MSIS 5223: Tutorial 8 – Classification in Python

Instructions

This reading covers a classification using decision trees. Decision trees are very good for exploratory purposes in probing your data.

1. Regression Trees

The first tree this exercise looks at is a regression tree because the dependent and independent variables are continuous. The variables in the data include *Pollution*, *Temp*, *Industry*, *Population*, *Wind*, *Rain*, and *Wet.days*. *Pollution*, the dependent variable, is the SO_2 concentration.

The tree function in Python comes from the library sklearn, or scikitlearn (for more information on this module, see http://scikit-learn.org). This is a class object you pass data into. Scikitlearn provides a classification tree and a regression tree. The tree function uses the Gini Index as default, but you can switch to entropy. Unlike the `tree` function in R which assumes the first column of the dataframe is the target variable, the target must be specified in Python; thus, there is no need to ensure the first column of you dataframe is the target variable. Additionally, the columns used to classify onto the target must be specified.

The classification tree and regression tree within the module sklearn provides many options. Some of these options are useful for pruning the tree. Some of the more important ones are listed below:

- `max_depth`: an integer representing the number of "levels" for nodes
- `max_leaf_nodes`: the best nodes are selecting when growing the tree until the number is reached; when used, `max_depth` is ignored
- min_samples_split: the minimum number of samples each node requires for a split to occur
- `min_samples_leaf`: the minimum number of samples required at a leaf node

Using the pollution data, the following code is run to create the regression tree. Unfortunately, the output is too large to display within this document. The output

contains many nodes and leaves and is not very useful. This tree and the one in Tutorial 07 are vastly different. After running this yourself, look at the first split for *Industry*. The split value is at 748, just like that in R. The right side of the split, that with *False*, reduces the sample size per node past 5. For example, after *Population*, *Rain* has a sample of only 4 while the next split of *Population* has 3.

```
col_names = list(pollute_data.ix[:,1:7].columns.values)

tre1 = tree.DecisionTreeRegressor().fit(pollute_data.ix[:,1:7],pollute_data.Pollution)

dot_data = StringIO()
tree.export_graphviz(tre1, out_file=dot_data,
                     feature_names=col_names,
                     filled=True,
                     rounded=True,
                     special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Figure 1 Code for the Tree Function

Try using the criteria min_samples_split and min_samples_leaf and set both to 5. The results are much improved and are closer to that in R. The results of the tree can be plotted for a visual representation, as seen in the figure to the side. The output reveals *Industry* is the first split and represents the most important variable for splitting. A difference exists between this tree and that in R. Within R, after the split of *Wet.days*, the variable *Temp* is next. The tree from Python has *Rain* instead of *Temp*.

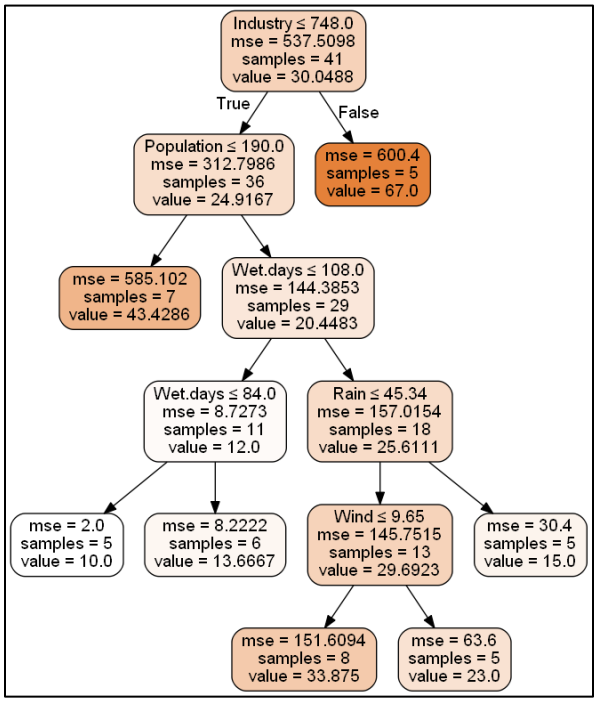

Figure 2 Visualization of Tree

2. Classification Trees

The data for the classification tree contains taxon data for various plant flora and fauna. The data is presented below in Figure 7 and should be familiar to you. The target variable is categorical, so we are dealing with a classification tree.

```
In [26]: taxon_data.head()
Out[26]:
  Taxon    Petals  Internode     Sepal      Bract    Petiole      Leaf
0     I  5.621498  29.480596  2.462107  18.203409  11.279097  1.128033
1     I  4.994617  28.360247  2.429321  17.652049  11.040838  1.197617
2     I  4.767505  27.254318  2.570497  19.408385  10.490722  1.003808
3     I  6.299446  25.924238  2.066051  18.379155  11.801823  1.614052
4     I  6.489375  25.211308  2.901583  17.313047  10.121590  1.813333

      Fruit
0  7.876151
1  7.025416
2  7.817479
3  7.672492
4  7.758443
```

Figure 3 Taxon Data

The resultant tree is plotted below. It may help to have the data in front of you to compare. The first class, Class IV, is distinguished by the size of the *Sepal*; that is, a size greater than 3.53. With a size less than or equal to 3.53 and a leaf size greater than 2, Class III plants are distinguished.

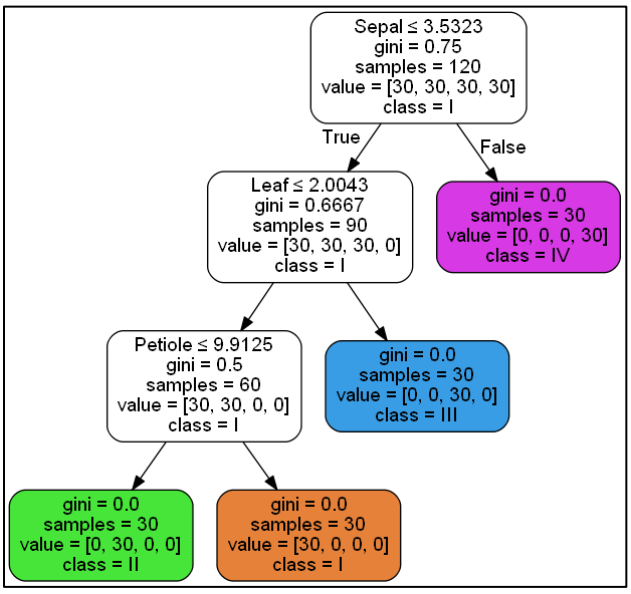

Figure 4 Tree Structure of Taxon Data

In addition to interpreting a decision tree, it is important to actually assess the quality of the tree. This can be done by viewing a report of the classification as well as a

confusion matrix. The figure below presents the code and the output for these two assessments.

```
In [28]: predicted = tre2.predict(taxon_data.ix[:,1:8])
   ....: print(metrics.classification_report(taxon_data.Taxon, predicted))
   ....: print(metrics.confusion_matrix(taxon_data.Taxon, predicted))
             precision    recall  f1-score   support

          I       1.00      1.00      1.00        30
         II       1.00      1.00      1.00        30
        III       1.00      1.00      1.00        30
         IV       1.00      1.00      1.00        30

avg / total       1.00      1.00      1.00       120

[[30  0  0  0]
 [ 0 30  0  0]
 [ 0  0 30  0]
 [ 0  0  0 30]]
```

Figure 5 Assessment for Classification of Taxon

The precision of classifying each group is 100% accurate. Additionally, the confusion matrix reveals that to be the case. The columns represent the actual values, the rows the predicted values. Below is a graphical representation of the confusion matrix. This is a gradient-based confusion matrix. This is not easily discerned because the accuracy is 100%. Tutorial 6 provides a matrix that better provides an understanding of how the gradients operate.

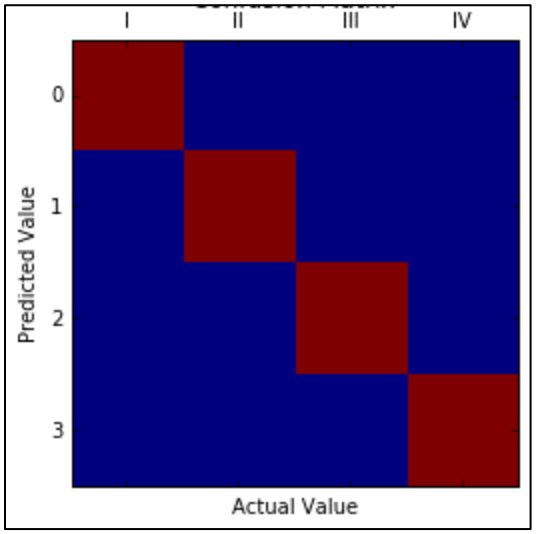

Figure 6 Confusion Matrix