ch04 연속형 값을 구간 분할 해보기

학습 목표

- 1. 구간 분할 방법에 대해 알아본다.

학습 내용

- 01 경고 메세지 설정 및 한글 설정
- 02 라이브러리 불러오기
- 03 데이터 셋 준비 및 모델 예측 값의 시각화
- 04 선형회귀 모델과 의사결정트리 모델로 학습 및 예측 수행
- 05 구간 분할을 적용시켜 보자.

01 경고 메세지 설정 및 한글 설정

In [1]:
```
import os, warnings
# 경고 메시지 무시하거나 숨길때(ignore), 다시보이게(default)
# warnings.filterwarnings(action='default')
warnings.filterwarnings(action='ignore')
```

In [2]:
```
# 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
```

02 라이브러리 불러오기

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import mglearn
```

03 데이터 셋 준비 및 모델 예측 값의 시각화

In [4]:
```
X, y = mglearn.datasets.make_wave(n_samples=100)
line = np.linspace(-3,3, 1000, endpoint=False).reshape(-1,1)

print(X.shape)
print(y.shape)
```

```
(100, 1)
(100,)
```

04 선형회귀 모델과 의사결정트리 모델로 학습 및 예측 수행

In [5]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

In [6]:
```
# 모델을 선택 및 학습
linear = LinearRegression().fit(X, y)
decision = DecisionTreeRegressor(min_samples_split=3).fit(X, y)
```

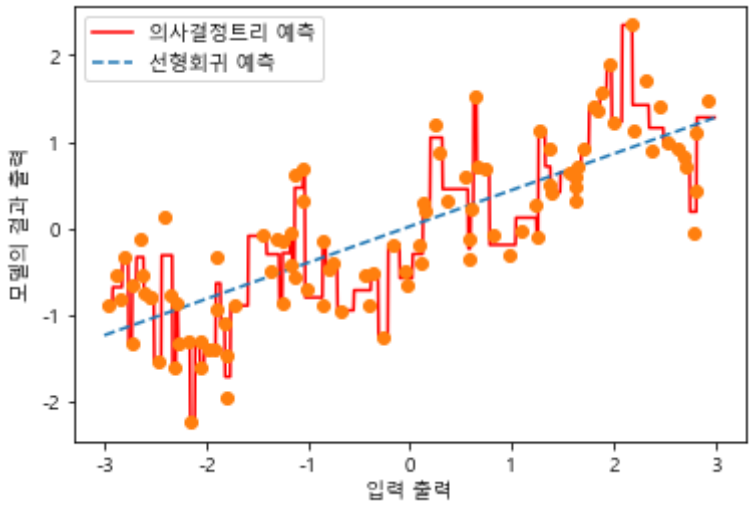
```
# 실선으로 -3~3까지의 값을 데이터로 넣어 의사 결정트리 모델 예측
plt.plot(line, decision.predict(line), 'r-', label='의사결정트리 예측')

# 점선으로 -3~3까지의 값을 데이터로 넣어 회귀 모델 예측
plt.plot(line, linear.predict(line), '--', label='선형회귀 예측')

# 모델을 만들었던 데이터 찍겠다.
plt.plot(X[:, 0], y, 'o')

plt.ylabel("모델의 결과 출력")
plt.xlabel("입력 출력")
plt.legend(loc="best")
```

Out[6]: <matplotlib.legend.Legend at 0x183e57dbee0>



05 구간 분할을 적용시켜 보자.

- 연속형 데이터에 아주 강력한 선형 모델 만드는 방법-구간 분할(bining)
- 구간의 대표값을 지정하여, 연속된 값을 구간값으로 변환하여 모델에 적용
- 함수 : KBinsDiscretizer 이용

In [7]:
```
bins = np.linspace(-3,3,11)
print("구간: {}".format(bins))
```

```
구간: [-3.  -2.4 -1.8 -1.2 -0.6  0.   0.6  1.2  1.8  2.4  3. ]
```

구간 나누기

In [8]:
```
from sklearn.preprocessing import KBinsDiscretizer
```

In [9]:
```
kb = KBinsDiscretizer(n_bins=10, strategy="uniform")
kb.fit(X)
print("bin edges : ₩n", kb.bin_edges_ )
```

```
bin edges :
 [array([-2.9668673 , -2.37804841, -1.78922951, -1.20041062, -0.61159173,
        -0.02277284,  0.56604605,  1.15486494,  1.74368384,  2.33250273,
         2.92132162])]
```

- 첫번째 구간은 -2.9668673~-2.37804741 모든 데이터 포인트를 담는다.
- 두번째 구간은 -2.37804741~-1.78922951까지 모든 데이터 포인트를 포함.

각각의 데이터를 구간에 매칭시키다.

In [10]:
```
X_binned = kb.transform(X)
X_binned
```

Out[10]:
```
<100x10 sparse matrix of type '<class 'numpy.float64'>'
        with 100 stored elements in Compressed Sparse Row format>
```

In [11]:
```
print(X[:10])  # 10개 데이터
X_binned[0:10].toarray()
```

```
[[-0.75275929]
 [ 2.70428584]
 [ 1.39196365]
 [ 0.59195091]
 [-2.06388816]
 [-2.06403288]
 [-2.65149833]
 [ 2.19705687]
 [ 0.60669007]
 [ 1.24843547]]
```

Out[11]:
```
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```

- encode='onehot-dense'로 지정하여 원-핫-인코딩된 밀집 배열 만들기

In [20]:
```
kb = KBinsDiscretizer(n_bins=10, strategy="uniform", encode='onehot-dense') # {'oneho
kb.fit(X)
X_binned = kb.transform(X)
X_binned[0:10]
```

Out[20]:
```
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```

In [21]:
```
print(X[:10])  # 10개 데이터
X_binned[:10]
```

```
[[-0.75275929]
 [ 2.70428584]
 [ 1.39196365]
 [ 0.59195091]
 [-2.06388816]
 [-2.06403288]
 [-2.65149833]
 [ 2.19705687]
 [ 0.60669007]
 [ 1.24843547]]
```

Out[21]:
```
array([[0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
```

```
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```

In [22]:
```
print(X.shape, X_binned.shape)
```

```
(100, 1) (100, 10)
```

In [23]:
```
line = np.linspace(-3,3, 1000, endpoint=False).reshape(-1,1)
line[0:10]
```

Out[23]:
```
array([[-3.   ],
       [-2.994],
       [-2.988],
       [-2.982],
       [-2.976],
       [-2.97 ],
       [-2.964],
       [-2.958],
       [-2.952],
       [-2.946]])
```

In [24]:
```
from sklearn.tree import DecisionTreeRegressor
```

line(-3~3)의 값들을 구간분할로 적용하여 시각화

```
reg = LinearRegression().fit(X_binned, y)
...
reg = DecisionTreeRegressor(min_samples_split=3).fit(X_binned, y)
```

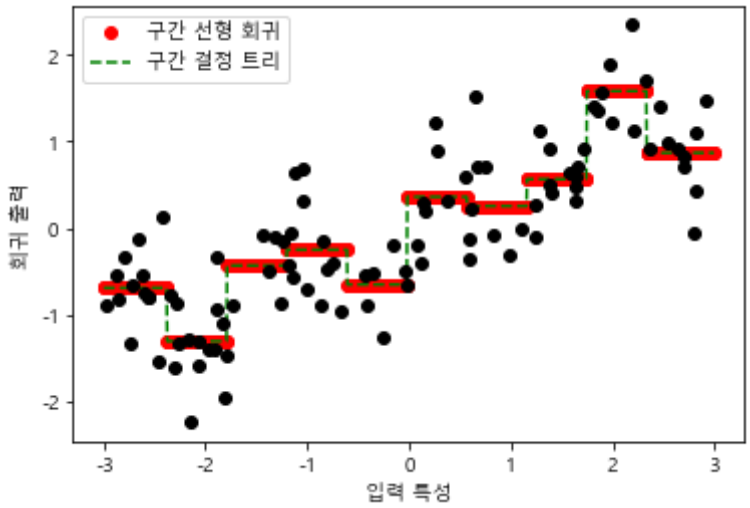
In [25]:
```
line_binned = kb.transform(line)

reg = LinearRegression().fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), 'ro', label="구간 선형 회귀")   # 점

reg = DecisionTreeRegressor(min_samples_split=3).fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), 'g--', label="구간 결정 트리")  # 점선

plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("회귀 출력")
plt.xlabel("입력 특성")
plt.legend(loc="best")
```

Out[25]:
```
<matplotlib.legend.Legend at 0x183e9ab4bb0>
```



- 선형 회귀 모델과 결정 트리가 같은 예측을 만들어냄.
- 전에 비해 선형 모델이 상당히 유연해짐
- 반면 결정트리는 특성의 값을 구간으로 나누는 것이 아무런 득이 없음.

06 원본 특성을 곱하여 선형회귀 모델 표현

- 특성을 풍부하게 나타내는 한 방법으로 원본 데이터에 **상호작용(interaction)**과 **다항식(polynomial)**을 추가하는 것.

원본 특성을 곱하여 선형 회귀 모델 표현

In [26]:
```
print(X.shape)
print(X_binned.shape)
```

```
(100, 1)
(100, 10)
```

In [38]:
```
np.min(line), np.max(line), line.shape
```

Out[38]:
```
(-3.0, 2.9939999999999998, (1000, 1))
```

In [36]:
```
X.shape, X_binned.shape
```

Out[36]:
```
((100, 1), (100, 10))
```

In [37]:
```
# 원본 데이터와 구간화된 데이터 결합
X_combined = np.hstack([X, X_binned])
print(X_combined.shape)
X_combined[0:3]
```

```
(100, 11)
```

Out[37]:
```
array([[-0.75275929,  0.        ,  0.        ,  0.        ,  1.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 2.70428584,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [ 1.39196365,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  1.        ,  0.        ,
         0.        ]])
```

In [39]:
```
line.shape, line_binned.shape
```

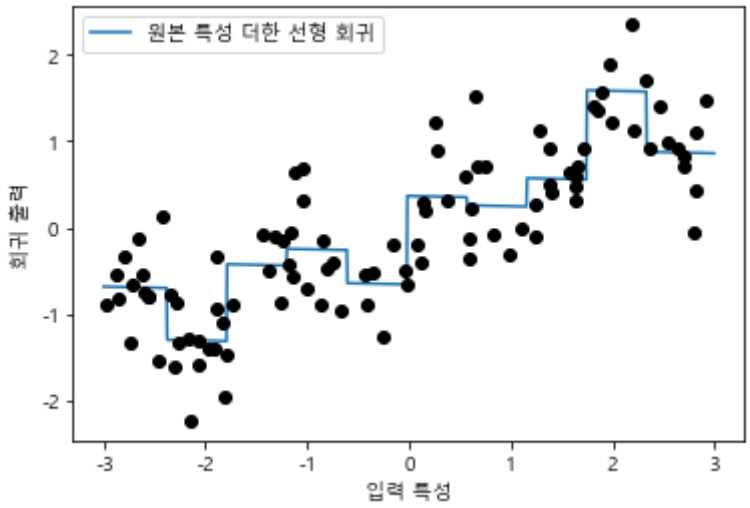
Out[39]:
```
((1000, 1), (1000, 10))
```

In [40]:
```
# 회귀 모델
reg = LinearRegression().fit(X_combined, y)

# line_binned = kb.transform(line)
line_combined = np.hstack([line, line_binned])

plt.plot(line, reg.predict(line_combined), label="원본 특성 더한 선형 회귀")
plt.ylabel("회귀 출력")
plt.xlabel("입력 특성")
plt.plot(X[:, 0], y, 'o', c='k')
plt.legend(loc="best")
```

Out[40]:
```
<matplotlib.legend.Legend at 0x183e9c45c10>
```



In [42]:
```
X_product = np.hstack([X_binned, X * X_binned])
print(X_product.shape)

# 회귀 모델
reg = LinearRegression().fit(X_product, y)

# line_binned = kb.transform(line)
line_product = np.hstack([line_binned, line * line_binned])

print(line.shape)
print(line_binned.shape)


plt.plot(line, reg.predict(line_product), label="원본 특성 곱한 선형 회귀")
plt.ylabel("회귀 출력")
plt.xlabel("입력 특성")
plt.legend(loc="best")
plt.plot(X[:, 0], y, 'o', c='k')
```

```
(100, 20)
(1000, 1)
(1000, 10)
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x183e9d01250>]
```