TP Report on Decision Tree

October 16, 2018

1 Question I Build the tree

In [2]:
```
from sklearn import tree
from sklearn.externals.six import StringIO
import pydotplus
import numpy as np

#Load the data and training
sky_data = np.loadtxt('skysurvey/training_data.csv', delimiter=",")
sky_class = np.loadtxt('skysurvey/training_class.csv')
clf = tree.DecisionTreeClassifier(min_samples_leaf=0.01)
clf.fit(sky_data, sky_class)

#Draw the tree into a pdf
dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data, filled= True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("Tree.pdf")
```

Out[2]:
```
True
```

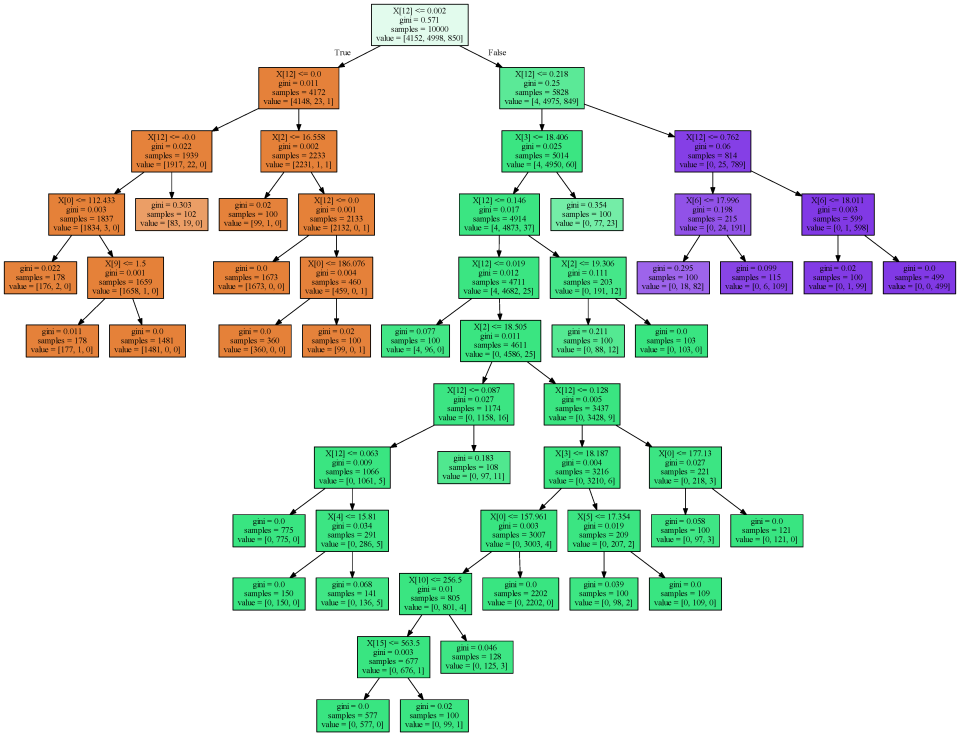
And the tree we get is as follows in imge Tree:

2 Question II The generalization error

From the definition : > Gener. error = training error + 0.5 x #of leaves

We can write the code of calculate the generalization error like this:

In [3]:
```
#Calculate the generalization error
training_error = (len(sky_class) - clf.score(sky_data,sky_class)*len(sky_class))/len(sk
array_children = clf.tree_.children_left
num_leaves = np.count_nonzero(array_children == -1)
gen_error = training_error + (0.5*num_leaves/len(sky_class))
print("Training error: "+str(100*training_error)+"%")
print("Number of nodes in the tree: " +str(clf.tree_.node_count))
print('Generalization error: ' + str(100*gen_error) + '%')
```



Tree

```
Training error: 1.13%
Number of nodes in the tree: 55
Generalization error: 1.27%
```

3 Question III Change the parameter

As we have seen in the results of the two precedent questions, the training error we got is 1.13%, which is near to the *minimun sample leafs* proportion we have chosen. If we observe the tree obtained in question 1, we can see that after the second layer, because the exceptions in each branch are less than 1% of the total instances and are distributed, which means they are not sufficient to become a leaf, so that they are not successfully extracted.

So, we cannot reduce the training error by pre-pruning the data with the parameters of this classifer, what we can do in order to minimize the number of leaves is to cut as many leaves as possible without changing the classification result. If we observe the tree generated in the first question, we can see that we can cut all the nodes in the left branch up to the first layer, and the right branch up to the second layer.

For achieving this, we can simply set ***max_leaf_node = 3***, this will minimize the leafs cause we need at least three nodes for three possible class and in the tree of question 1, no new significant division is generated from the first three nodes.

the code, the result is shown below:

```
In [38]: #Load the data and training
```

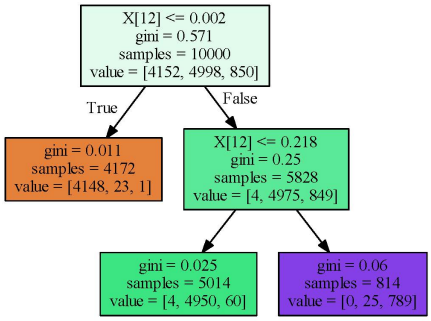
```
clf = tree.DecisionTreeClassifier(min_samples_leaf=0.01, max_leaf_nodes = 3)
clf.fit(sky_data, sky_class)

#Draw the tree into a pdf
dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data, filled=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("Tree_modified.pdf")

#Calculate the generalization error
training_error = (len(sky_class) - clf.score(sky_data,sky_class)*len(sky_class))/len(sky_
array_children = clf.tree_.children_left
num_leaves = np.count_nonzero(array_children == -1)
gen_error = training_error + (0.5*num_leaves/len(sky_class))
print('Training error: ' + str(100*training_error) + '%')
print('Number of leaves: ' + str(num_leaves))
print('Generalization error: ' + str(100*gen_error) + '%')
```

```
Training error: 1.13%
Number of leaves: 3
Generalization error: 1.145%
```

We can see that the training error has not changed even if we has set the number of leaves to 3, the minimum value possible for this case. Yet another way to has a tree like this is to set ***min_impurity_decrease=t***, where ***t*** is a value big enough to prevent the second spilt in the left main branch but no too big to cease the spilt in the right branch. Amd we can also set *t* in a value that can exactly give us the three leaves tree we need, this is possible because except the three splits , all other spilt will just decrease the impurity a very little bit, any value between ***0.01(1%, the minimum sample leafs)*** and ***0.113(1.13%, the training error)*** will do (**En effet, the real range will be wider than this, but the calculation of the exact range is pointless**).

Or, more simply, we can add *max_depth*=2, and use a very samll value of *t*, because the split in the leaf has just decreased the impurity for a negligible amount. For instance, ***max_depth=2, min_impurity_decrease=0.00003*** will do.

Below is the tree obtained in these three ways.



Tree Modified

4 Question IV Comparison

The answer is direct, I will choose **the best one I obtained in point 2**, If the final classification result is the same, we do not need to perform the following classification procedures, the latter tree will deliver the same result, but just in two comparisons in maximum.

5 Question V Insights

```
In [39]: three_object = np.array([[199, 0.2, 19, 18, 18, 18, 16, 777, 301, 3.0, 270, 3.312, 0.0
                                  [199, 0.2, 19, 18, 18, 18, 16, 777, 301, 3.0, 270, 3.312, 0.
                                  [199, 0.2, 19, 18, 18, 18, 16, 777, 301, 3.0, 270, 3.312, 0.

Class_value = clf.predict(three_object)
print(Class_value)

[0. 1. 2.]
```

For this test, I have constructed three objects which carry the exact same data except the differences in **X[12]**, which is the **"Final Redshift"** of the sky object, but we have seen that the three object are successsfully classified.

The **insights** for this tree need to be discussed in two aspects. **On one hand**, if a 1.145% error rate is acceptable, we can conclude that only the redshift feature is enough to distinguish stars, galaxies and Quasars so that we can use it as a criterion, or at least we can say that the redshift is the main difference between these three categories. **On the other hand**, a 1.145% error rate is still a little high, and this tree is even kind of robust because all the other features haven't been used, we can still get the result from the redshift no matter how ridiculous the other features are. But we have to admit that by *pruning* like this, we do have minimized the generalization error.

Consequently, it will be kind of a dilemma if we can neither accept an error over 1% nor simply take redshift as a criterion. However in fact an error that is just a little larger than 1% is quite normal with the minimum sample numbers for a leaf being 1% of total instances, this tree don't have problems like *underfitting* or *overfitting* to this extent.

6 Question VI Post-pruning or not ?

The answer is no. Now I only have three leaves and three classes to classify in the meantime, thus it is not possible to perform further pruning. Any further pruning will cause a whole class of instances to be misplaced, which will greatly increase the training error.

7 Question VII The implement of post-pruning

I have implemented the post-pruning policy like this. But on top of what we have discussed in the precedent questions, I choose to **add an Exception** to throw when the tree got in can no longer be pruned. When the tree can be pruned, in this case, I have firstly caculated the misplaced instances in a node, if the number of misplaced instances in this node is smaller than the minimum requirement for a leaf, we can cut it because it will either not increase the training error, or increase the error by a amount less than 0.5 x #leaves_cutted, which will in turn **reduce the generalization error**.

```
In [55]: def post_pruning(tree):

    min_leaf = 1
    tempm = tree.min_samples_leaf
    if tempm != None:
        if tempm < 1:
            min_leaf = tempm * clf.tree_.n_node_samples[0]
        if tempm > 1:
            min_leaf = tempm
    array_children = tree.tree_.children_left
    num_leaves = np.count_nonzero(array_children == -1)
    n = tree.n_classes_
    if n >= num_leaves:
        raise Exception('Tree can no longer be pruned.')
    else:
        for i in range(tree.tree_.node_count):
            values = tree.tree_.value[i,0]
            instance_misplaced = sum(values)-max(values)
            if instance_misplaced < min_leaf:
                tree.tree_.children_left[i] = -1
                tree.tree_.children_right[i] = -1

post_pruning(clf)

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data, filled= True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("Tree_pruned.pdf")
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)

<ipython-input-55-576f7ac0de04> in <module>()
     20                 tree.tree_.children_right[i] = -1
     21
---> 22 post_pruning(clf)


<ipython-input-55-576f7ac0de04> in post_pruning(tree)
     11     n = tree.n_classes_
     12     if n >= num_leaves:
---> 13         raise Exception('Tree can no longer be pruned.')
     14     else:
     15         for i in range(tree.tree_.node_count):


Exception: Tree can no longer be pruned.
```

From the result above we can see that, this funtion returned the exception as expected for the **best tree in point 2**, But if we test this function with the tree we got from the default configuration, alias **the tree in point 1**, we can get a tree same as the best tree in point 2, which means this function works.