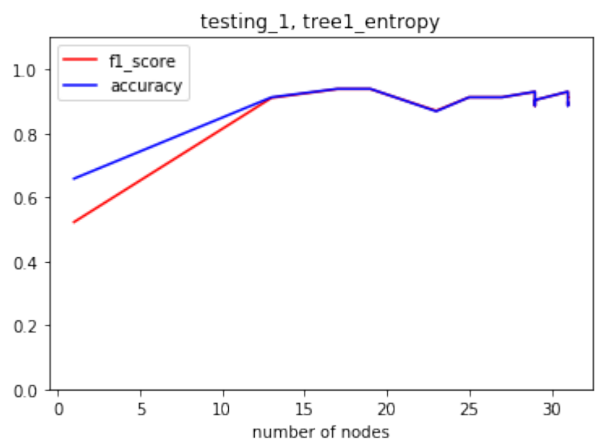
*F1 score and accuracy of each tree tested on training, testing and validation set.*

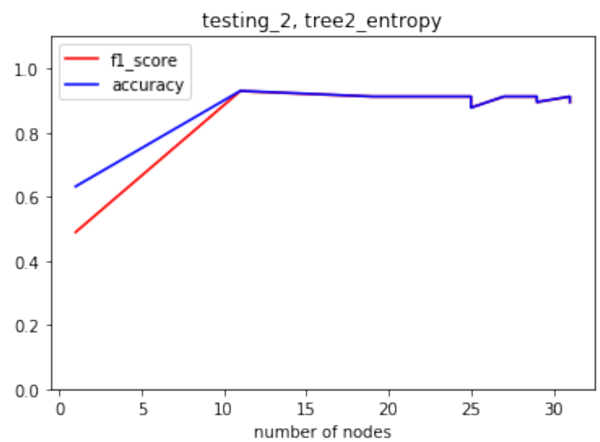
## Training set



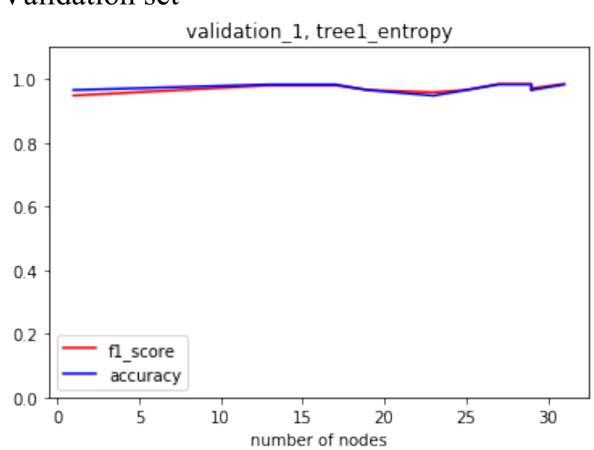



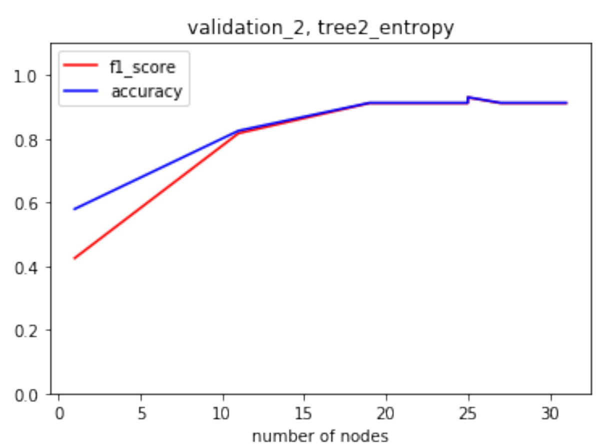

## Testing set




## Validation set



Discussion:

(1) Training set shows almost monotonically increase in f1 score and accuracy wrt to the number of nodes. This is because the unpruned tree was trained using the exact same training set data. Thus any removal of the tree will likely lead to compromise in f1 score and accuracy.

(2) Note that when number of nodes exceeds around 12, f1 score and accuracy overlaps, meaning that the false positive and false negative are balanced.

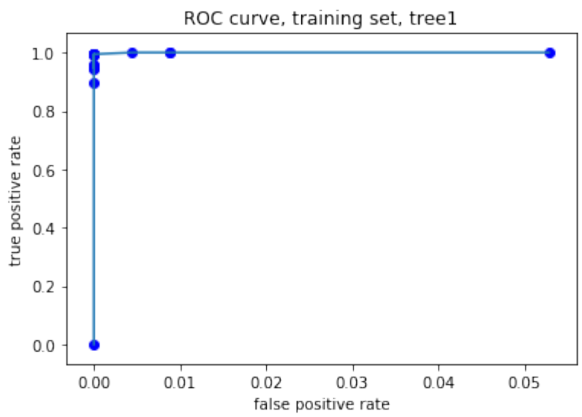
(3) From testing set: at around node# = 12-16, the classifier gives the highest f1 score and accuracy. This suggested that the original unpruned tree tends to overfit.

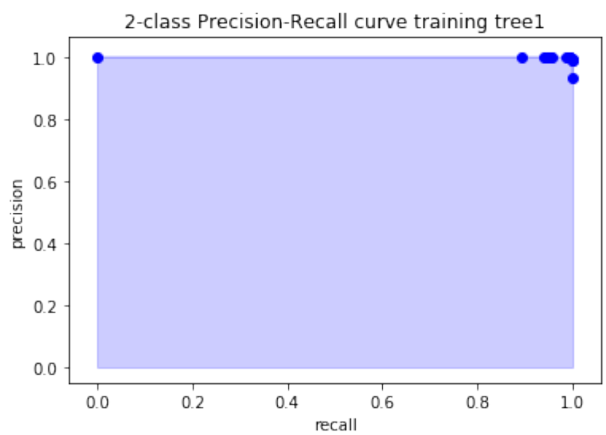
(4) From validation set: validation set 1 is extremely biased and does not give a good generalization for testing the proposed classifier. Validation set 2, on the other hand, is more balanced and is a better indicator of the fit of the classifier.

(5) From the plots, the best pruned tree for training set 1 is around 19 nodes, and for training set 2 is 11 nodes.

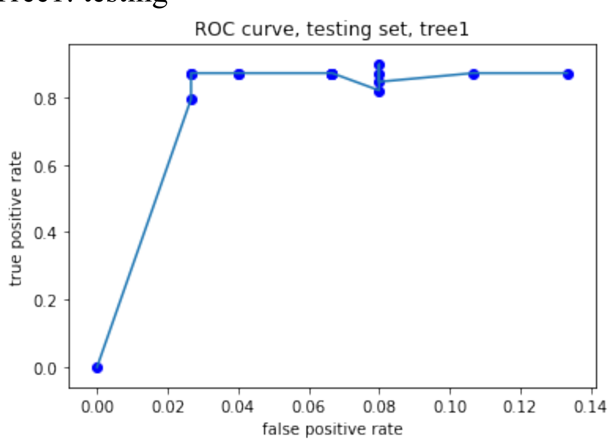
*Plot precision-recall curve and ROC curves*

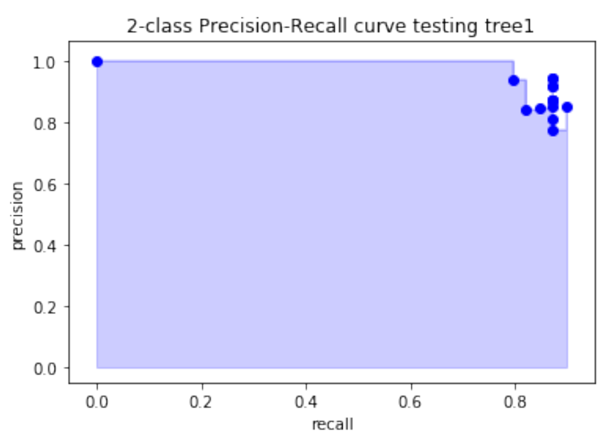
Tree 1: training



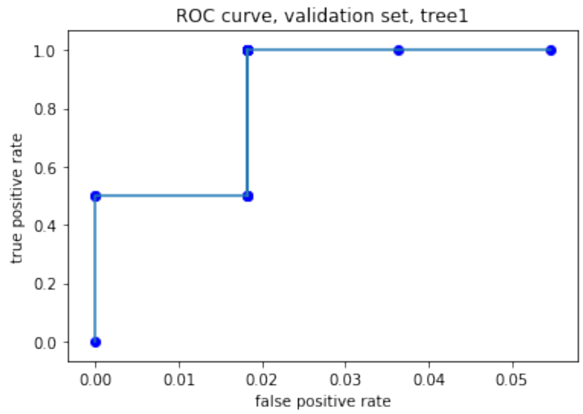



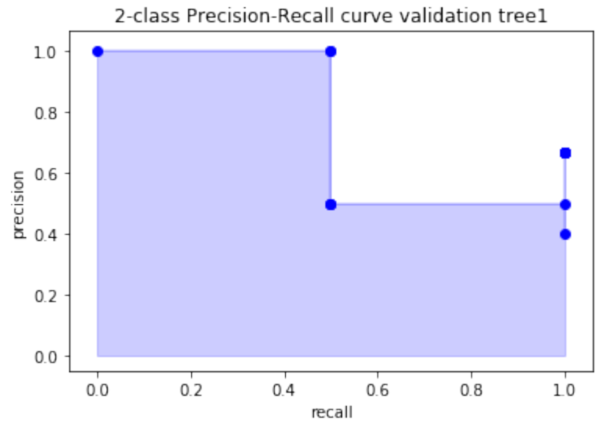

Tree1: testing



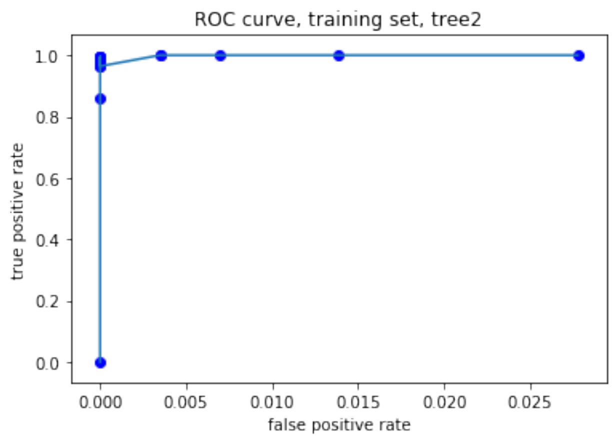



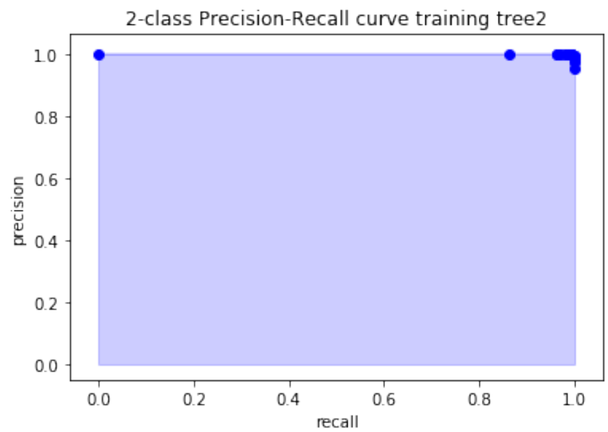
## Tree1: validation



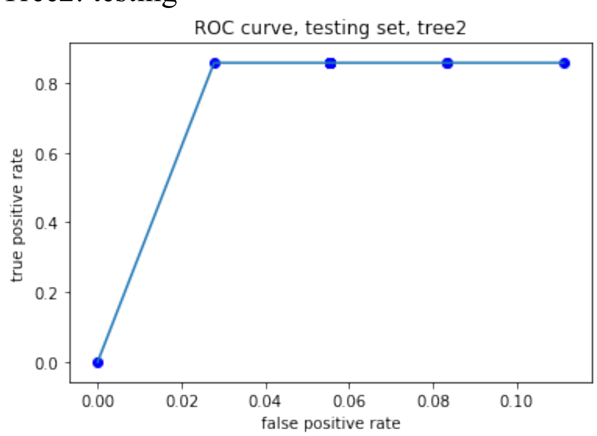



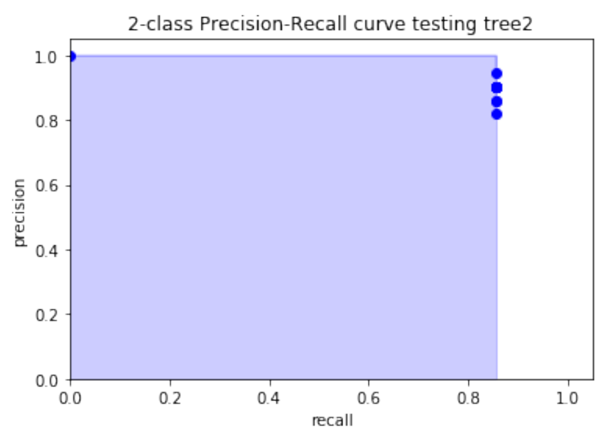

## Tree2: training



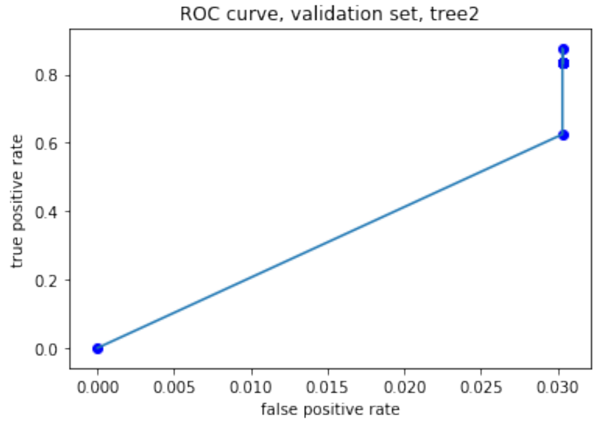



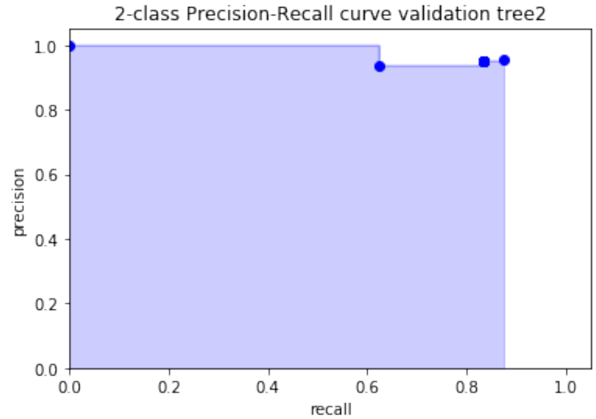

## Tree2: testing



Tree2: validation




Discussion:

(1) Training set: PR and ROC curves for both training sets look perfect, indicating that each of the possible pruned tree can almost perfectly classify positive and negative labels.
(2) Testing sets: the curves of both trees look pretty similar for testing set. The optimal pruned tree is easy to tell. Tree1 has 17 nodes after pruning, and tree2 has 11 nodes.
(3) Validation sets: due to the extreme bias of validation set 1, the area under ROC curve is extremely large. In this case, PR curve seems to be a better measure, but the best tree is slightly difficult to tell from PR curve alone. The second validation set gives a more reasonable ROC curve, indicating a more balanced data.
(4) Compared to the measure using f1 score and accuracy curve, both measures agree that the second tree is much more overfitted than the first one, although the best pruned tree produced by each measure is slightly different. Table 2 summarizes the best tree properties produced by each of the two measures.

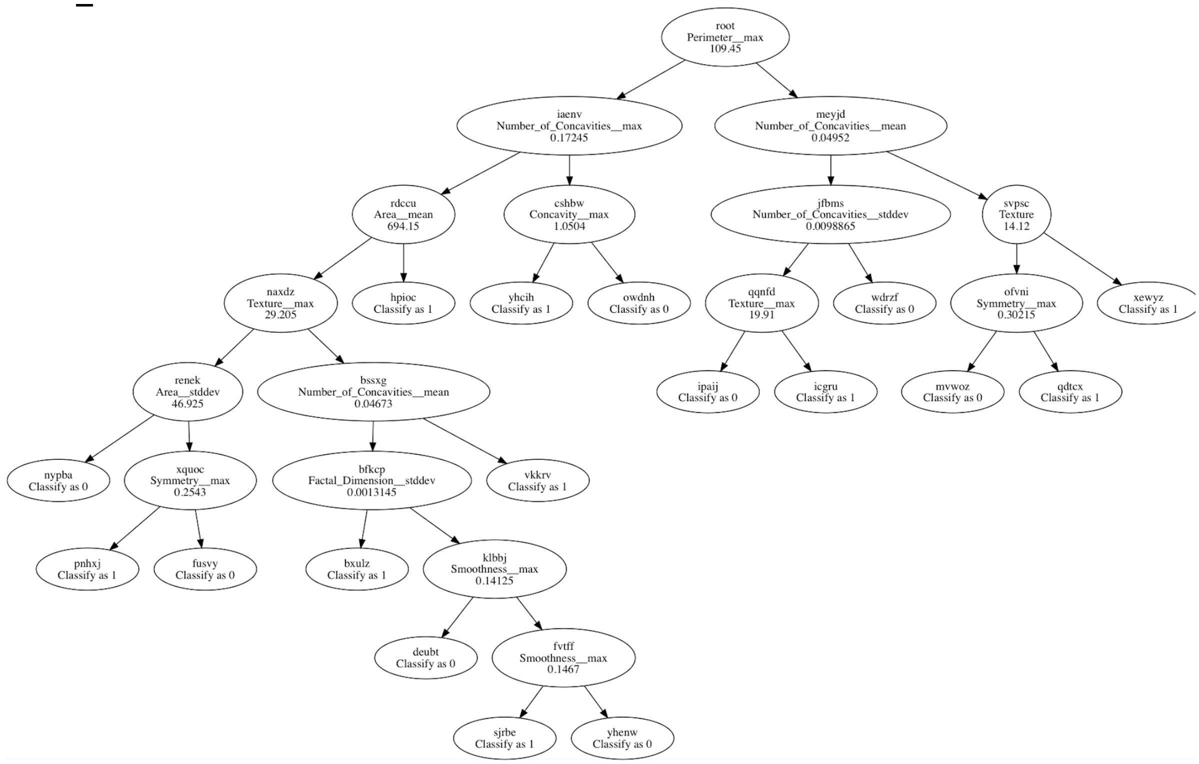
*Table 2: Properties of the best tree obtained from ROC/PR curves of each dataset*

| Dataset | Tree | Node length | Accuracy | F1score | Fp_rate | Tp_rate | recall | precision |
|---|---|---|---|---|---|---|---|---|
| F1 and accuracy | 1 | 19 | 0.979899 | 0.979833 | 0.000000 | 0.953216 | 0.953216 | 1.000000 |
| | 2 | 11 | 0.986813 | 0.986761 | 0.000000 | 0.964072 | 0.964072 | 1.000000 |
| PR and ROC | 1 | 17 | 0.912281 | 0.910313 | 0.026667 | 0.794872 | 0.794872 | 0.939394 |
| | 2 | 11 | 0.912281 | 0.911812 | 0.055556 | 0.857143 | 0.857143 | 0.900000 |

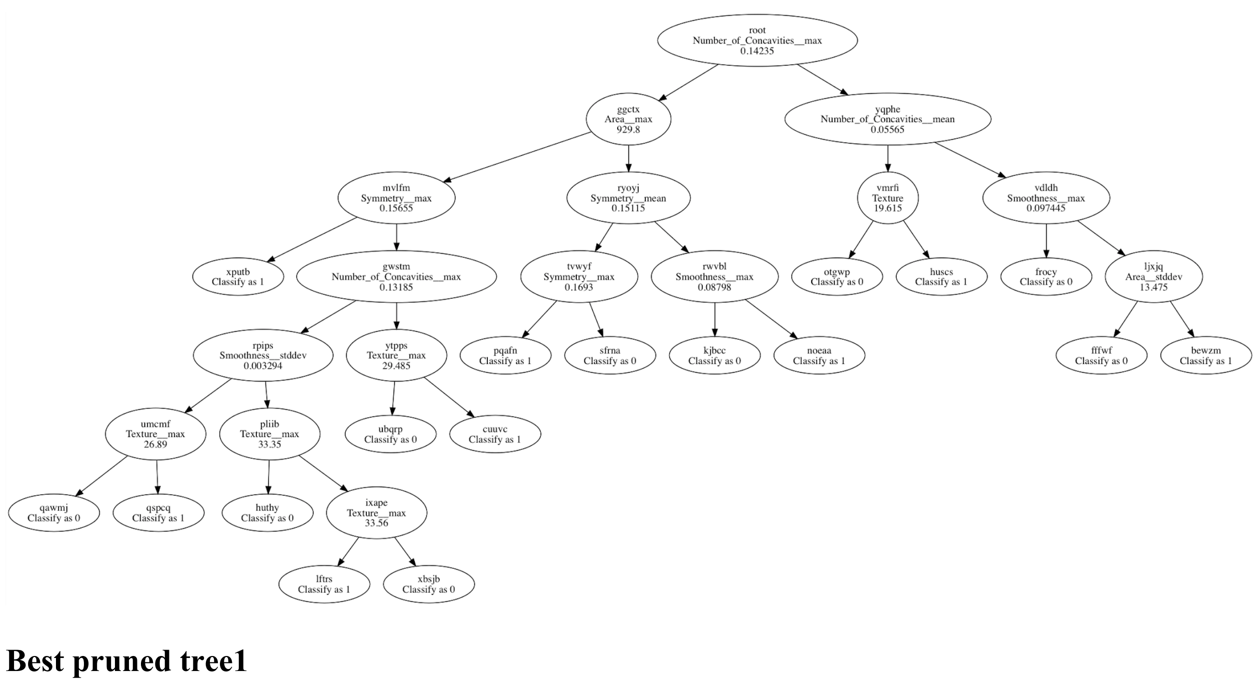
## Appendix

**Tree1: use entropy ratio, unpruned**

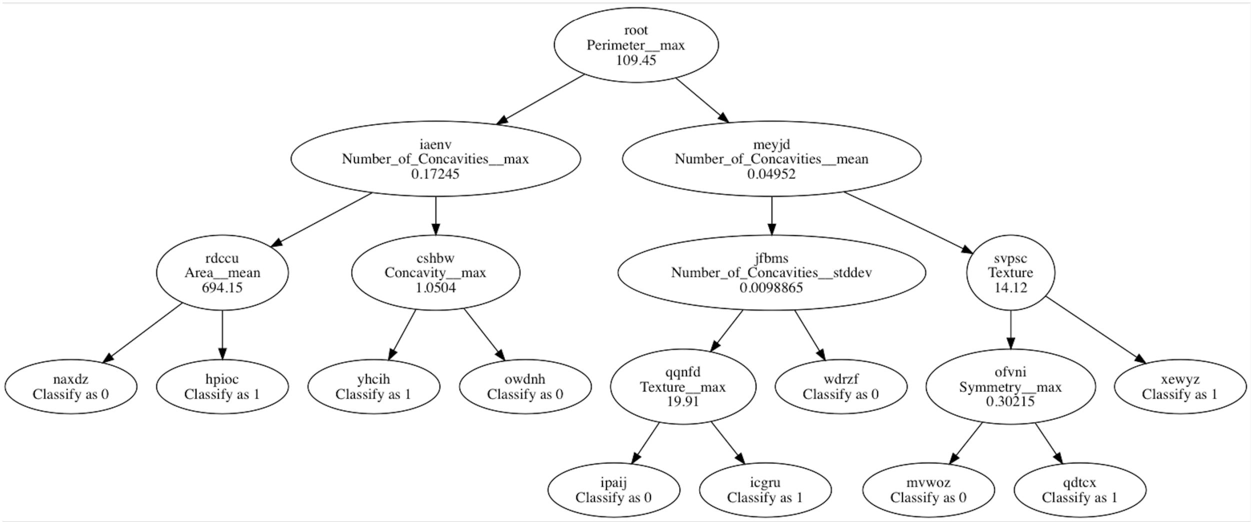
**Node_count: 33**




## Tree2: use entropy ratio, unpruned
## Node_count: 33




## Best pruned tree1
## Node_count: 19




**Best pruned tree2**
**Node_count: 11**