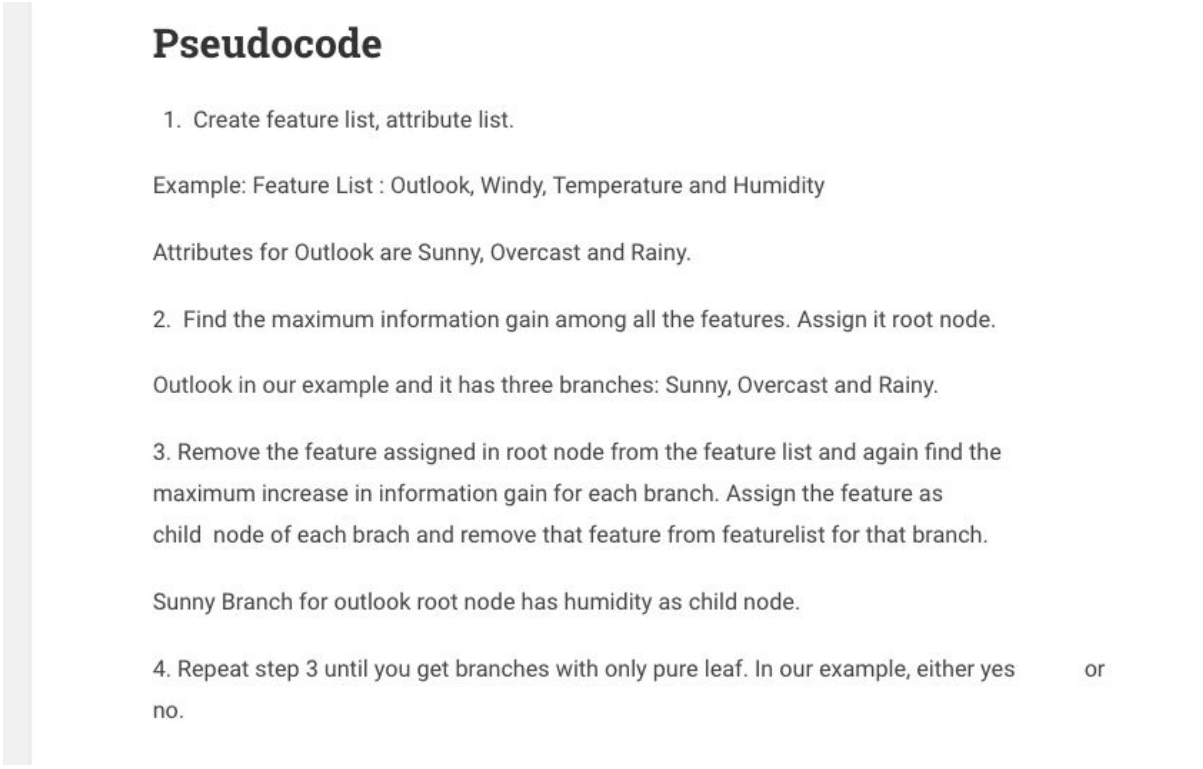
Project report

I have followed this turtorial to build my own decision tree:



Pseudocode

1. Create feature list, attribute list.

Example: Feature List : Outlook, Windy, Temperature and Humidity

Attributes for Outlook are Sunny, Overcast and Rainy.

2. Find the maximum information gain among all the features. Assign it root node.

Outlook in our example and it has three branches: Sunny, Overcast and Rainy.

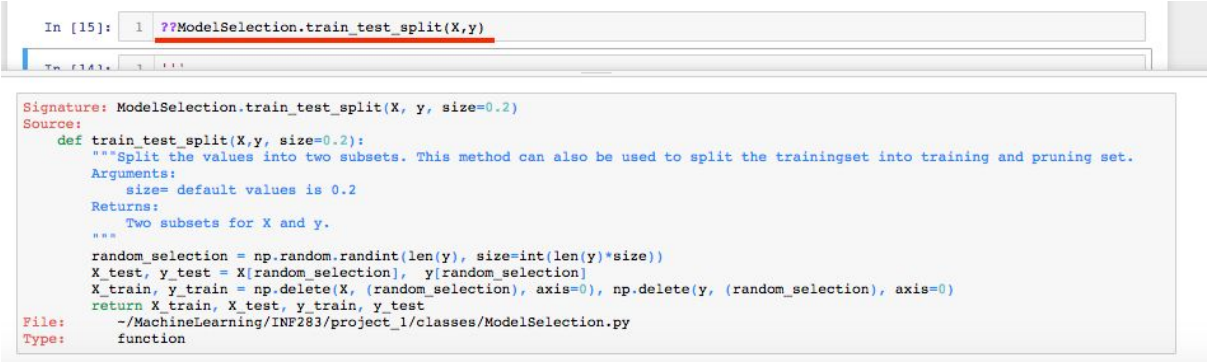
3. Remove the feature assigned in root node from the feature list and again find the maximum increase in information gain for each branch. Assign the feature as child node of each brach and remove that feature from featurelist for that branch.

Sunny Branch for outlook root node has humidity as child node.

4. Repeat step 3 until you get branches with only pure leaf. In our example, either yes or no.

https://nullpointerexception1.wordpress.com/2017/12/16/a-tutorial-to-understand-decision-tree-id3-learning-algorithm/

All the methods in Decision_tree are described and you can type *??* in front of methods outside this notebook to get a description of them.
Example:



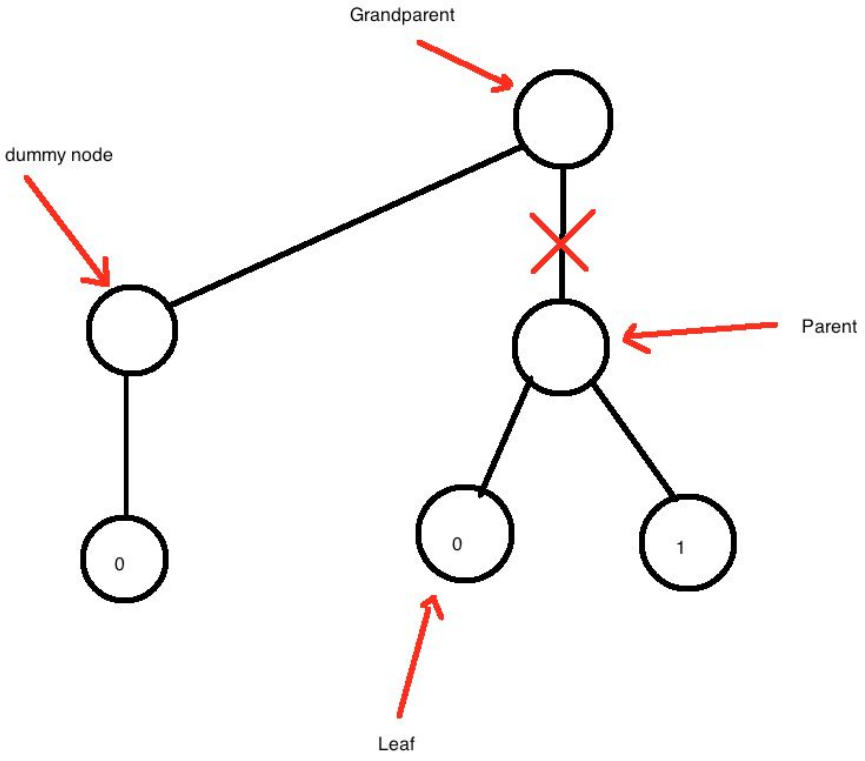
```
In [15]: 1 ??ModelSelection.train_test_split(X,y)
```

```
Signature: ModelSelection.train_test_split(X, y, size=0.2)
Source:
    def train_test_split(X,y, size=0.2):
        """Split the values into two subsets. This method can also be used to split the trainingset into training and pruning set.
        Arguments:
            size= default values is 0.2
        Returns:
            Two subsets for X and y.
        """
        random_selection = np.random.randint(len(y), size=int(len(y)*size))
        X_test, y_test = X[random_selection],  y[random_selection]
        X_train, y_train = np.delete(X, (random_selection), axis=0), np.delete(y, (random_selection), axis=0)
        return X_train, X_test, y_train, y_test
File:      ~/MachineLearning/INF283/project_1/classes/ModelSelection.py
Type:      function
```

I have built the node class based on the first answer from this stackoverflow link:
https://stackoverflow.com/questions/41760856/most-simple-tree-data-structure-in-python-that-can-be-easily-traversed-in-both

Description of the pruning process in the code.

I am using a recursive method to find the leaf of a subtree. Then I find the parent node of the leaf and check how many children it has. This defines how many values the dummy node needs to be tested on. After this I find the grandparent node of the leaf and change the references from the parent node to a dummy node.
Now you can predict on the “new tree” and see if it gives better results. If it improves the accuracy, then we keep it, otherwise we retain the “old tree”(with the parent)



Code for this:

1. In [21]:

```
'''
node: Node object. Example a child node of root.
return the leaf node for the node.
'''
def find_leaf(node):
    if node.isLeaf:
        return (node)
    else:
        for key, child in node.children.items():
            return find_leaf(child)
```

2. Find the variabel value of parent node in grandparent node, so we can change the reference to dummy node in the *prune* method

In [22]:

```
'''
find the variabel value in grandparent node, so we can change the variables node to another node. In our case paren
returns the variabel value.
'''
def find_parent_variabel(parent_node,grand_parent_node):
    for key, node in grand_parent_node.children.items():
        if node == parent_node: return key
```

3. Endrer til dummy node og predikerer på det “nye treet”. Hvis den gjør det bedre, så beholder vi endringen, så returnerer vi tilbake at en endring har blitt gjort, slik at man kjører en ny runde med pruning gjennom subtrees.

4. Changes to a dummy node and predicts on the “new tree”. If it’s better, then we will keep it and return a true boolean value that says a change has been made. This values is used in the *learn* method to check if we have to do more pruning on the tree.

In [23]:

```
'''
leaf_node: leaf_node of a subtree
pruning_accuracy: accurcy on the pruning set
X_pruning:
y_pruning
tree: the root node

The method checks if the accurcy on the pruning set increases if we change the parent node of the leaf_node to leaf
The parent node will be the new leaf if it imporves the accuracy and the children of the parent node will be remove
return variable says if there have been any changes in the three (True/False)
'''
def prune(leaf_node, pruning_accuracy, X_pruning, y_pruning, tree):
    prun_acc = pruning_accuracy
    changes = False
    parent = leaf_node.parent_node

    child_values_to_check = []
    child_values_to_check.append(bool(leaf_node.data))
    if(len(parent.children) >1):
        child_values_to_check.append(not leaf_node.data)

    for i in child_values_to_check:
        grand_parent = parent.parent_node

        dummy_node = Node()
        dummy_node.category = parent.category
        dummy_node.isLeaf = True
        dummy_node.data = i
        dummy_node.parent_node = grand_parent

        if not (grand_parent == None):
            parent_variable_in_grand_parent = find_parent_variabel(parent, grand_parent)
            grand_parent.children[parent_variable_in_grand_parent] = dummy_node

        #predict
        pred = predict(X_pruning, tree)
        pred_acc = Metrics.accuracy(y_pruning,pred)

        if (prun_acc < pred_acc):
            prun_acc = pred_acc
            parent = dummy_node
            changes = True
        elif not (grand_parent == None ):
            grand_parent.children[parent_variable_in_grand_parent] = parent
    return changes
```