Introduction to Machine Learning HW1

- Task：Implement ID3_Tree for classifying Iris dataset
- Environment：OSX MAC、Ubuntu 16.04.3 LTS
- Language：Python 2.7.12
- Library：numpy
- Result：
  1. ID3_Tree, K-Fold-Validation K = 5, with data standard normalization
     50 times average accuracy, 3 class precision & recall

```
DiawChen /Users/DiawChen/NCTU/Senior/Intro to Machine Learning/HW1 /ID3_Tree
 python DecisionTree_Iris.py                                        18:47:45
0.944
1.000 0.991
0.920 0.913
0.919 0.930
```

  2. Random Forest, K-Fold-Validation K = 5, with data standard normalization
     50 times average accuracy, 3 class precision & recall

```
DiawChen /Users/DiawChen/NCTU/Senior/Intro to Machine Learning/HW1 /Random_Fo
rest  python Random_forest.py                                       18:53:03
0.952
1.000 0.995
0.928 0.929
0.931 0.929
```

- How does my code work：
  1. ID3_Decision_Tree (Training)

     DecisionTree_Iris.py is the main function, Node.py define the Tree node class, and util.py contains function like std_normalize, k_fold_dataset such utility function.

     First, data will be shuffled and standard normalized through std_normalize(), use k-fold-validation and then create the root of decision tree.

```
std_normalize(data)
shuffle(data)
training_set, testing_set = k_fold_dataset(data)
ID3_Tree = Node(training_set[i])
```

     Second, run the ID3_algorithm()

```
ID3_algorithm(ID3_Tree)
```

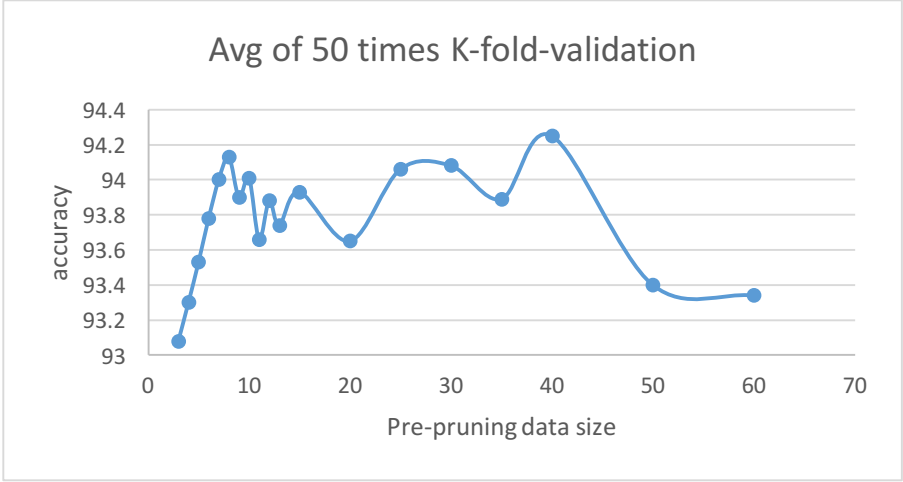
```
def ID3_algorithm(root):

    current_node = find_impure_leaf(root)

    while current_node != None:

        (threshold, feature_index) = find_threshold_spiltIdx(current_node.data)
        (left, right) = split(current_node.data, threshold, feature_index)

        current_node.set_threshold(threshold)
        current_node.set_threshold_idx(feature_index)

        left_node = Node(left)
        right_node = Node(right)
        current_node.left = left_node
        current_node.right = right_node
        current_node.leaf = False

        current_node = find_impure_leaf(root)
```

in the ID3_algorithm(), first find the leaf with impure data by find_impure_leaf() (not yet be labelled), and then find the best feature & threshold through find_threshold_splitIdx() which calculate the lowest entropy with different criteria(boundary value in sorted data within two continuous data) , also set the current node threshold/feature with it.

```
def find_threshold_spiltIdx(data):
    best_feather_index = -1
    best_entropy = float('inf')
    best_threshold = float('inf')
    print "haha"
    print len(data[0][:-1])

    for i in range(len(data[0][:-1])):

        (entropy, threshold) = cal_lowest_entropy(data, i)

        if entropy < best_entropy:
            best_feather_index = i
            best_entropy = entropy
            best_threshold = threshold

    return (best_threshold, best_feather_index)
```

then split the data into left and right by comparing with specific feature threshold calculated above. Last set current_node.leaf to False and try to find the next impure node which may need to be further split.

Also, I implement pre-pruning during training in Node.py. After experiment, I figure out that choosing data size 40 as pre-pruning criteria out perform other size, set the node.label with majority class within 40 data candidate.




```python
class Node:

    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data
        self.threshold_idx = -1
        self.threshold = -1
        self.leaf = True
        self.pure = True
        self.label = -1
        self.check_prune = False
        # determine whether this set data is pure or not
        if len(data) > 1:
            check_label = data[0][-1]

            # set majority class as pre-pruning
            for i in range(1,len(data)):
                    if check_label != data[i][-1]:
                        self.pure = False

            if len(data) < 40 and not(self.pure):
                #print "pre-pruning!!"
                #pprint.pprint(self.data)
                label_count = [0,0,0]
                for label in self.data:
                    if label[-1] == 'setosa':
                        label_count[0] += 1
                    elif label[-1] == 'versicolor':
                        label_count[1] += 1
                    elif label[-1] == 'virginica' :
                        label_count[2] += 1

                max_idx = max(enumerate(label_count),key=lambda x: x[1])[0]

                if max_idx == 0:
                    self.label = 'setosa'
                elif max_idx == 1:
                    self.label = 'versicolor'
                else:
                    self.label = 'virginica'
                self.pure = True
                self.check_prune = True


        # if all elements are same label, set the node.label
        if self.pure and not(self.check_prune):
            #print "here"
            #print data[0]
            self.label = data[0][-1]
```

ID3_Decision_Tree (Testing)

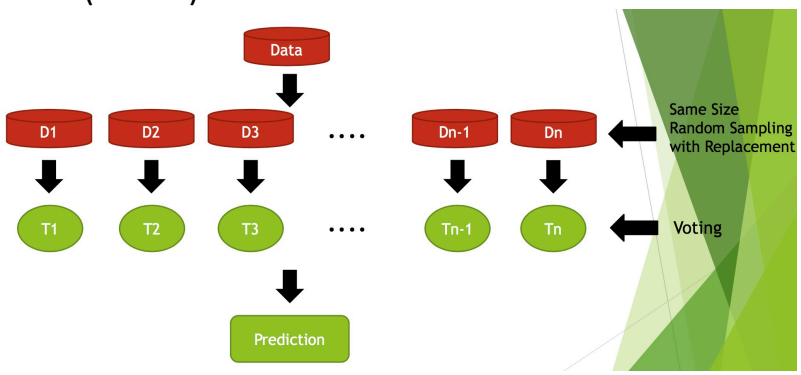
First predict the input data with Decision_Tree, and then calculate the accuracy, precision, recall according to TA's homework assignment ppt. (implement through calc_error())

```
def predict(datapoint, Tree):
    curr_node = Tree
    while not(curr_node.pure):
        threshold = curr_node.threshold
        feature_index = curr_node.threshold_idx
        if datapoint[feature_index] <= threshold:
            curr_node = curr_node.left
        else:
            curr_node = curr_node.right

    return curr_node.label
```

```
accuracy, precision, recall  = calc_error(testing_set[i],ID3_Tree)
```

Plus, all the accuracy, precision, recall are calculate through K-fold-validation (K=5, implement in k_fold_dataset() in util.py)

2. Random Forest (Bonus)



Implement detail:
Each tree training data size = 30
Number of decision tree = 15
Pre-pruning data size = 10

```
def Random_Forest(dataset):
    Forest = list()
    data = dataset
    # create a forest consists of 15 trees
    for i in range(15):
        training_set = data[:30]
        tree = Node(training_set)
        # each tree perform pre-pruning when node contain less than 10 data
        ID3_algorithm(tree)
        Forest.append(tree)
        shuffle(data)

    return Forest
```

Prediction function needs to be modified (add voting process)

```
def predict(datapoint, Forest):
    count = [0,0,0]
    prediction = None
    #print len(Forest)
    for tree in Forest:
        curr_node = tree
        while not(curr_node.pure):
            threshold = curr_node.threshold
            feature_index = curr_node.threshold_idx
            if datapoint[feature_index] <= threshold:
                curr_node = curr_node.left
            else:
                curr_node = curr_node.right
        if curr_node.label == 'setosa':
            count[0] += 1
        elif curr_node.label == 'versicolor':
            count[1] += 1
        elif curr_node.label == 'virginica':
            count[2] += 1

    max_idx = max(enumerate(count),key=lambda x: x[1])[0]

    if max_idx == 0:
        #print 'setosa'
        prediction = 'setosa'
    elif max_idx == 1:
        #print 'versicolor'
        prediction = 'versicolor'
    else:
        #print 'virginica'
        prediction = 'virginica'

    return prediction
```