01. 기본- 결정트리(decision tree)

- Machine Learning with sklearn @ DJ,Lim
- date : 20/10

(가) decision tree는 classification(분류)와 regression(회귀) 문제에 널리 사용하는 모델이다.
(나) 스무고개 놀이의 질문과 비슷하다.

In [1]:

```
from IPython.display import display, Image
```

In [2]:

```
import matplotlib.pyplot as plt
```

In [3]:

```
import mglearn
```

의사결정 트리 설명

(가) 세개의 feature(속성, 변수)가 있다.
'Has feathers?'(날개가 있나요?)
'Can fly?'(날수 있나요?)
'Has fins?'(지느러미가 있나요?)

(나) 우리가 분류하고자 하는 문제는 네 개의 클래스로 구분하는 모델을 만든다.
네 개의 클래스(매, 펭권, 돌고래, 곰)

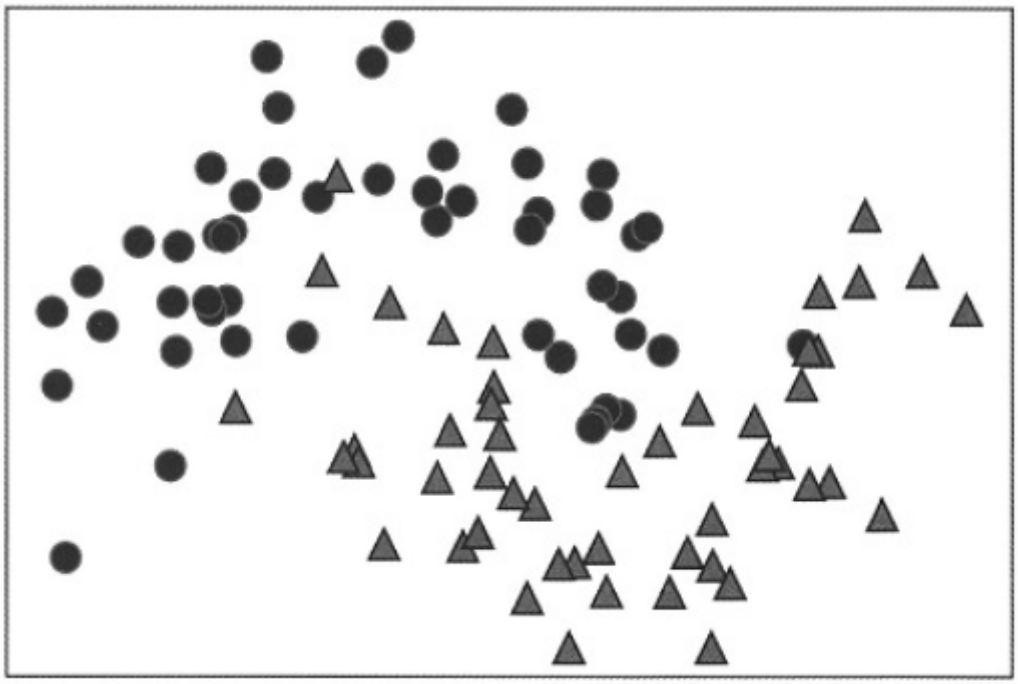
(다) 맨 위의 노드는 Root Node(루트 노드)라 한다.
(라) 맨 마지막 노드를 우리는 Leaf Node(리프노드)라 불린다.
(마) 범주형은 데이터를 구분하는 질문을 통해 데이터를 나누고, 연속형 데이터는 특성 i가 값 a보다 큰가? 라는 질문을 통해 나눈다.
(바) target이 하나로만 이루어진 리프 노드를 순수 노드(pure node)고 한다.

의사결정트리- 모델 만들기

(1) 데이터 셋 - 연속형 데이터
(2) 첫번째 나누기 x[1] = 0.0596
(3) 두번째 나누기 x[0] <= 0.4177, x[0] <= 1.1957

In [4]:

```
display(Image(filename='img/decisiontree01.png'))
```



In [5]:

```
display(Image(filename='img/decisiontree02.png'))
```

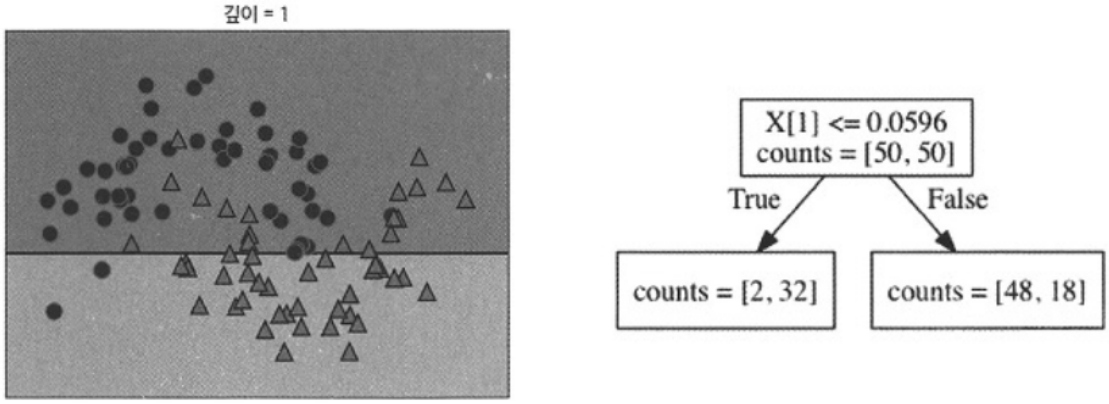


그림 2-24 깊이 1인 결정 트리(오른쪽)가 만든 결정 경계(왼쪽)

- 노드1 : class 0에 속한 데이터 수 2개, class 1에 속한 데이터 수 32개
- 노드2 : class 0에 속한 데이터 수 48개, class 1에 속한 데이터 수 18개

In [6]:

```
display(Image(filename='img/decisiontree03.png'))
```

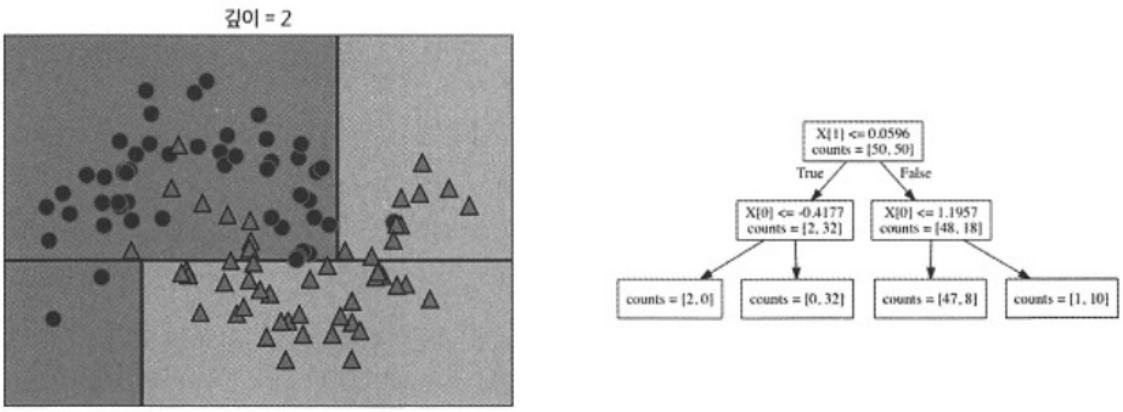


그림 2-25 깊이 2인 결정 트리(오른쪽)가 만든 결정 경계(왼쪽)

(참조 : 파이썬을 활용한 머신러닝 그림 참조)

설명

(가) 의사결정트리는 반복된 프로세스에 의해 노드가 두 개를 가진 이진의사결정트리를 만든다. (하나의 축을 따라 데이터를 둘로 나눈다.)
(나) 각 노드의 테스트(각 노드의 질문)는 하나의 특성(feature)에 관해서만 이루어진다.
(다) 데이터를 분할하는 것은 결정트리 리프(leaf)가 노드가 하나의 target(목표 값)을 가질 때까지 반복.
(라) Target(목표 값) 하나로만 이루어진 Leaf node(리프 노드)를 순수노드(pure node)라고 한다.

In [7]:

```
display(Image(filename='img/decisiontree04.png'))
```

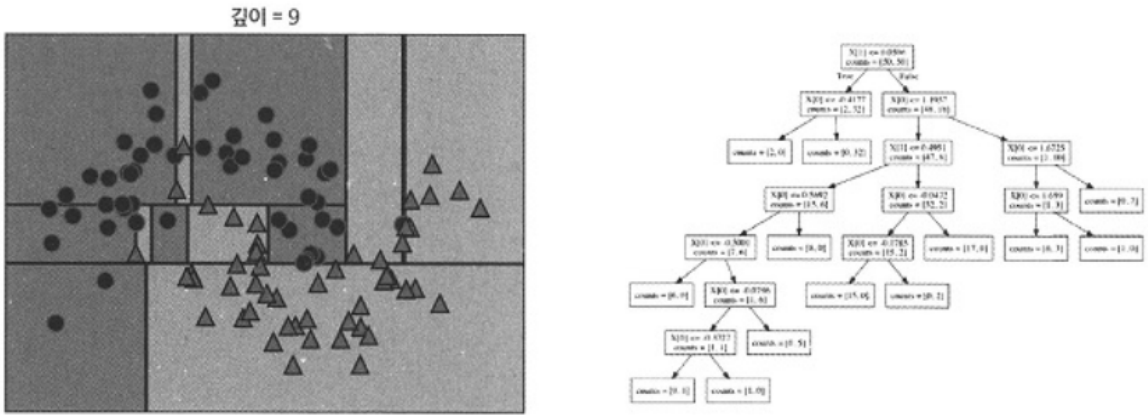


그림 2-26 깊이 9인 결정 트리의 일부(오른쪽)와 이 트리가 만든 결정 경계(왼쪽) (전체 트리는 너무 커서 일부만 표시했습니다.[37])

새로운 데이터(test) 셋에 대한 예측은 주어진 데이터 포인트가 분할한 영역 중에 어디에 놓이는 가를 확인하면 된다.

회귀 문제에서의 의사결정트리(decision tree)

(1) 각 노드의 테스트 결과에 따라 트리를 탐색(루트노드->리프노드)해 나가고 새로운 데이터 포인트에 해당되는 리프 노드(leaf node)를 찾는다.
(2) 찾은 리프 노드(leaf node)의 훈련 데이터 평균값이 이 데이터 포인트의 출력이 된다.
  * 리프노드가 8,9의 값을 갖고 있다면 출력은 8.5가 된다.

의사결정 트리의 단점(복잡하다) - Overfitting

(가) 리프 노드가 순수 노드가 될때까지 진행하면
모델이 매우 복잡해지고 훈련 데이터의 과대적합(overfitting)이 된다.
  -> 순수 노드로 이루어진 트리는 훈련 세트에 100% 정확하게 맞는다.

Overfitting(과적합)을 막는 두가지 전략

(1) 트리 생성을 일찍 중단하는 전략(pre-pruning) - 사전 가지치기
(2) 트리를 만든 후, 데이터 포인트가 적은 노드를 삭제(사후 가지치기-post-pruning) 하거나 병합하는 전략.(가지치기)-pruning

그렇다면 어떻게 사전 가지치를 할 수 있을까?

트리의 최대 깊이 제한 (max_depth)

리프의 최대 개수 제한 (max_leaf_nodes)

노드 분할을 위한 포인트의 최소 개수 지정 (min_sample_leaf)

사전 가지치기만 지원, *DecisionTreeRegressor, DecisionTreeClassifier*

실습1

- (1) 의사결정트리를 이용하여 데이터 셋을 나누고, (train_test_split)
- (2) cancer.target를 예측하는 모델을 만들어,
  - 훈련 세트 정확도와 테스트 세트 정확도를 만들어보자. --> 모델의 테스트 데이터 사이즈, 평가용 데이터 사이즈를 지정할 수 있도록 함수로 만들어보자.

In [8]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import seaborn as sns
```

데이터를 불러와 입력과 출력으로 분리

In [9]:

```
cancer = load_breast_cancer()
all_X = cancer.data
all_Y = cancer.target
```

데이터를 나눈다. 비율을 지정이 가능함.

In [11]:

```
def testTreeModel(TestSize=0.3):
    cancer = load_breast_cancer()
    all_X = cancer.data
    all_Y = cancer.target
    X_train, X_test, y_train, y_test = train_test_split(all_X,
                                                        all_Y,
                                                        stratify=cancer.target,
                                                        test_size = TestSize,
                                                        random_state=77)

    tree = DecisionTreeClassifier(random_state=0)
    tree.fit(X_train, y_train)
    print("학습용 세트 정확도 : {:.3f}".format(tree.score(X_train, y_train)))
    print("테스트 세트 정확도 : {:.3f}".format(tree.score(X_test, y_test)))
```

- 모든 리프 노드가 **순수 노드**이므로 훈련 세트의 정확도는 100%이다.

In [12]:

```
testTreeModel(0.3)  # 테스트 사이즈 30%
testTreeModel(0.1)  # 테스트 사이즈 10%
testTreeModel(0.2)  # 테스트 사이즈 20%
```

```
학습용 세트 정확도 : 1.000
테스트 세트 정확도 : 0.918
학습용 세트 정확도 : 1.000
테스트 세트 정확도 : 0.912
학습용 세트 정확도 : 1.000
테스트 세트 정확도 : 0.912
```

실습2

위의 함수에 max_depth=4로 모델에 추가해보자.

In [13]:

```
def testTreeModel(TestSize=0.3, treedepth=3):
    cancer = load_breast_cancer()
    X_train, X_test, y_train, y_test = train_test_split(all_X,
                                                        all_Y,
                                                        stratify=cancer.target,
                                                        test_size = TestSize,
                                                        random_state=77)
    tree = DecisionTreeClassifier(max_depth=treedepth, random_state=0)
    tree.fit(X_train, y_train)
    print("훈련 세트 정확도 : {:.3f}".format(tree.score(X_train, y_train)))
    print("테스트 세트 정확도 : {:.3f}".format(tree.score(X_test, y_test)))
```

(실습과제2번 풀이)

In [14]:

```
for i in range(1,8):
    testTreeModel(0.3, i)
```

```
훈련 세트 정확도 : 0.932
테스트 세트 정확도 : 0.883
훈련 세트 정확도 : 0.972
테스트 세트 정확도 : 0.912
훈련 세트 정확도 : 0.982
테스트 세트 정확도 : 0.906
훈련 세트 정확도 : 0.985
테스트 세트 정확도 : 0.906
훈련 세트 정확도 : 0.992
테스트 세트 정확도 : 0.889
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.901
훈련 세트 정확도 : 0.997
테스트 세트 정확도 : 0.912
```

In [15]:

```
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(all_X,
                                                    all_Y,
                                                    stratify=cancer.target,
                                                    test_size = 30,
                                                    random_state=77)
tree = DecisionTreeClassifier(max_depth=2, random_state=0)
tree.fit(X_train, y_train)
```

Out[15]:

```
DecisionTreeClassifier(max_depth=2, random_state=0)
```

트리(tree)의 특성 중요도(feature importance)

- 특성 중요도 : 이 값은 0과 1사이의 숫자.
- 0은 전혀 사용되지 않음.
- 1은 완벽하게 타깃 클래스를 예측했다.
- 특성 중요도의 전체 합은 1이다.
- 특성의 feature_importance_ 값이 낮다고 해서 특성이 유용하지 않다는 것이 아니다. 단지 트리가 그 특성을 선택하지 않았다는 것.

In [16]:

```
## 특성의 이름
cancer.feature_names
```

Out[16]:

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [17]:

```
## 특성의 중요도
tree.feature_importances_
```

Out[17]:

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01305268, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.85298388, 0.        , 0.        ,
       0.        , 0.        , 0.13396343, 0.        , 0.        ])
```

In [18]:

```
cancer.data.shape[1]  # 특성 개수
```

Out[18]:

```
30
```

In [19]:

```
import matplotlib.pyplot as plt
import numpy as np
```

In [20]:

```
# 한글
import matplotlib
from matplotlib import font_manager, rc
font_loc = "C:/Windows/Fonts/malgunbd.ttf"
font_name = font_manager.FontProperties(fname=font_loc).get_name()
matplotlib.rc('font', family=font_name)
matplotlib.rcParams['axes.unicode_minus'] = False

%matplotlib inline
```

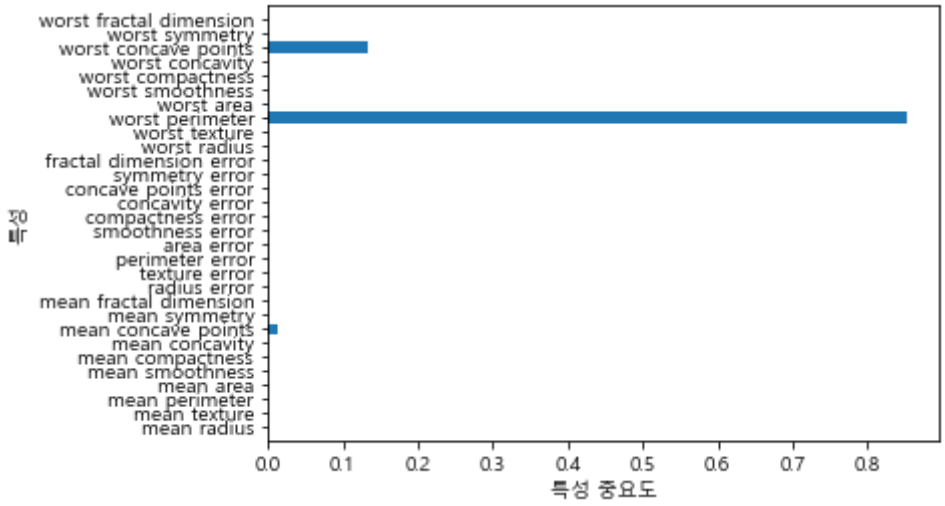
In [21]:

```
def plot_feature_imp(model, n_features):
    n_features = cancer.data.shape[1]
    imp = model.feature_importances_
    plt.barh(range(n_features) , imp, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)

    plt.xlabel("특성 중요도")
    plt.ylabel("특성")
    plt.ylim(-1, n_features)
```

In [22]:

```
fea_num = cancer.data.shape[1]
# 모델, 피처의 개수
plot_feature_imp(tree, fea_num)
```



실습과제 2

- max_depth를 1,8까지 변경해 보고 각각의 결과값을 확인해 보자.

실습해 보기

- (1) Bike 데이터 셋을 이용하여 의사결정트리 모델을 만들어 보고, 이를 이용하여 예측을 수행해 보자.
- (2) Titanic 데이터 셋을 이용하여 의사결정트리 모델을 만들어 보고, 이를 이용하여 예측을 수행해 보자.

업그레이드

- (1) https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr (https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr) 대회 데이터 다운로드 후, 의사결정트리 모델 만들어보기

History

- 2020/10 업데이트 v11
- Machine Learning with sklearn @ DJ,Lim