DecisionTree

April 16, 2019

Gahan Saraiya (18MCEC10)

AIM: Implementation of Decision Tree and Naive Bayes

Decision Tree

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

Balance Scale data set consists of 5 attributes, 4 as feature attributes and 1 as the target attribute. We will try to build a classifier for predicting the Class attribute. The index of target attribute is 1st.

1. 3 (L, B, R)
2. Left-Weight: 5 (1, 2, 3, 4, 5)
3. Left-Distance: 5 (1, 2, 3, 4, 5)
4. Right-Weight: 5 (1, 2, 3, 4, 5)
5. Right-Distance: 5 (1, 2, 3, 4, 5)

```
In [1]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import accuracy_score
        from sklearn import tree
        import os
```

```
In [2]: balance_data = pd.read_csv(os.path.join('input', 'balance-scale.data'), sep= ',', head
```

```
In [3]: print("Dataset:: ")
        balance_data.head()
```

```
Dataset::
```

Out[3]:

	0	1	2	3	4
0	B	1	1	1	1
1	R	1	1	1	2

```
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

In [4]:
```
X = balance_data.values[:, 1:5]
Y = balance_data.values[:,0]
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, random_sta
```

0.0.1 Decision Tree Classifier with criterion gini index

In [6]:
```
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=3,
clf_gini.fit(X_train, y_train)
```

Out[6]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=100,
                       splitter='best')
```

In [7]:
```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,
 max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)
```

Out[7]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=100,
                       splitter='best')
```

1 Prediction

In [8]:
```
clf_gini.predict([[4, 4, 3, 3]])
```

Out[8]:
```
array(['R'], dtype=object)
```

In [9]:
```
y_pred = clf_gini.predict(X_test)
y_pred
```

Out[9]:
```
array(['R', 'L', 'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'R', 'L', 'L',
       'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L',
       'L', 'L', 'R', 'L', 'L', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'L',
       'L', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'R',
       'L', 'R', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'R', 'L', 'L', 'L',
       'L', 'L', 'R', 'R', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R', 'R',
       'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R',
       'R', 'L', 'L', 'L', 'R', 'R', 'L', 'L', 'L', 'R', 'L', 'R', 'R',
```

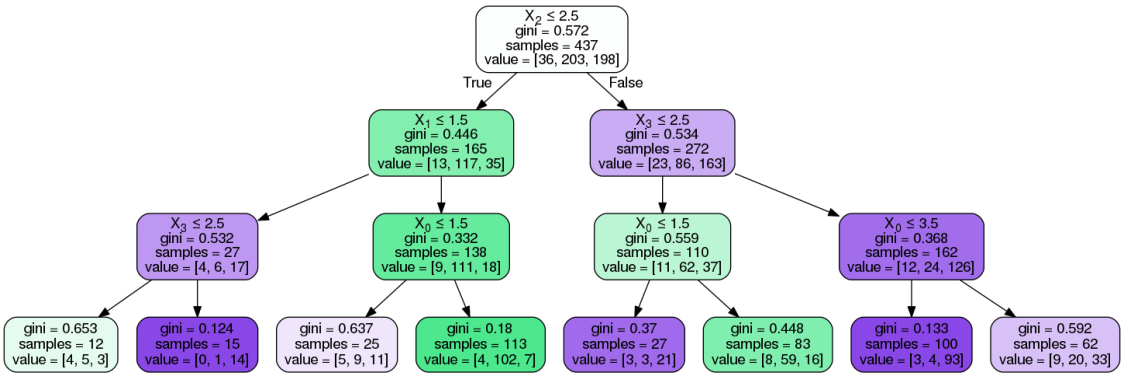
```
       'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R',
       'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'R',
       'R', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'L',
       'R', 'L', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'R', 'R', 'R',
       'L', 'R', 'R', 'R', 'R', 'R', 'L', 'L', 'R', 'R', 'R', 'R', 'L',
       'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R',
       'R', 'L', 'L', 'R', 'R', 'R'], dtype=object)
```

In [10]:
```
y_pred_en = clf_entropy.predict(X_test)
y_pred_en
```

Out[10]:
```
array(['R', 'L', 'R', 'L', 'R', 'L', 'R', 'L', 'R', 'R', 'R', 'R', 'L',
       'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L',
       'R', 'L', 'R', 'L', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'R',
       'L', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'L', 'R',
       'L', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R',
       'L', 'L', 'R', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R', 'R',
       'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R',
       'R', 'L', 'L', 'L', 'R', 'R', 'L', 'L', 'L', 'R', 'L', 'L', 'R',
       'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R',
       'L', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'L', 'L', 'L', 'R',
       'R', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'L',
       'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'R', 'R', 'R', 'R',
       'L', 'R', 'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L',
       'R', 'L', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'R', 'R', 'R',
       'L', 'L', 'L', 'R', 'R', 'R'], dtype=object)
```

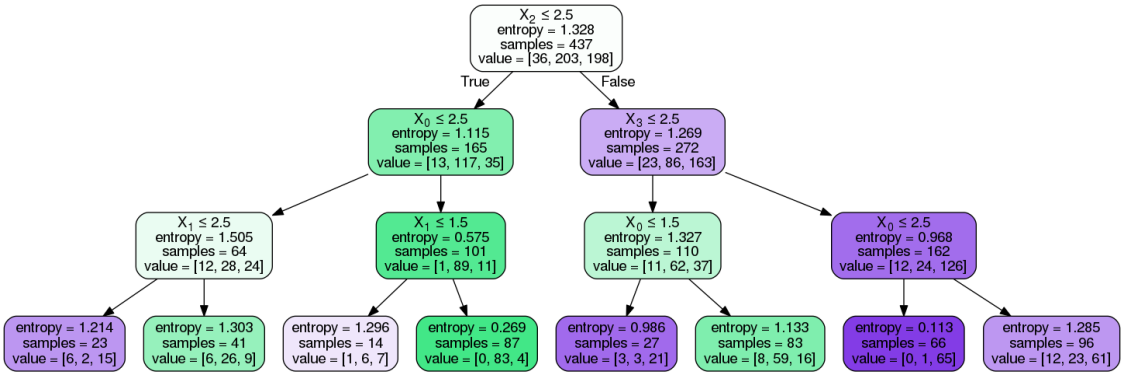
In [11]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf_gini, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[11]:



```
In [12]: dot_data = StringIO()
         export_graphviz(clf_entropy, out_file=dot_data,
                         filled=True, rounded=True,
                         special_characters=True)
         graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         Image(graph.create_png())
```

Out[12]:



1.1 Navie bayes

Naive Bayes is a classification algorithm for binary and multi-class classification problems.

Naive Bayes can be extended to real-valued (continuous value) attributes, by assuming a Gaussian distribution.

This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because it only need to estimate the mean and the standard deviation from the training data.

```
In [13]: from sklearn import datasets
         from sklearn import metrics
         from sklearn.naive_bayes import GaussianNB

         dataset = datasets.load_iris()
         model = GaussianNB()
         model.fit(dataset.data, dataset.target)
         expected = dataset.target
         predicted = model.predict(dataset.data)
         print(metrics.classification_report(expected, predicted))
         print(metrics.confusion_matrix(expected, predicted))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.94      0.94      0.94        50
           2       0.94      0.94      0.94        50

   micro avg       0.96      0.96      0.96       150
   macro avg       0.96      0.96      0.96       150
weighted avg       0.96      0.96      0.96       150

[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

Naive Bayes Classifier

dummy dataset with three columns: weather, temperature, and play. The first two are features(weather, temperature) and the other is the label.

In [ ]: