01. 기본- 결정트리(decision tree)

- Machine Learning with sklearn @ DJ,Lim
- date : 21/10

학습 내용

- 의사결정트리를 이미지로 이해하기
- 의사결정트리 회귀 모델 만들기

In [39]:

```
from IPython.display import display, Image
import matplotlib.pyplot as plt
import mglearn
```

In [65]:

```
import matplotlib
from matplotlib import font_manager, rc
import matplotlib.pyplot as plt
import platform
```

In [66]:

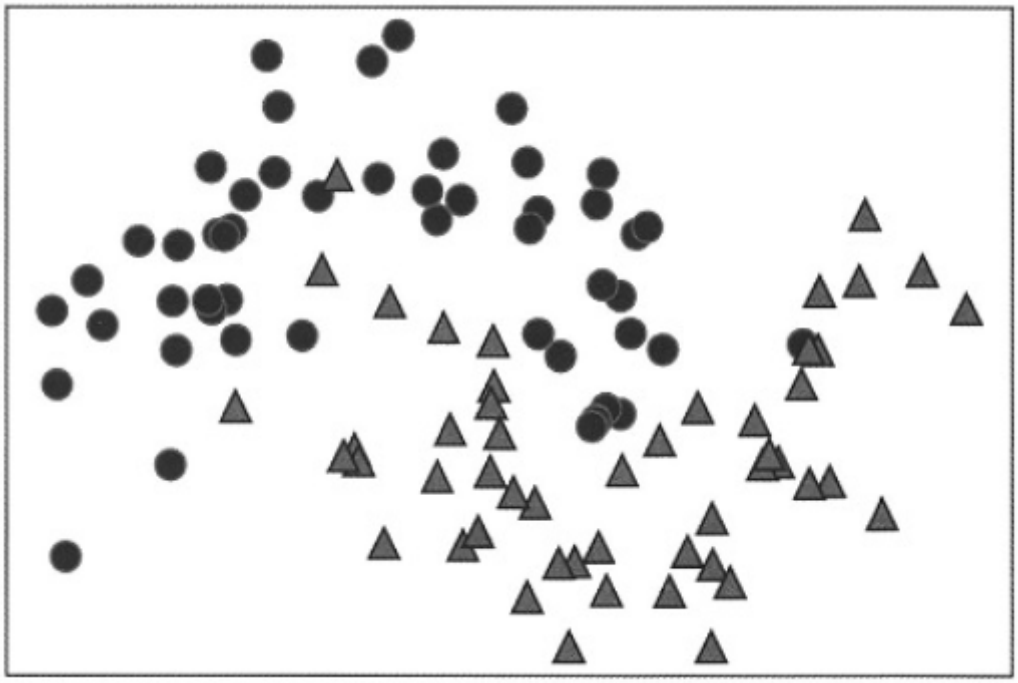
```
# 한글 및 마이너스 표시 설정
path = "C:/Windows/Fonts/malgun.ttf"
if platform.system() == "Windows":
    font_name = font_manager.FontProperties(fname=path).get_name()
    matplotlib.rc('font', family=font_name)
elif platform.system()=="Darwin":
    rc('font', family='AppleGothic')
else:
    print("Unknown System")

matplotlib.rcParams['axes.unicode_minus'] = False
%matplotlib inline
```

01. 의사결정트리- 모델 이해

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

In [67]:

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



In [68]:

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

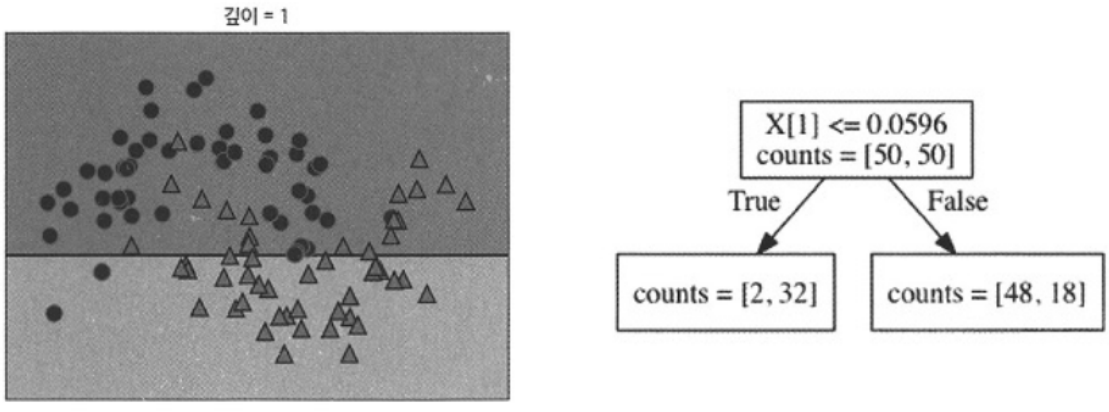


그림 2-24 깊이 1인 결정 트리(오른쪽)가 만든 결정 경계(왼쪽)

- 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 [69]:

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

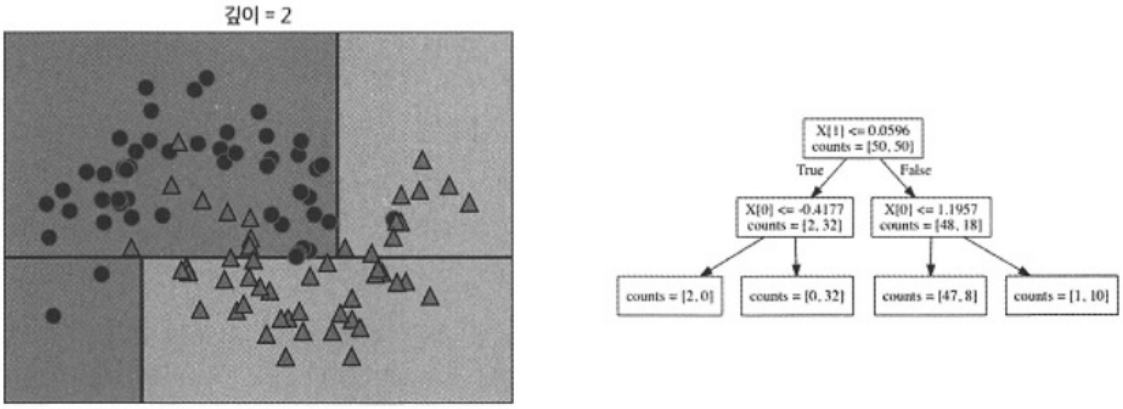


그림 2-25 깊이 2인 결정 트리(오른쪽)가 만든 결정 경계(왼쪽)

(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 [70]:

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

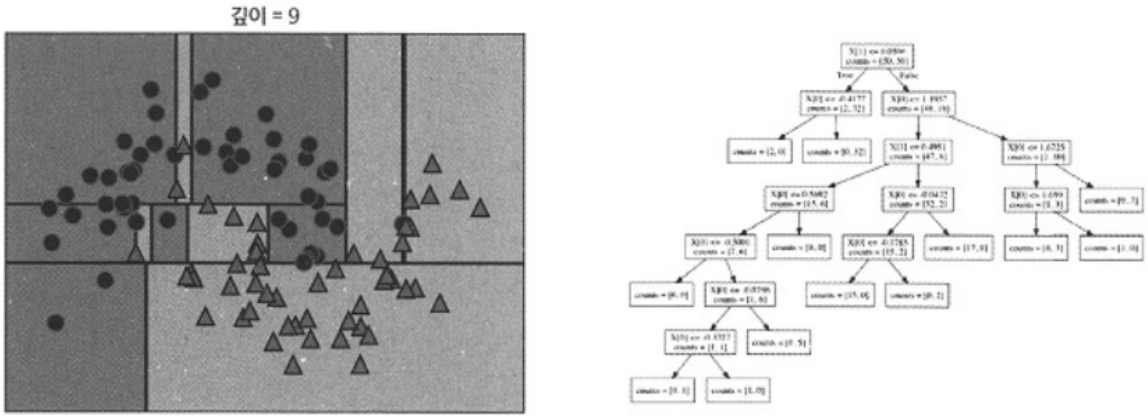


그림 2-26 깊이 9인 결정 트리의 일부(오른쪽)와 이 트리가 만든 결정 경계(왼쪽) (전체 트리는 너무 커서 일부만 표시했습니다.[37])

- 모델을 학습 후, 예측할 때, 새로운 데이터(test) 셋에 대한 예측은 **주어진 데이터 포인트가 분할한 영역 중에 어디**에 놓이는 가를 확인하면 된다.

02. 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.

실습

- 컴퓨터 메모리 가격 동향 데이터 셋 활용해 보기

In [71]:

```
import os
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
```

데이터 로드

In [72]:

```
ram_prices = pd.read_csv(os.path.join(mglearn.datasets.DATA_PATH,
                                      "ram_price.csv"))
```

시각화

- x축 : 년 (날짜)
- y축 : 가격 (해당 년도의 램(RAM) 1메가바이트당 가격) - 로그 스케일

In [73]:

```
import numpy as np
import seaborn as sns
```

In [74]:

```
np.min(ram_prices.price), np.min( np.log(ram_prices.price) )
```

Out[74]:

```
(0.0037, -5.599422459331958)
```

In [75]:

```
np.log(0.0037)
```

Out[75]:

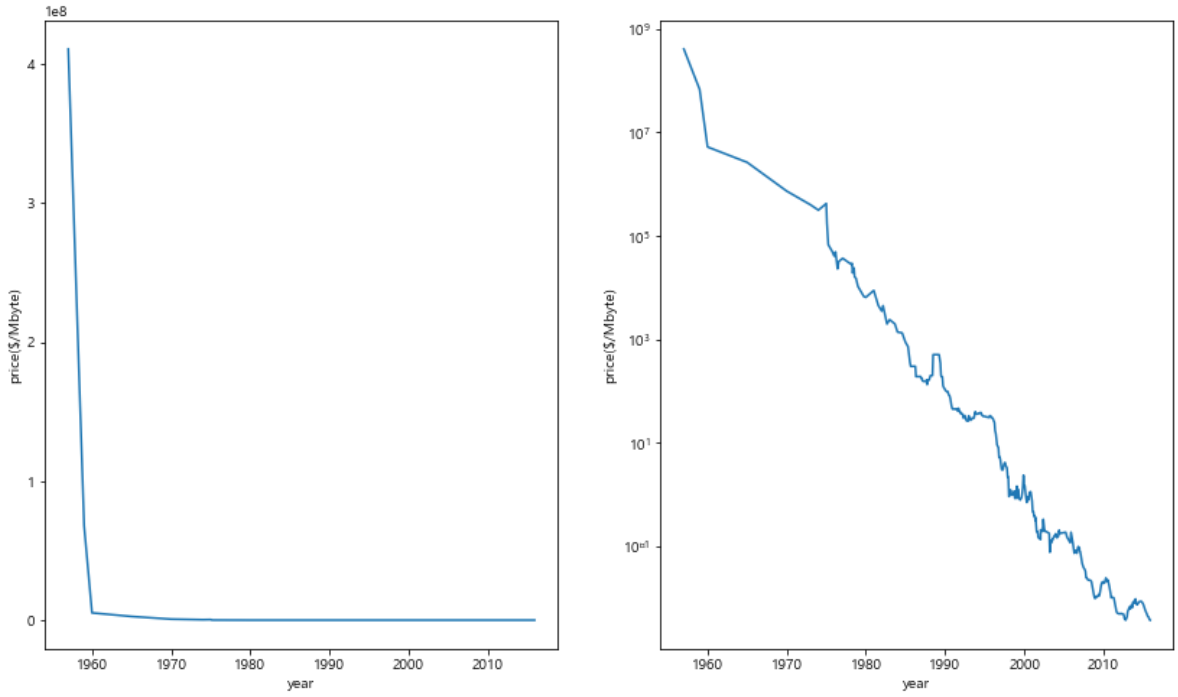
```
-5.599422459331958
```

In [76]:

```
# 한글 폰트가 지수에 음수를 표시하지 못하므로 ytick의 폰트를 바꾸어 줍니다.
# plt.yticks(fontname = "Arial")

plt.figure(figsize=(14,8))

plt.subplot(1,2,1)
plt.plot(ram_prices.date, ram_prices.price)
plt.xlabel("year")
plt.ylabel("price($/Mbyte)")

# Make a plot with log scaling on the y axis.(y축 로그 스케일)
plt.subplot(1,2,2)
plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("year")
plt.ylabel("price($/Mbyte)")
```

Out[76]:

```
Text(0, 0.5, 'price($/Mbyte)')

Font 'default' does not have a glyph for '−' [U+2212], substituting with a dummy sym
bol.
Font 'default' does not have a glyph for '−' [U+2212], substituting with a dummy sym
bol.
Font 'default' does not have a glyph for '−' [U+2212], substituting with a dummy sym
bol.
Font 'default' does not have a glyph for '−' [U+2212], substituting with a dummy sym
bol.
```



price 분포 확인

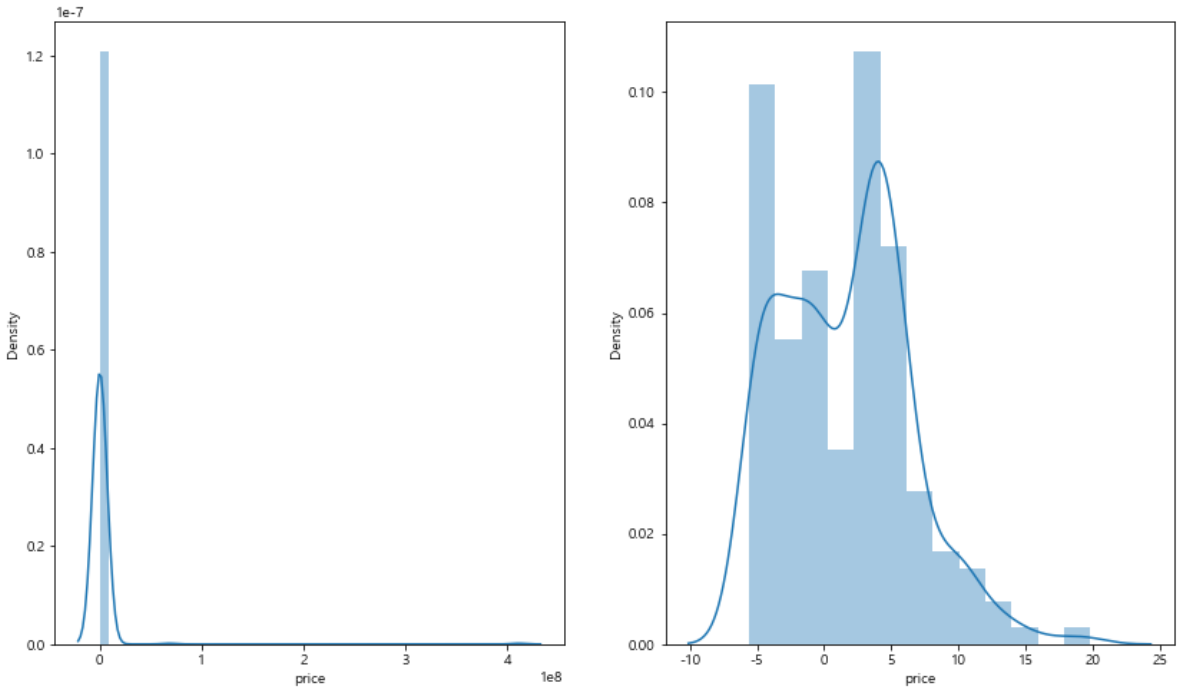
In [77]:

```
plt.figure(figsize=(14,8))

plt.subplot(1,2,1)
sns.distplot(ram_prices.price)

plt.subplot(1,2,2)
log_price = np.log(ram_prices.price)
sns.distplot(log_price)
```

```
C:₩Users₩toto₩anaconda3₩lib₩site-packages₩seaborn₩distributions.py:2557: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with similar fl
exibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:₩Users₩toto₩anaconda3₩lib₩site-packages₩seaborn₩distributions.py:2557: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with similar fl
exibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[77]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



모델 선택 및 학습

In [78]:

```
from sklearn.tree import DecisionTreeRegressor

# 2000년 이전을 학습 데이터로,습
# 2000년 이후를 테스트 데이터로 변경
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# 가격 예측을 위해 날짜 특성만을 이용합니다
X_train = data_train.date[:, np.newaxis]

# 데이터와 타깃 사이의 관계를 간단하게 만들기 위해 로그 스케일로 바꿉니다
y_train = np.log(data_train.price)
```

```
<ipython-input-78-6de4b1fb924c>:9: FutureWarning: Support for multi-dimensional inde
xing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  C
onvert to a numpy array before indexing instead.
  X_train = data_train.date[:, np.newaxis]
```

In [79]:

```
tree = DecisionTreeRegressor().fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)

# 예측은 전체 기간에 대해서 수행합니다
X_all = ram_prices.date[:, np.newaxis]

pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

# 예측한 값의 로그 스케일을 되돌립니다
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

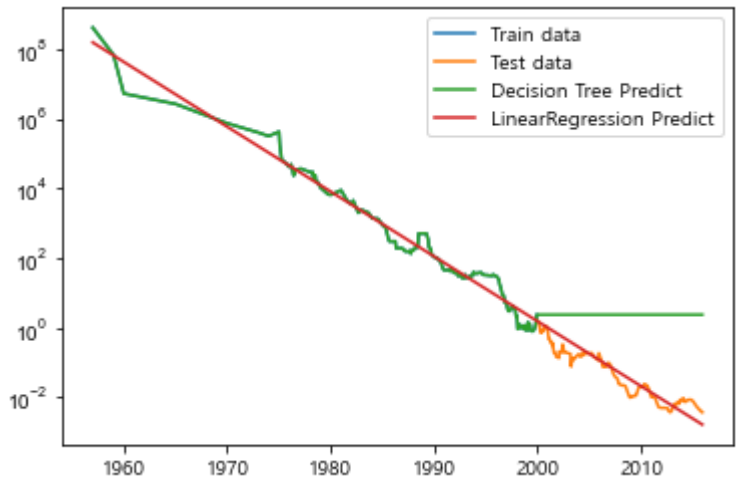
```
<ipython-input-79-95e0b1948a38>:5: FutureWarning: Support for multi-dimensional inde
xing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  C
onvert to a numpy array before indexing instead.
  X_all = ram_prices.date[:, np.newaxis]
```

In [80]:

```
plt.yticks(fontname = "Arial") # 한글 폰트가 지수에 음수를 표시하지 못하므로 ytick의 폰트를 바꾸어 줌
plt.semilogy(data_train.date, data_train.price, label="Train data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Decision Tree Predict")
plt.semilogy(ram_prices.date, price_lr, label="LinearRegression Predict")
plt.legend()
```

Out[80]:

<matplotlib.legend.Legend at 0x1ae078a8340>



두 모델은 확연한 차이를 보인다.

- (1) 선형모델은 직선으로 데이터를 근사한다.
- (2) 트리모델은 훈련 데이터를 완벽하게 예측한다.
- (3) 트리 모델은 훈련 데이터 밖의 새로운 데이터를 예측할 능력이 없다. - 과적합
- (3)번의 내용이 트리 기반 모델의 공통된 단점이다.

03. 의사결정 트리의 단점 및 장점

- 장점
  - 첫째, 만들어진 모델을 쉽게 시각화할 수 있어서, 비전문가도 이해하기 쉽다.
  - 둘째, 데이터의 스케일에 구애받지 않는다.(정규화, 표준화 전처리 과정 필요 없다.)
- 단점
  - 사전 가지치기의 사용하지만 Overfitting(과대적합) 되는 경향이 있다.
  - 리프 노드가 순수 노드가 될때까지 진행하면, **모델이 매우 복잡해지고 훈련 데이터의 과대적합 (overfitting)**이 된다. -> 순수 노드로 이루어진 트리는 훈련 세트에 100% 정확하게 맞는다.

04. 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

05. 그렇다면 어떻게 사전 가지치를 할 수 있을까?

트리의 최대 깊이 제한 (max_depth)

- max_depth = 4라면 연속된 질문의 옵션을 최대 4개로 제한
- 트리 깊이를 제한하면 과대적합이 줄어든다.
- 훈련 세트의 정확도는 떨어지지만 테스트 성능은 개선

리프의 최대 개수 제한 (max_leaf_nodes)

노드 분할을 위한 포인트의 최소 개수 지정 (min_sample_leaf)

사전 가지치기만 지원, DecisionTreeRegressor, DecisionTreeClassifier

In [81]:

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

In [82]:

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

test_size를 변경해 가면서 모델 생성

In [83]:

```
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("훈련 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 [84]:

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

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

max_depth를 변경해 가면서 모델 생성

In [85]:

```
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)))
```

In [86]:

```
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[86]:

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

06. 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 [87]:

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

Out[87]:

```
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 [88]:

```
cancer.data.shape[1]  # 특성 개수
```

Out[88]:

```
30
```

In [89]:

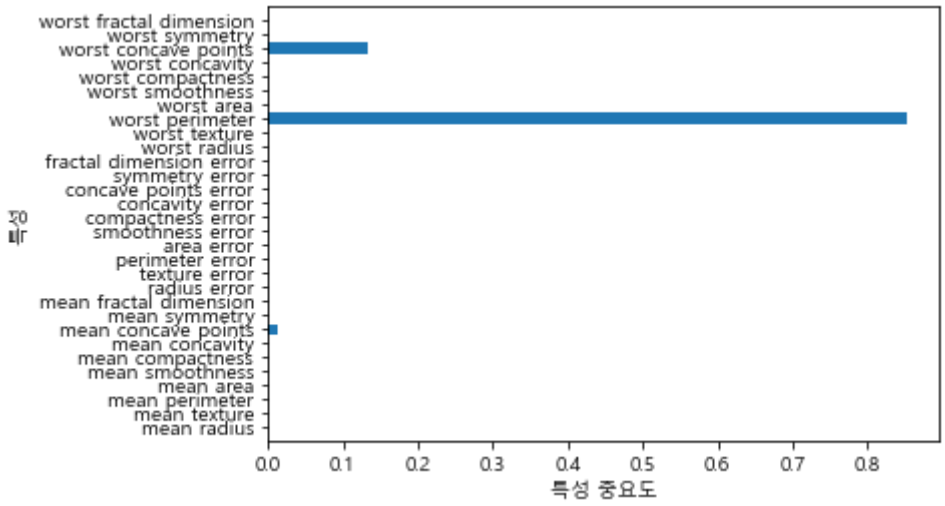
```
import matplotlib.pyplot as plt
import numpy as np

import matplotlib
from matplotlib import font_manager, rc
import matplotlib.pyplot as plt
import platform
```

In [90]:

```
def plot_feature_imp_cancer(model):
    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 [91]:

```
plot_feature_imp_cancer(tree)
```



실습과제 2

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

실습과제 3

- (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.