분류용 선형 모델 - Logistic, SVM ¶

- 선형 모델은 분류에도 널리 사용된다.

학습 내용

01. Logistic에 대해 알아본다.

02. 선형 분류 알고리즘 LogisticRegression, LinearSVC

03. 예제를 통한 이해(유방암 데이터 셋)

04. 다중 클래스 분류용 선형 모델

In [37]:
```
import mglearn
import matplotlib.pyplot as plt
import numpy as np
```

In [50]:
```
### 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   # 그래프 마이너스 표시
```

1-1 이진 분류에 대해 알아본다.

이진분류를 위한 예측 방정식

$$\hat{y} = w_1 \text{ x } x_1 + w_2 \text{ x } x_2 + \ldots + w_p \text{ x } x_p + \text{b}$$

$$\hat{y} = w_1 \text{ x } x_1 + w_2 \text{ x } x_2 + \ldots + w_p \text{ x } x_p + \text{b} > 0$$

- 특성(feature)의 개수 p, 가중치(w), 특성(feature) : x, b:(절편)
- 선형 회귀와 비슷하다. 특성(x)와 가중치(w)의 곱의 합을 임계치 0과 비교한다.
- 0보다 작으면 클래스를 -1, 0보다 크면 +1이라고 예측

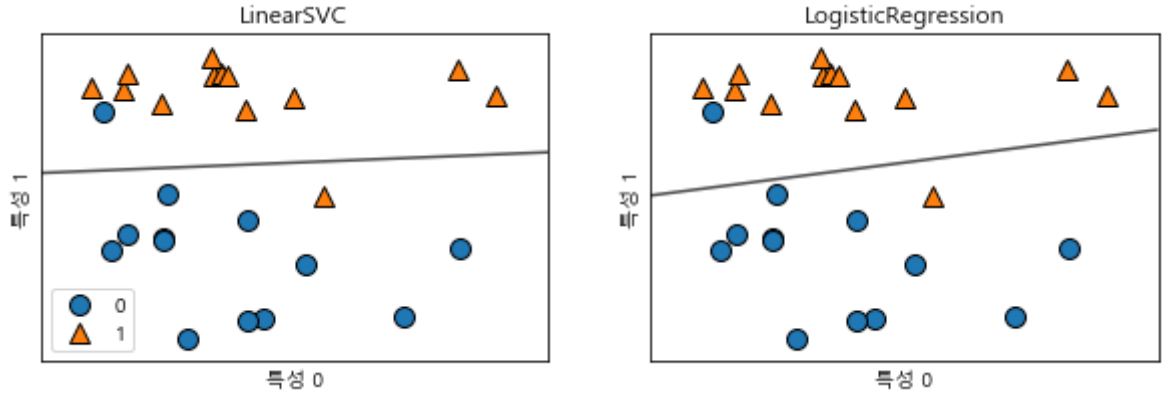
(이진) 선형 분류기는 선, 평면, 초평면(3차원 이상)을 사용해서 두 개의 클래스를 구분하는 분류기이다.

1-2 선형 분류 알고리즘 LogisticRegression, LinearSVC

(가) 가장 잘 알려진 선형 분류 알고리즘은 로지스틱 회귀(Logistic regression)과 서포트 벡터 머신(SVM:support vector machine)이다.

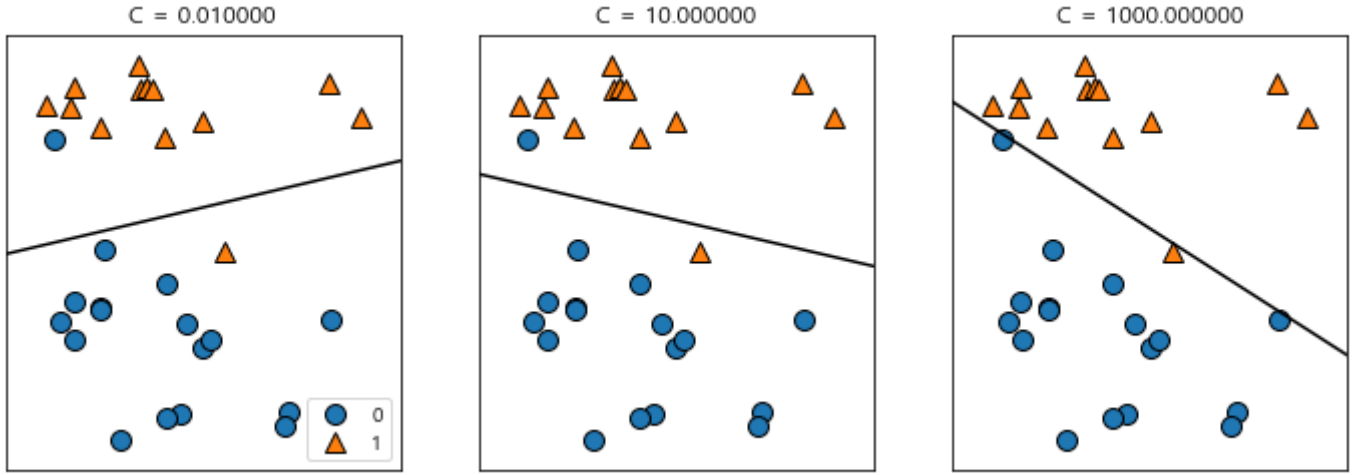
In [51]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

X, y = mglearn.datasets.make_forge()   # 데이터 만들기

fig, axes = plt.subplots(1, 2, figsize=(10, 3))   # 1행 2열의 크기 10, 3인 그래프 틀 만들기

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)              # 모델 학습(fit)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
                                    ax=ax, alpha=.7)              # 그래프를 통한 확인(mglearn)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))  # 모델명
    ax.set_xlabel("특성 0")
    ax.set_ylabel("특성 1")
axes[0].legend()
```

Out[51]: <matplotlib.legend.Legend at 0x15e1bd54470>



- 첫번째 특성을 x축에 놓고, 두번째 특성을 y축에 위치 시킴.
- 위쪽은 클래스 1, 아래쪽은 클래스 0으로 나뉘어짐.
- 두 모델은 기본적으로 Ridge와 마찬가지로 L2규제를 사용함.
- 매개변수 C는 높아지면 규제가 감소되어 0에서 멀어지고, C의 값이 낮아지면 모델은 계수(w)가 0에 가까워지도록 만든다.

In [52]:
```
mglearn.plots.plot_linear_svc_regularization()
```



- C가 커지면 커질수록 잘못된 데이터에 민감해서 최대한 분류하려고 결정경계가 기울어짐.
- C가 큰 모델이 학습용 데이터를 잘 분류하려고 함. 과대적합의 가능성.

주의할 점 : feature(특성)이 적은 차원의 데이터에서는 결정 경계가 직선이거나 평면이된다. 매우 제한적이다. 하지만 고차원(특성이 많은)의 분류에서는 선형 모델이 매우 강력해 진다. 특성이 많아지면 과대적합하지 않도록 하는 것이 매우 중요하다.

03. 예제를 통한 이해(유방암 데이터 셋)

In [53]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
print(cancer.data.size)    # 크기
print(cancer.data.shape)   # 행과 열
print(cancer.data.ndim)    # 배열 차원 수
print(cancer.DESCR[:1500]) # 설명
print(cancer.keys())       # 키값
```

```
17070
(569, 30)
2
Breast Cancer Wisconsin (Diagnostic) Database
=============================================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
```

```
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

In [54]:
```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)

logreg = LogisticRegression().fit(X_train, y_train)

print("훈련 세트 점수: {:.3f}".format(logreg.score(X_train, y_train)))
print("테스트 세트 점수: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
훈련 세트 점수: 0.955
테스트 세트 점수: 0.958
```

기본값 C=1이 훈련세트와 테스트 세트 양쪽에서 95%의 정확도로 꽤 훌륭한 성능을 내고 있다.

C가 100일때

- 훈련세트의 정확도가 높아졌고, 테스트 세트의 정확도도 조금 증가함. (복잡도가 높을 수록 성능이 좋다.)

In [55]:
```
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("훈련 세트 점수: {:.3f}".format(logreg100.score(X_train, y_train)))
print("테스트 세트 점수: {:.3f}".format(logreg100.score(X_test, y_test)))
```

```
훈련 세트 점수: 0.972
테스트 세트 점수: 0.965
```

C가 0.01일때

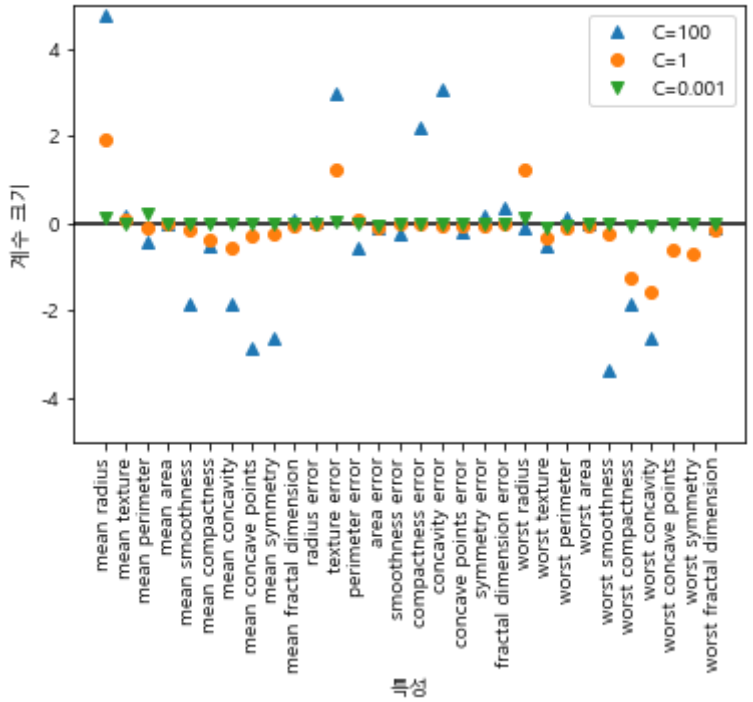
In [56]:
```
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("훈련 세트 점수: {:.3f}".format(logreg001.score(X_train, y_train)))
print("테스트 세트 점수: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
훈련 세트 점수: 0.934
테스트 세트 점수: 0.930
```

모델의 계수를 비교

In [57]:
```
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
xlims = plt.xlim()
plt.hlines(0, xlims[0], xlims[1])
plt.xlim(xlims)
plt.ylim(-5, 5)
plt.xlabel("특성")
plt.ylabel("계수 크기")
plt.legend()
```

Out[57]: <matplotlib.legend.Legend at 0x15e1bda6d30>



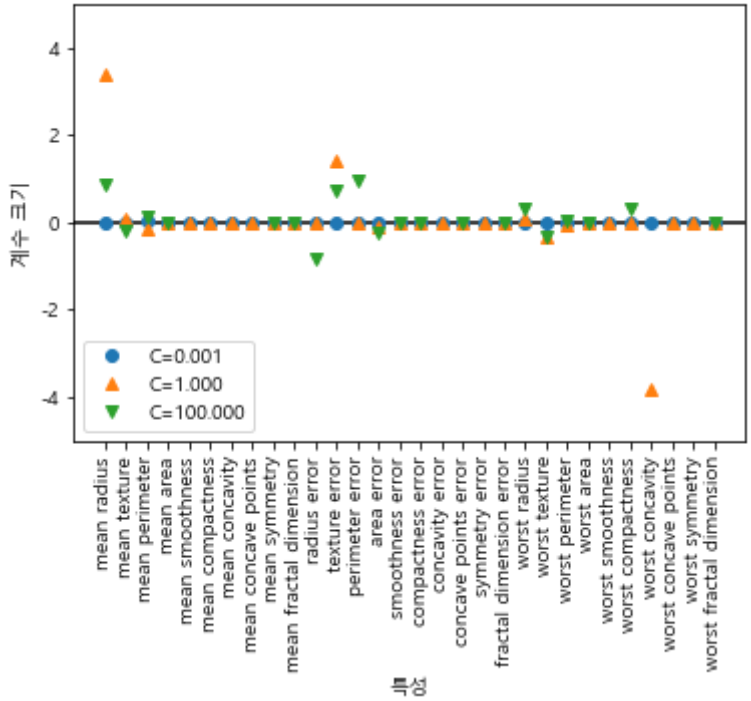
LogisticRegression은 기본으로 L2규제를 적용하여 Ridge로 만든 모습과 비슷하다.

더 쉬운 모델을 사용하려면 L1규제를 사용하는 것이 좋음.(Lasso: 특성을 0으로 만든다.)

In [58]:
```
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
    lr_l1 = LogisticRegression(C=C, penalty="l1").fit(X_train, y_train)
    print("C={:.3f} 인 l1 로지스틱 회귀의 훈련 정확도: {:.2f}".format(
          C, lr_l1.score(X_train, y_train)))
    print("C={:.3f} 인 l1 로지스틱 회귀의 테스트 정확도: {:.2f}".format(
          C, lr_l1.score(X_test, y_test)))
    plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))

plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
xlims = plt.xlim()
plt.hlines(0, xlims[0], xlims[1])
plt.xlim(xlims)
plt.xlabel("특성")
plt.ylabel("계수 크기")

plt.ylim(-5, 5)
plt.legend(loc=3)
```

```
C=0.001 인 l1 로지스틱 회귀의 훈련 정확도: 0.91
C=0.001 인 l1 로지스틱 회귀의 테스트 정확도: 0.92
C=1.000 인 l1 로지스틱 회귀의 훈련 정확도: 0.96
C=1.000 인 l1 로지스틱 회귀의 테스트 정확도: 0.96
C=100.000 인 l1 로지스틱 회귀의 훈련 정확도: 0.99
C=100.000 인 l1 로지스틱 회귀의 테스트 정확도: 0.98
```

Out[58]:
```
<matplotlib.legend.Legend at 0x15e1bdb7240>
```

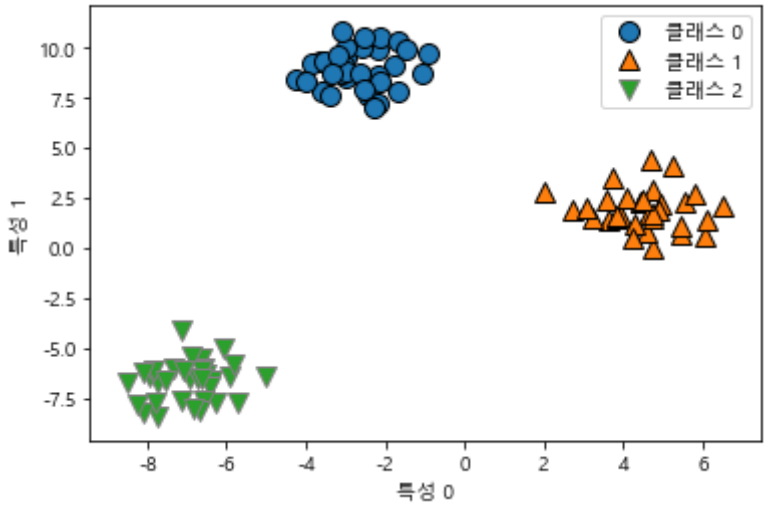


04. 다중 클래스 분류용 선형 모델

- 많은 선형 분류 모델은 태생적으로 이진 분류만을 지원한다.
- 즉 다중 클래스(multiclass)를 지원하지 않는다.
- 다중 클래스의 예측은 클래스의 수만큼 이진 분류 모델이 만들어집니다.
- 예측을 할 때 이렇게 만들어진 모든 이진 분류기가 작동하여 가장 높은 점수를 내는 분류기의 클래스를 예측값으로 선택하게 된다.

In [59]:
```
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
plt.legend(["클래스 0", "클래스 1", "클래스 2"])
```

Out[59]: <matplotlib.legend.Legend at 0x15e182b54a8>



클래스가 3개이기에 계수 배열의 크기 3행이다.

특성을 2개를 사용하여 특성에 따른 계수 값을 두개 갖고 있음.

절편(intercept)는 클래스가 3개이므로 이도 또한 3개이다.

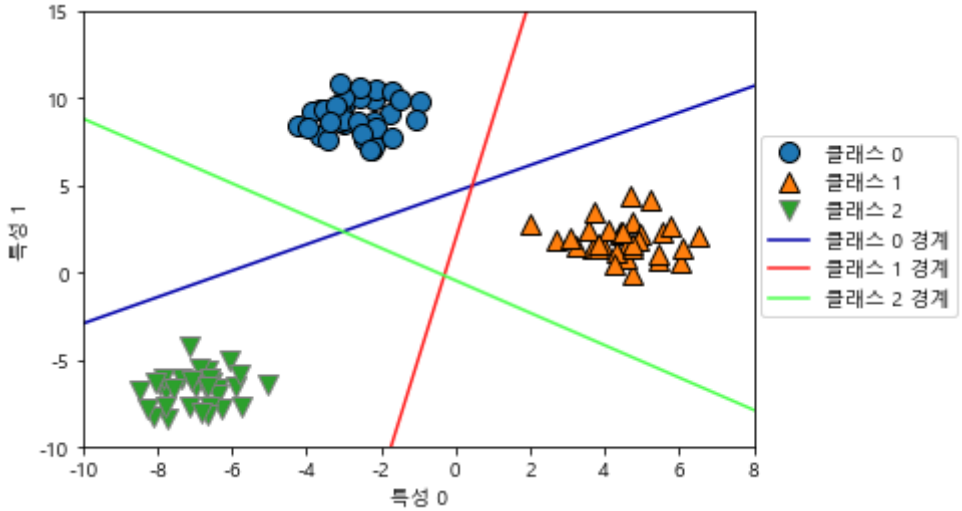
In [60]:
```
linear_svm = LinearSVC().fit(X, y)
print("계수 배열의 크기: ", linear_svm.coef_.shape)
print("절편 배열의 크기: ", linear_svm.intercept_.shape)
```

```
계수 배열의 크기:  (3, 2)
절편 배열의 크기:  (3,)
```

세개의 이진 분류기가 만드는 경계의 시각화

In [61]:
```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
                                  mglearn.cm3.colors):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
plt.legend(['클래스 0', '클래스 1', '클래스 2', '클래스 0 경계', '클래스 1 경계',
            '클래스 2 경계'], loc=(1.01, 0.3))
```

Out[61]: <matplotlib.legend.Legend at 0x15e1be9d8d0>

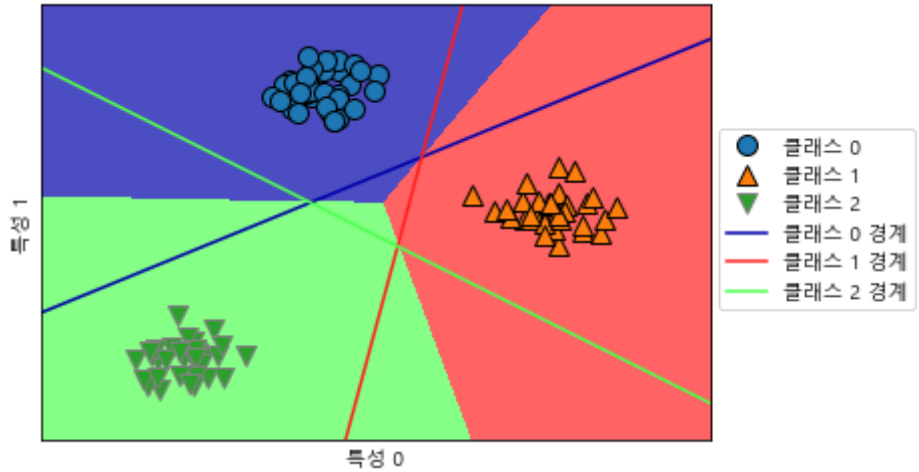


In [62]:
```
#### 데이터의 클래스 0에 속한 모든 포인트는 클래스 0을 구분하는 직선 위에 놓인다.
```

여기서 잠깐! 그림 중앙의 삼각형 영역은 어떻게 분류가 될까?

In [63]:
```
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
                                  mglearn.cm3.colors):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['클래스 0', '클래스 1', '클래스 2', '클래스 0 경계', '클래스 1 경계',
            '클래스 2 경계'], loc=(1.01, 0.3))
plt.xlabel("특성 0")
plt.ylabel("특성 1")
```

Out[63]: Text(0,0.5,'특성 1')



Summary

(가) 회귀모델에서의 매개변수는 alpha이다.

(나) LinearSVC와 LogisticRegression에서의 매개변수는 C이다.

(다) alpha값이 클수록, C값이 작을 수록 모델이 단순해 진다.

(라) 회귀모델에서의 매개변수를 조정하는 일이 매우 중요하다

(마) 중요한 특성이 많지 않다고 생각하면 L1규제(Lasso), 그렇지 않으면 기본적으로 L2규제를 사용한다.

(바) 선형 모델은 학습 속도가 빠르고 예측도 빠르다. 매우 큰 데이터셋과 희소한 데이터 셋에 잘 작동한다.

(사) 선형 모델은 샘플에 비해 특성이 많을 때 잘 작동한다.

(아) 저차원의 데이터 셋에서는 다른 모델들의 일반화 성능이 더 좋다.

이외의 내용

(가) 수십만개 수백만 개의 샘플로 이루어진 대용량 데이터셋이라면 기본 설정보다 빨리 처리하도록 Ridge에 solver='sag'옵션을 둔다.

(나) 다른 대안으로 대용량 처리 버전으로 구현된 SGDClassifier와 SGDRegressor를 사용할 수 있다.

In [ ]: