WEEK- 7: DecisionTreeClassifier.

Session No.5

sklearn.tree.DecisionTreeClassifier

```
DecisionTreeClassifier(*, criterion='gini',
                       splitter='best', max_depth
                       = None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.0,
                       max_features=None,
                       random_state=None,
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.0,
                       class_weight=None, ccp_alpha=0.0)
```

Important Parameters:

criterion{“gini”, “entropy”, “log_loss”}, default=”gini”

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurityand “log_loss” and “entropy” both for the Shannon information gain.

splitter{“best”, “random”}, default=”best”

The strategy used to choose the split at each node. Supported strategies are “best” to choose thebest split and “random” to choose the best random split.

max_depth:int, default=None

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure oruntil all leaves contain less than min_samples_split samples.

min_samples_split:int or float, default=2

The minimum number of samples required to split an internal node:

min_samples_leaf:int or float, default=1

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left andright branches. This may have the effect of smoothing the model, especially in regression.

min_weight_fraction_leaf:float, default=0.0
The minimum weighted fraction of the sum total of weights (of all the input samples) requiredto be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features:int, float or {“auto”, “sqrt”, “log2”}, default=None The number of features to consider when looking for the best split: If “auto”, then max_features=sqrt(n_features).

If “sqrt”, then max_features=sqrt(n_features). If “log2”, then max_features=log2(n_features).If None, then max_features=n_features.

random_state:int, RandomState instance or None, default=None
Controls the randomness of the estimator. The features are always randomly permuted at eachsplit, even if splitter is set to "best".

max_leaf_nodes:int, default=None
Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relativereduction in impurity. If None then unlimited number of leaf nodes.

ccp_alpha: non-negative float, default=0.0
Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largestcost complexity that is smaller than ccp_alpha will be chosen. By default, no pruning is performed.

Build decision tree-based model in python for Iris dataset from sci-kit learn

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
%pylab inline
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
#Load dataset
```

In [3]:
```
iris=datasets.load_iris()
```

Out[32]:
```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5 1  3 8  1 5  0 3]
```

In [4]:
```
df=pd.DataFrame(data=np.c_[iris['data'],iris['target']],
                columns=iris['feature_names']+['target'])
```

In [32]:
```
iris
```

In [5]:
```
df.head()
```

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

In [6]:
```
df['target'].value_counts()
```

Out[6]:
```
0.0    50
1.0    50
2.0    50
Name: target, dtype:
```

In [34]:
```
int64 df.shape
```

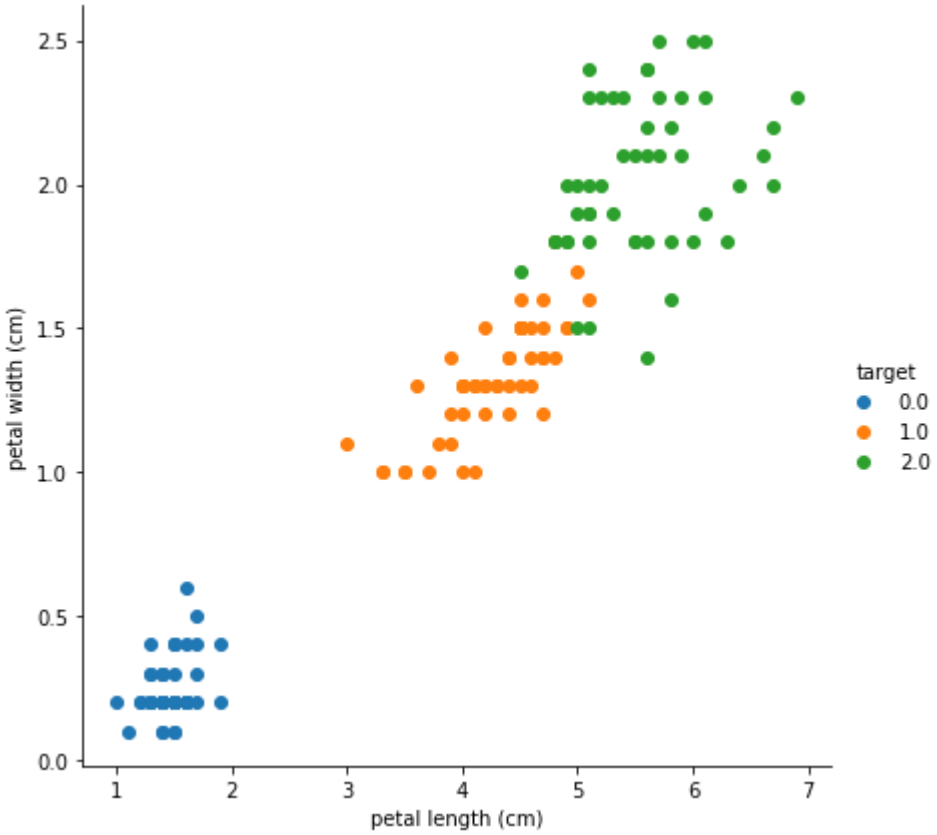
Out[34]:
```
(150, 5)
```

In [35]:
```
df.isnull().sum()
```

Out[35]:
```
sepal length (cm)    0 sepal
width (cm)     0 petal
length (cm)    0 petal width
(cm)     0 target
0 dtype: int64
```

In [7]:
```
sns.FacetGrid(df, hue='target',size=6).map(plt.scatter,
                "petal length (cm)","petal width (cm)").add_legend()
```

Out[7]:
```
<seaborn.axisgrid.FacetGrid at 0x1b84984c070>
```



In [8]:
```
#appling iris dataset
```

In [9]:
```
#fit a CART model to the data
```

In [10]:
```
model=DecisionTreeClassifier()
```

In [11]:
```
#model DecisionTreeClassifier='entropy',max_depth=3
```

In [12]:
```
model.fit(iris.data,iris.target)
```

Out[12]:
```
DecisionTreeClassifier()
```

In [13]:
```
model.score(iris.data,iris.target)
```

Out[13]:
```
1.0
```

In [14]:
```
#make prediction
```

In [15]:
```
expected=iris.target
```

In [22]:
```
predicted=model.predict(iris.data)
```

In [23]:
```
predicted
```

Out[23]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [28]:
```
pred=model.predict(iris.data)
pred
```

Out[28]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [26]:
```
#campare actual value and predict value
dff=pd.DataFrame({'ACTUAL':iris.target,'PREDICT':pred})
```

In [37]:
```
dff.tail(56)
```

Out[37]:

	ACTUAL	PREDICT

94	1	1
95	1	1
96	1	1
97	1	1
98	1	1
99	1	1
100	2	2
101	2	2
102	2	2
103	2	2
104	2	2
105	2	2
106	2	2
107	2	2
108	2	2
109	2	2
110	2	2
111	2	2
112	2	2
113	2	2
114	2	2
115	2	2
116	2	2
117	2	2
118	2	2
119	2	2
120	2	2
121	2	2
122	2	2
123	2	2
124	2	2
125	2	2
126	2	2

	ACTUAL	PREDICT
127	2	2
128	2	2
129	2	2
130	2	2
131	2	2
132	2	2
133	2	2
134	2	2
135	2	2
136	2	2
137	2	2
138	2	2
139	2	2
140	2	2
141	2	2
142	2	2
143	2	2
144	2	2
145	2	2
146	2	2
147	2	2
148	2	2
149	2	2

In [18]:
```
#summuaize the fit of the model
```

In [19]:
```
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 1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50

    accuracy                           1.00       150
macro avg        1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

```
[[50  0  0] [
 0 50  0]
 [ 0  0 50]]
```

In [ ]:

In [ ]: