Documentation for main functions

YUE HU

```
def __computeOptimalSplit(self, X, y, criterion):
“’ Private method that returns the best single split that produces the maximum gain.
   X: training feature data
   Y: labels for training data
   criterion: gini or entropy.
“’
        calculate the impurity of current node
        set current best gain as 0
        initialize best feature and best split as None
        for (each feature in training set):
                list all distinct values in increasing order
                find the splits, each split is the middle of two consecutive values
                for (each split):
                        divide the data based on the splitting point
                        calculate the impurity value of the two subsets
                        if (criterion is gini):
                                calculate weighted gini gain
                        if (criterion is entropy):
                                calculate  entropy ratio gain
                        if (current gain is larger than the best gain up to now):
                                set best feature to be current feature
                                set best splitting point to be current split
                                set best gain as current gain
        return (the best feature and the best gain)
```

```
def grow(self, X, y, criterion = 'gini'):
“’A recursive function that grows a tree given input training set and a specific criterion
   X: training feature data
   Y: labels for training data
   criterion: gini or entropy.
“’
        Set the prediction label of this node to be the label with more samples
        If (the subset of data is pure or features are identical):
                Stop growing and mark it a leaf
        Else:
                Split the data based on the best feature and best split
                Initiate its left and right children
                Grow its children with the splitted data respectively.
```

```
def __list_all_trees(self):
“’A private method that returns a listing of all possible trees that can be formed by removing a
single node from a base tree
“’
    Initiate an empty list
    for (each node in breadth first search order):
        if (this node is not leaf):
            store the list of its children
            delete the children
            mark this node as a leaf node
            make a deep copy of the pruned tree
            append the deep copy tree to the list
            add back its children
            mark the node as non-leaf node
    return (the list of trees)

def pruneSingleGreedyNode(self, X_val, y_val):
“’An exhaustive search for the single node for which removing it (and its children) produces
   the largest increase (or smallest decrease) in classification accuracy as measured using
   validation data.
   X_val: features of the validation data
   y_val: labels of the validation data
“’
    Initiate the best tree as None
    Initiate the best accuracy as negative infinity
    List all possible pruned trees from the original tree itself
    for (each tree in the list):
        predict the labels given the validation features
        calculate the accuracy of predicted labels compared with the true label
        if (current accuracy is larger than the true accuracy):
            set best accuracy to be current accuracy
            set best tree to be current tree
    return (the best tree)
```