

Introduction to Machine Learning with Python

2. Supervised Learning(1)

Honedae Machine Learning Study Epoch #2

Contacts

Haesun Park

Email : haesunpark@gmail.com

Meetup: <https://www.meetup.com/Hongdae-Machine-Learning-Study/>

Facebook : <https://facebook.com/haesunpark>

Blog : <https://tensorflow.blog>

Book

파이썬 라이브러리를 활용한 머신러닝, 박해선.

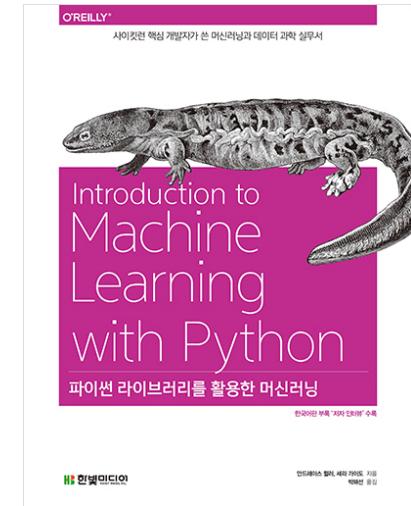
(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

Github:

https://github.com/rickiepark/introduction_to_ml_with_python/



지도 학습

지도 학습supervised learning

훈련(training, learning)

입력과 출력의 샘플 데이터가 있을 때 새로운 입력에 대한 출력을 예측합니다.

예측(predict, Inference)

사전에 데이터를 만드는 노력이 필요합니다(자동화 필요).

분류 classification와 회귀 regression

분류는 가능성 있는 클래스 레이블 중 하나를 예측합니다.

이진 분류 binary classification의 예: 두 개 클래스 분류(스팸, 0-음성클래스, 1-양성클래스)

다중 분류 multiclass classification의 예: 셋 이상의 클래스 분류(붓꽃 품종)

언어의 종류를 분류(한국어와 프랑스어 사이에 다른 언어가 없음)

대부분 머신러닝 문제가 분류의 문제입니다.

회귀는 연속적인 숫자(실수)를 예측합니다.

예) 교육, 나이, 주거지를 바탕으로 연간 소득 예측

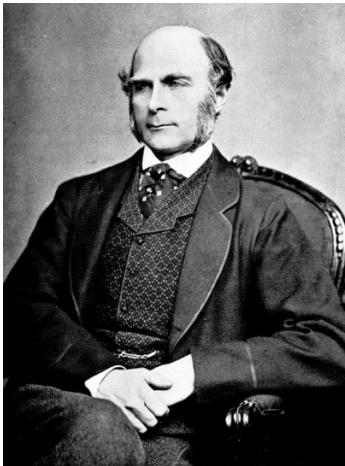
전년도 수확량, 날씨, 고용자수로 올해 수확량 예측

예측 값에 미묘한 차이가 크게 중요하지 않습니다.(연봉 예측의 경우 40,000,001 원이나 39,999,999원이 문제가 되지 않습니다.)

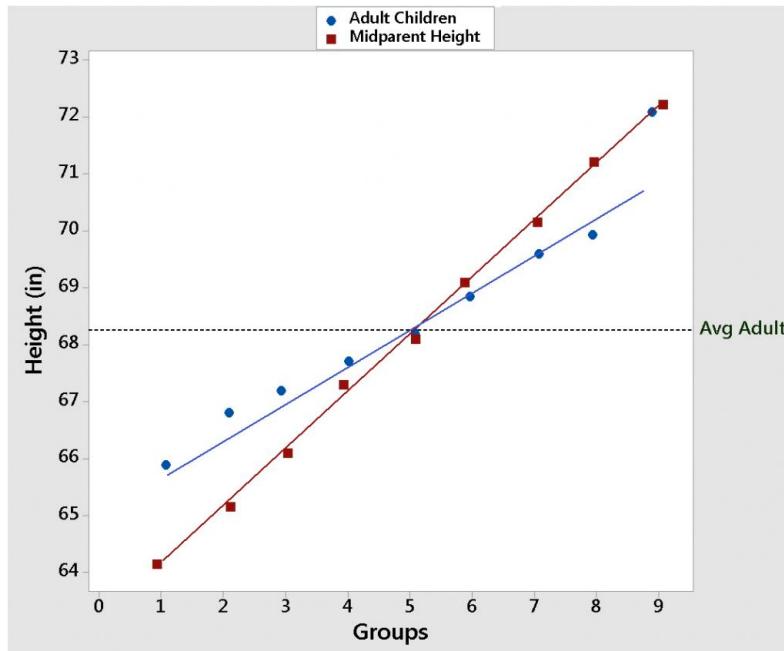
통계학에서 많이 연구되어 왔습니다.

회귀 - Regression

“regression toward the mean”



프랜시스 골턴 Francis Galton



일반화, 과대적합, 과소적합

일반화 generalization

훈련 세트로 학습한 모델을 테스트 세트에 적용하는 것입니다.
(예, 모델이 테스트 세트에 잘 일반화가 되었다)

과대적합 overfitting

훈련 세트에 너무 맞추어져 있어 테스트 세트의 성능 저하되는 현상입니다.
(예, 모델이 훈련 세트에 과대적합되어 있다)

과소적합 underfitting

훈련 세트를 충분히 반영하지 못해 훈련 세트, 테스트 세트에서 모두 성능 저하되는 현상입니다. (예, 모델이 너무 단순하여 과소적합되어 있다)

너무 상세한 모델 → 과대적합

요트회사의 고객 데이터를 활용해 요트를 사려는 사람을 예측하려고 합니다.

나이	보유차량수	주택보유	자녀수	혼인상태	애완견	보트구매
66	1	yes	2	사별	no	yes
52	2	yes	3	기혼	no	yes
22	0	no	0	기혼	yes	no
25	1	no	1	미혼	no	no
44	0	no	2	이혼	yes	no
39	1	yes	2	기혼	yes	no
26	1	no	2	미혼	no	no
40	3	yes	1	기혼	yes	no
53	2	yes	2	이혼	no	yes
64	2	yes	3	이혼	no	no
58	2	yes	2	기혼	yes	yes
33	1	no	1	미혼	no	no

45세 이상, 자녀 셋 미만,
이혼하지 않은 고객이
요트를 살 것이다.

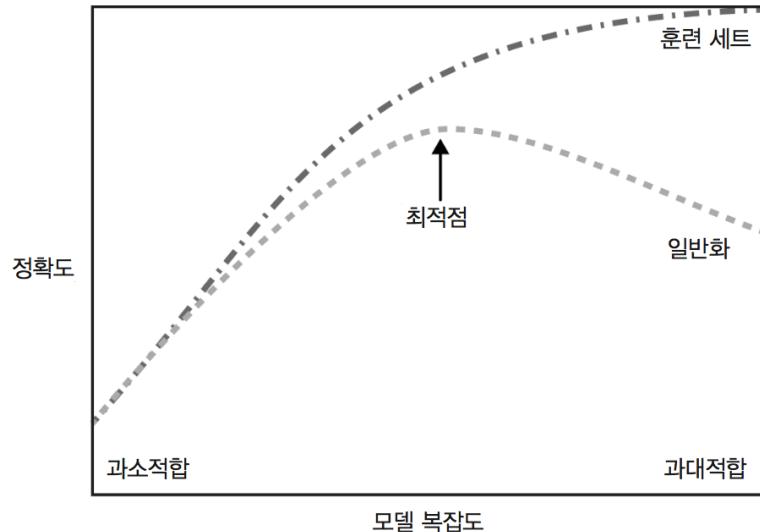
만약 10,000개의
데이터가 모두 이
조건을 만족한다면?

너무 간단한 모델 → 과소적합

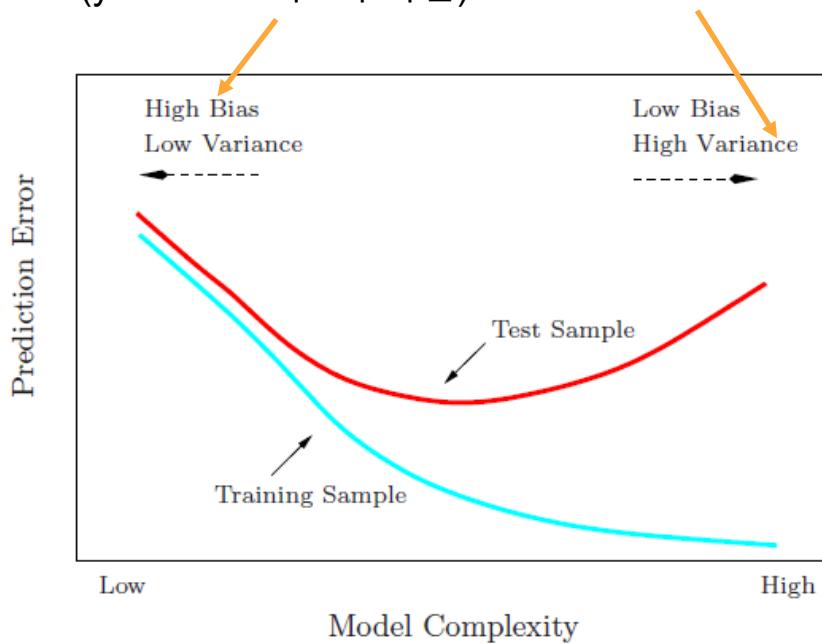
집이 있는 고객은
모두 요트를 살 것이다.

나이	보유차량수	주택보유	자녀수	혼인상태	애완견	보트구매
66	1	yes	2	사별	no	yes
52	2	yes	3	기혼	no	yes
22	0	no	0	기혼	yes	no
25	1	no	1	미혼	no	no
44	0	no	2	이혼	yes	no
39	1	yes	2	기혼	yes	no
26	1	no	2	미혼	no	no
40	3	yes	1	기혼	yes	no
53	2	yes	2	이혼	no	yes
64	2	yes	3	이혼	no	no
58	2	yes	2	기혼	yes	yes
33	1	no	1	미혼	no	no

모델 복잡도 곡선



알고리즘 자체 오차
($y = ax + b$ 의 b 가 아님) 데이터에 대한 민감성

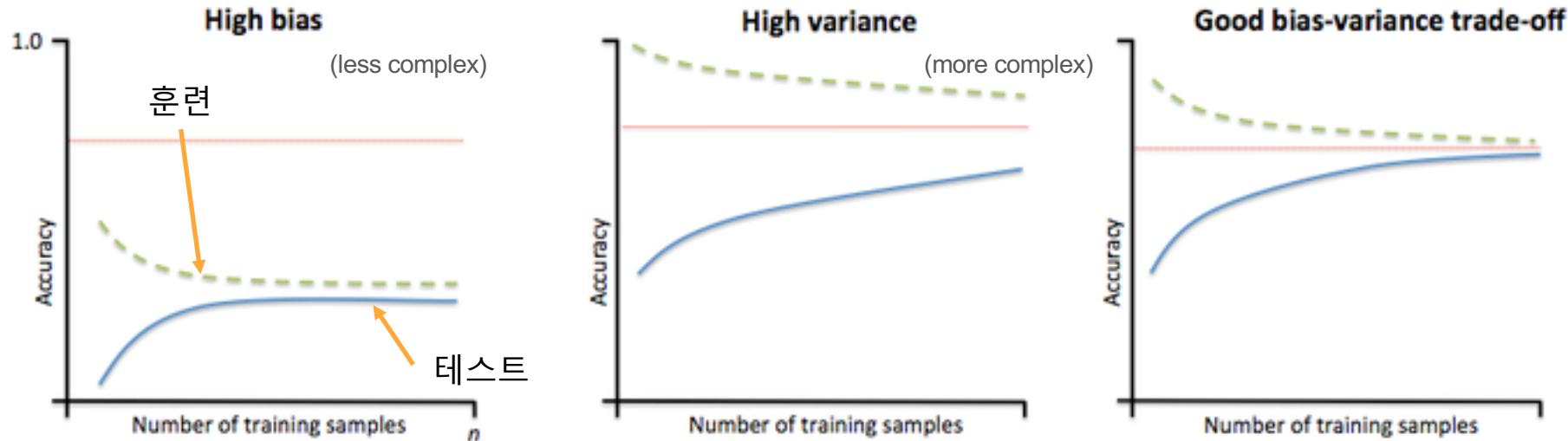


편향-분산 트레이드오프 bias-variance tradeoff라고도 부릅니다.

데이터셋과 복잡도 관계

데이터가 많으면 다양성이 커져 복잡한 모델을 만들 수 있습니다.

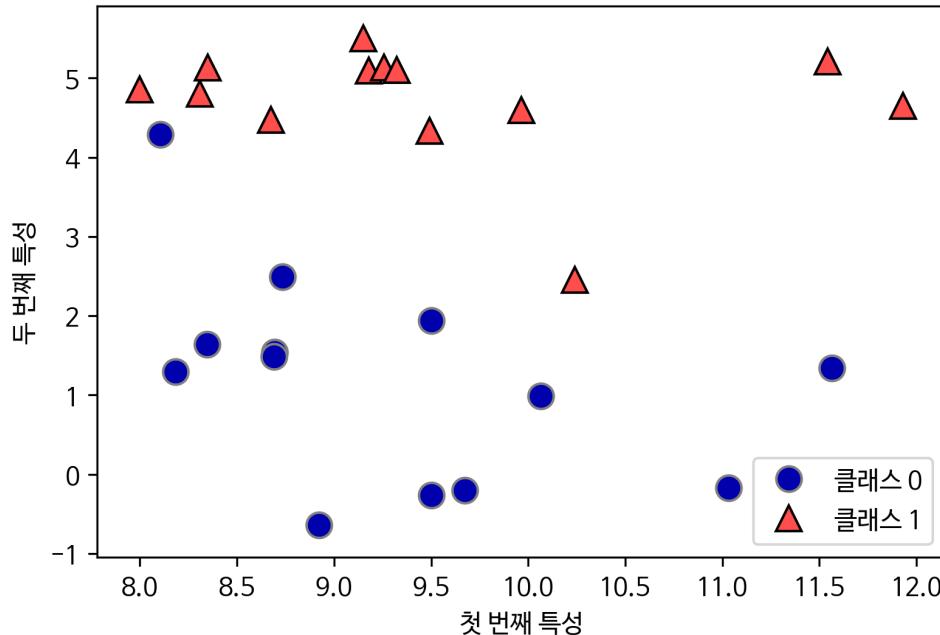
10,000명의 고객 데이터에서 45세 이상, 자녀 셋 미만, 이혼하지 않은 고객이
요트를 사려한다면 이전보다 훨씬 신뢰 높은 모델이라고 할 수 있습니다.



데이터셋

forge 데이터셋

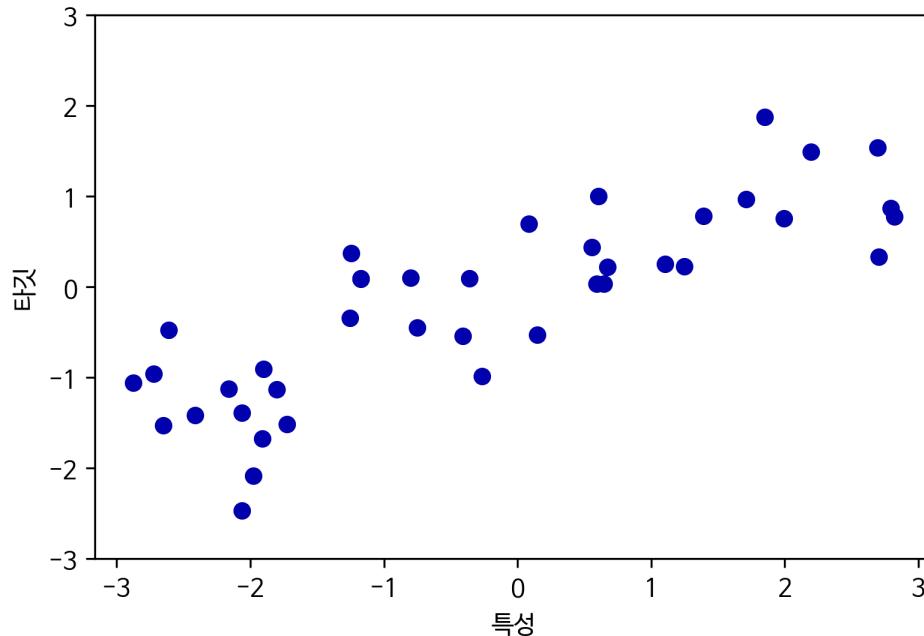
인위적으로 만든 이진 분류 데이터셋, 26개의 데이터 포인트, 2개의 속성



데이터셋

wave 데이터셋

인위적으로 만든 회귀 데이터셋, 40개의 데이터 포인트, 1개의 속성



데이터셋

위스콘신 유방암 데이터셋

악성 Malignant(1)/양성 Benign(0)의 이진 분류, `load_breast_cancer()`,
569개의 데이터 포인트, 30개의 특성

보스턴 주택가격 데이터셋

1970년대 보스턴 주변의 주택 평균가격 예측, 회귀 데이터셋, `load_boston()`,
506개의 데이터 포인트, 13개의 특성

확장 보스턴 주택가격 데이터셋

특성끼리 곱하여 새로운 특성을 만듦(특성 공학, 4장, `PolynomialFeatures`),
`mglearn.datasets.load_extended_boston()`, 104개의 데이터 포인트

중복 조합 공식은 $\binom{n}{k} = \binom{n+k-1}{k}$ 이므로 $\binom{13}{2} = \binom{13+2-1}{2} = \frac{14!}{2!(14-2)!} = 91$

특성의 제곱이 만들어짐

양성 positive, 음성 negative



load_breast_cancer()

```
In [5]: from sklearn.datasets import load_breast_cancer  
cancer = load_breast_cancer()  
print("cancer.keys(): {}".format(cancer.keys()))  
  
cancer.keys(): dict_keys(['feature_names', 'DESCR', 'target', 'target_names', 'data'])
```

Bunch 클래스의 키

```
In [6]: print("유방암 데이터의 형태: {}".format(cancer.data.shape))
```

유방암 데이터의 형태: (569, 30) ← (샘플, 특성)

```
In [7]: print("클래스별 샘플 갯수:\n{}".format(  
    {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
```

클래스별 샘플 갯수:
{'benign': 357, 'malignant': 212} → 타깃

```
In [8]: print("특성 이름: \n{}".format(cancer.feature_names))
```

특성 이름:
['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']

load_boston()

```
In [9]: from sklearn.datasets import load_boston  
boston = load_boston()  
print("데이터의 형태: {}".format(boston.data.shape))
```

데이터의 형태: (506, 13) ← 원래 13개 특성

```
In [10]: X, y = mglearn.datasets.load_extended_boston()  
print("X.shape: {}".format(X.shape))
```

X.shape: (506, 104)

확장된 104개 특성

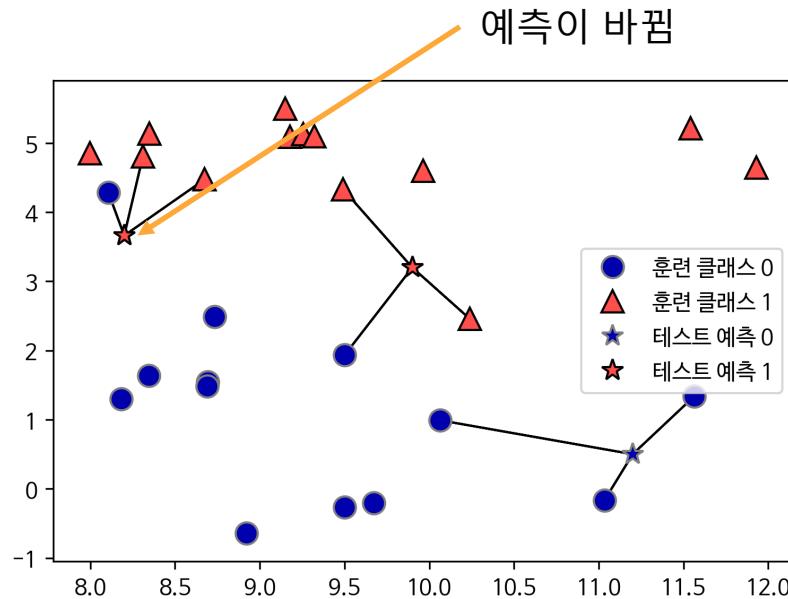
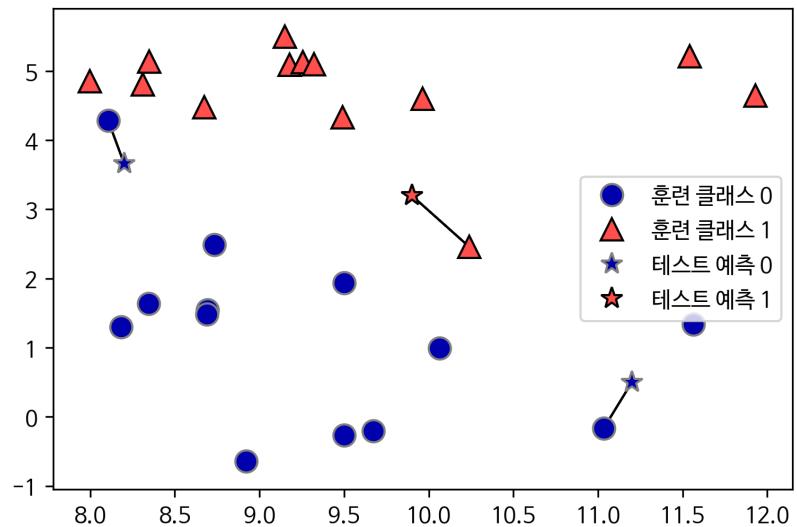
이 책의 예제를 위해서 만든
확장된 데이터셋

k-최근접 이웃 분류

k-최근접 이웃 분류

새로운 데이터 포인트에 가까운 이웃 중 다수 클래스 majority voting를 예측합니다.

forge 데이터셋에 1-최근접 이웃, 3-최근접 이웃 적용하면 다음과 같습니다.



k-NN 분류기

훈련 세트와 테스트 세트로 분리

```
In [13]: from sklearn.model_selection import train_test_split  
X, y = mglearn.datasets.make_forge()  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
In [14]: from sklearn.neighbors import KNeighborsClassifier  
clf = KNeighborsClassifier(n_neighbors=3) ← 모델 객체 생성
```

```
In [15]: clf.fit(X_train, y_train) ← 모델 학습
```

```
Out[15]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
metric_params=None, n_jobs=1, n_neighbors=3, p=2,  
weights='uniform')
```

```
In [16]: print("테스트 세트 예측: {}".format(clf.predict(X_test)))
```

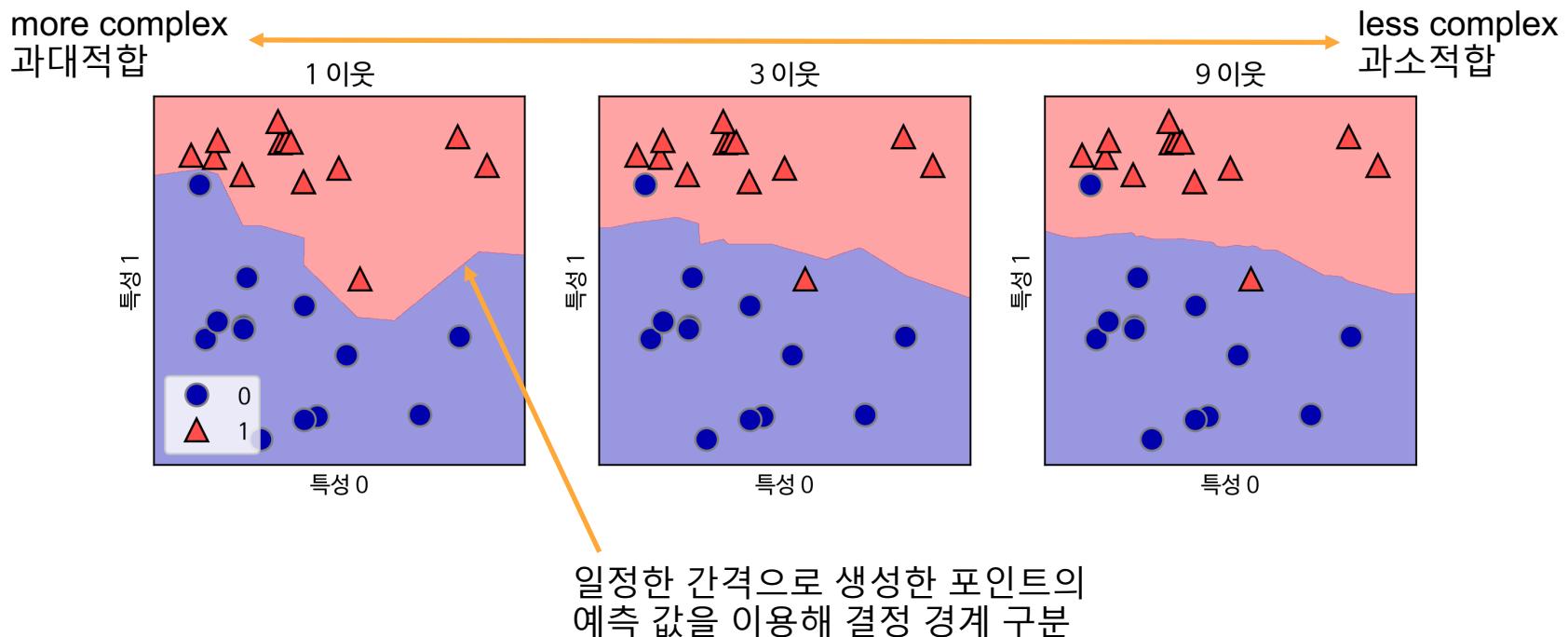
테스트 세트 예측: [1 0 1 0 1 0 0]

모델 평가

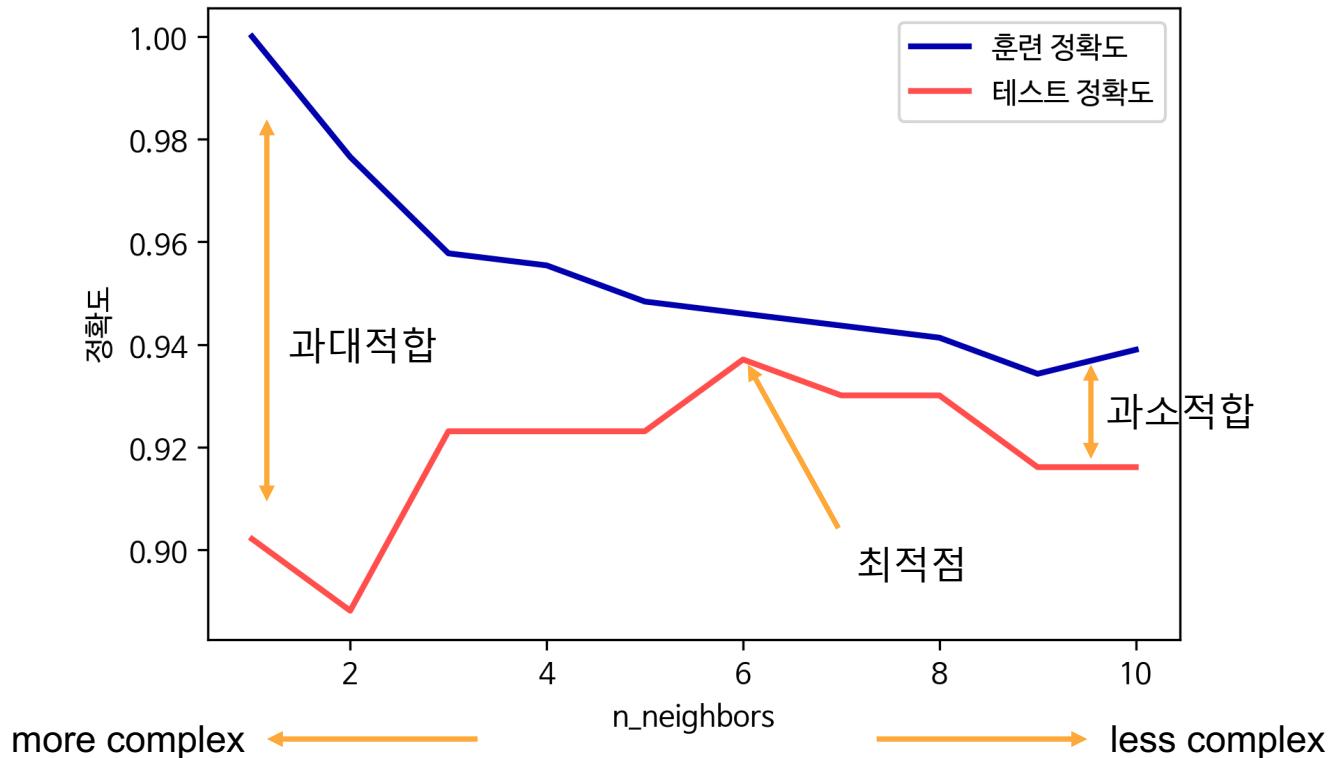
```
In [17]: print("테스트 세트 정확도: {:.2f}".format(clf.score(X_test, y_test)))
```

테스트 세트 정확도: 0.86

KNeighborsClassifier 분석



k-NN 분류기의 모델 복잡도

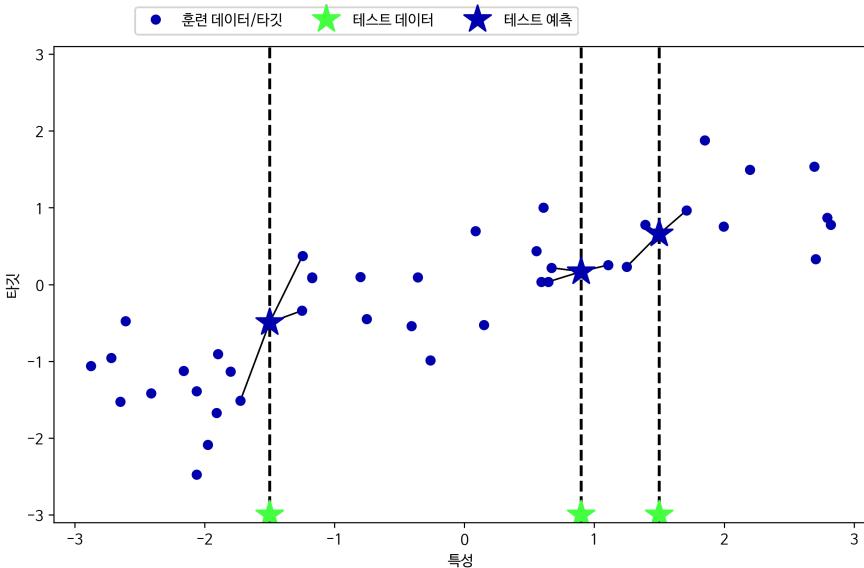
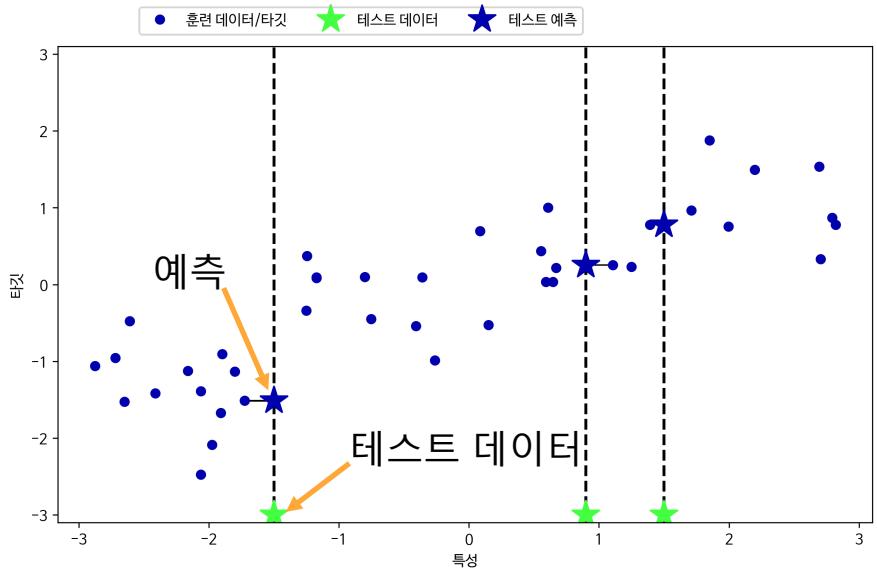


k-최근접 이웃 회귀

k-최근접 이웃 회귀

새로운 데이터 포인트에 가까운 이웃의 출력값 평균이 예측이 됩니다.

wave 데이터셋(1차원)에 1-최근접 이웃, 3-최근접 이웃 적용하면 다음과 같습니다.



k-NN 추정기

```
In [22]: from sklearn.neighbors import KNeighborsRegressor
```

훈련 세트와 테스트 세트로 분리

```
X, y = mglearn.datasets.make_wave(n_samples=40)
```

wave 데이터셋을 훈련 세트와 테스트 세트로 나눕니다

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

이웃의 수를 3으로 하여 모델의 객체를 만듭니다

```
reg = KNeighborsRegressor(n_neighbors=3)
```

후려 데이터와 타겟을 사용하여 모델을 학습시킵니다

```
reg.fit(X_train, y_train)
```

모델 객체 생성

모델 학습

```
Out[22]: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                               weights='uniform')
```

```
In [23]: print("테스트 세트 예측:\n{}".format(reg.predict(X_test)))
```

테스트 세트 예측:

```
[ -0.054   0.357   1.137  -1.894  -1.139  -1.631   0.357   0.912  -0.447  -1.139]
```

모델 평가

```
In [24]: print("테스트 세트 R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

테스트 세트 R²: 0.83

회귀 모델의 평가

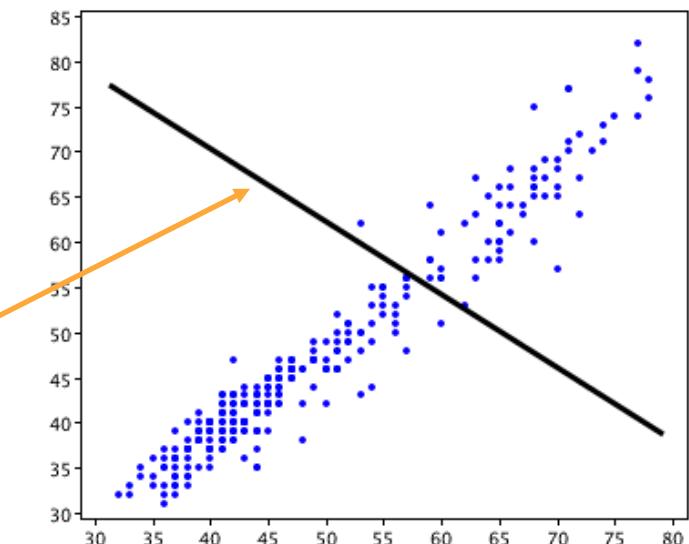
회귀 모델의 score() 함수는 결정 계수 R^2 를 반환

$$R^2 = 1 - \frac{\sum_{i=0}^n (y - \hat{y})^2}{\sum_{i=0}^n (y - \bar{y})^2} \quad y: \text{타깃값} \quad \bar{y}: \text{타깃값의 평균} \quad \hat{y}: \text{모델의 예측}$$

완벽 예측: 타깃값==예측 \rightarrow 문자==0, $R^2 = 1$

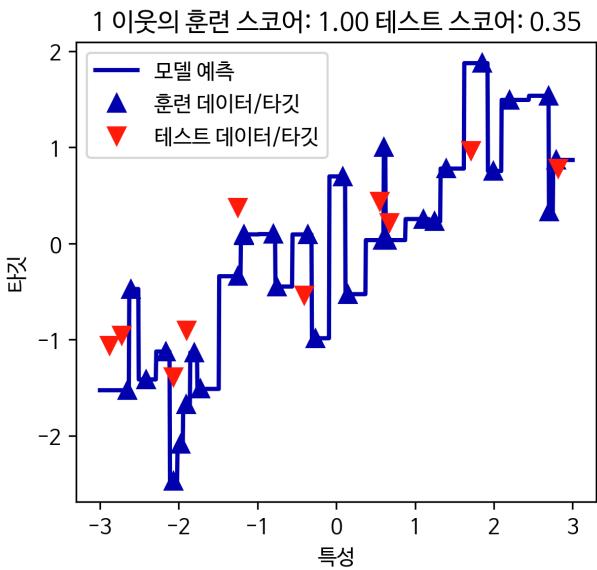
타깃값의 평균 정도 예측: 문자≈분모, $R^2 = 0$ 이 됨

평균 보다 나쁘게 예측하면 음수가 될 수 있음

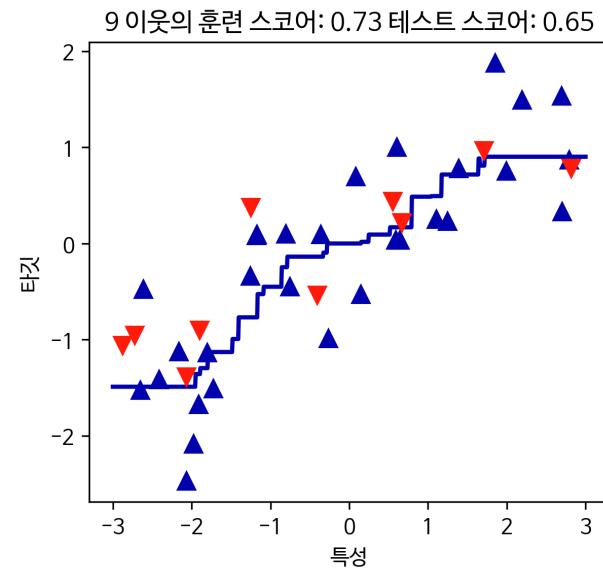
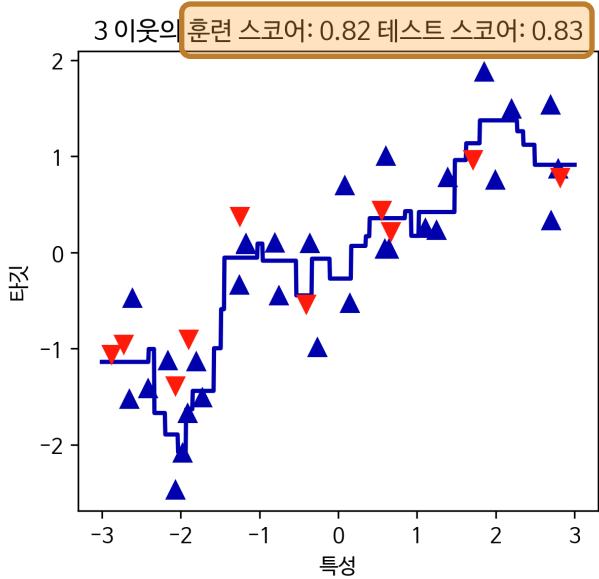


KNeighborsRegressor 분석

more complex
과대적합



less complex
과소적합



장단점과 매개변수

중요 매개변수

포인트 사이의 거리 측정 방법: metric 매개변수 기본값 ‘minkowski’이고
p 매개변수 기본값 2 일 때,

$$\text{유클리디안 거리 } \sqrt{\sum_{i=0}^m (x_1^{(i)} - x_2^{(i)})^2}$$

이웃의 수(n_neighbors): 3개나 5개(기본값)가 보편적

장점 이해하기 쉬움, 특별한 조정 없이 잘 작동, 처음 시도하는 모델로 적합,
 비교적 모델을 빠르게 만들 수 있음

단점 특성 개수나 샘플 개수가 크면 예측이 느림,
 데이터 전처리 중요(스케일이 작은 특성에 영향을 줄이지 않으려면 정규화 필요),
 (수백개 이상의) 많은 특성을 가진 데이터셋에는 잘 작동하지 않음,
 희소한 데이터셋에 잘 작동하지 않음

선형 모델 - 회귀

회귀의 선형 모델

선형 함수(linear model)를 사용하여 예측

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[p] \times x[p] + b$$
$$\hat{y} = w_0 \times x_0 + w_1 \times x_1 + \cdots + w_p \times x_p + b$$

특성 : $x[0] \sim x[p]$ 특성 개수: $(p + 1)$

모델 파라미터^{model parameter}: $w[0] \sim w[p], b$

ω : 가중치 weight, 계수 coefficient, θ, β

b : 절편 intercept, 편향 bias

e.g. `model.coef_`

e.g. `model.intercept_`

사용자가 지정한
매개변수와 구분

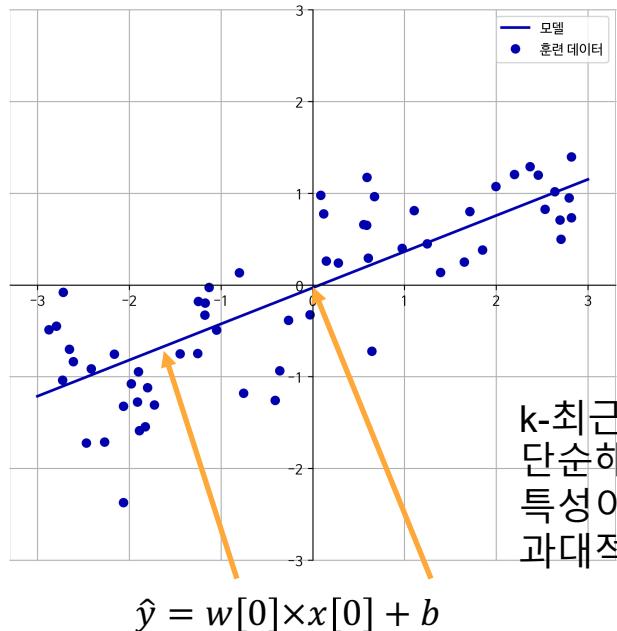
하이퍼파라미터^{hyperparameter}: 학습되지 않고 직접 설정해 주어야 함(매개변수)

e.g. KNeighborsRegressor의 `n_neighbors`

wave 데이터셋(특성 1개)으로 비교

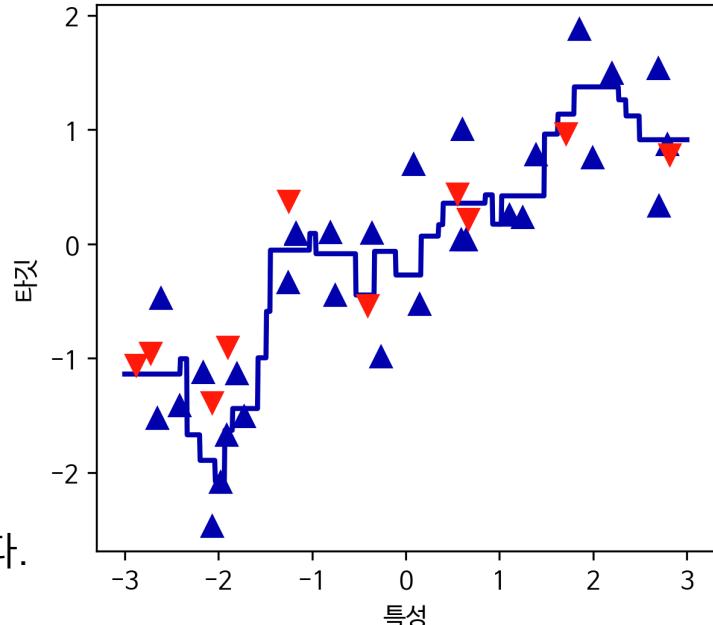
선형 모델은 특성이 두 개이면 평면, 세 개 이상은 초평면^{hyperplane}이 됩니다.

선형 모델



k-최근접 이웃에 비해
단순해 보이지만
특성이 많으면 오히려
과대적합 되기 쉽습니다.

k-최근접 이웃



최소제곱법 OLS, ordinary least squares

평균제곱오차 mean square error ($MSE = \frac{1}{n} \sum_{i=0}^n (y_i - \hat{y}_i)^2$)를 최소화

LinearRegression: 정규방정식 normal equation $\hat{\beta} = (X^T X)^{-1} X^T y$ 을 사용하여 w, b를 구함

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

lr = LinearRegression().fit(X_train, y_train)           ← 모델 객체 생성 & 학습

print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))

lr.coef_: [ 0.394]                                ← 가중치는 특성의 개수만큼
lr.intercept_: -0.031804343026759746

print("훈련 세트 점수: {:.2f}".format(lr.score(X_train, y_train)))
print("테스트 세트 점수: {:.2f}".format(lr.score(X_test, y_test)))    ← 모델 평가

훈련 세트 점수: 0.67
테스트 세트 점수: 0.66                            ← 과소적합(1차원 데이터셋이라 예상한 대로)
```

최소제곱법 OLS, ordinary least squares

보스턴 주택 가격 데이터셋, 506개 샘플, 104개 특성

동일한 기본 매개변수로 훈련

```
In [30]: X, y = mglearn.datasets.load_extended_boston()  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)  
lr = LinearRegression().fit(X_train, y_train)
```

```
In [31]: print("훈련 세트 점수: {:.2f}".format(lr.score(X_train, y_train)))  
print("테스트 세트 점수: {:.2f}".format(lr.score(X_test, y_test)))
```

훈련 세트 점수: 0.95
테스트 세트 점수: 0.61

훈련 세트와 테스트 세트의 R^2 점수 차이가 큼
특성이 많아 가중치가 풍부해져(104개의 차원) 과대적합 됨

릿지 ridge

선형 모델(MSE 최소화) + 가중치 최소화(가능한 0에 가깝게)

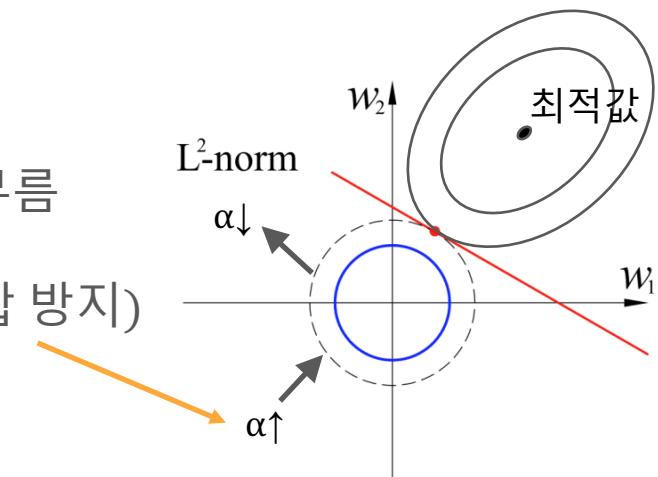
L2 규제 regularization : L2 노름^{norm}의 제곱 $\|w\|_2^2 = \sum_{j=1}^m w_j^2$

비용 함수 cost function : $MSE + \underbrace{\alpha \sum_{j=1}^m w_j^2}_{\text{페널티 penalty}}$

손실 함수 loss function, 목적 함수 objective function 라고도 부름

α 가 크면 페널티가 커져 w_j 가 작아져야 함(과대적합 방지)

w_j 가 0에 가깝게 되지만 0이 되지는 않음



Ridge 클래스

기본값 alpha=1.0

```
In [32]: from sklearn.linear_model import Ridge  
  
ridge = Ridge().fit(X_train, y_train)  
print("훈련 세트 점수: {:.2f}".format(ridge.score(X_train, y_train)))  
print("테스트 세트 점수: {:.2f}".format(ridge.score(X_test, y_test)))
```

훈련 세트 점수: 0.89
테스트 세트 점수: 0.75

(가중치가 규제되어) 과대적합이 줄고
테스트 세트 점수가 상승됨

최소제곱법

```
X, y = mglearn.datasets.  
X_train, X_test, y_train = LinearRegression()  
  
print("훈련 세트 점수: {:.2f}")  
print("테스트 세트 점수: {:.2f}")
```

훈련 세트 점수: 0.95
테스트 세트 점수: 0.61

```
In [33]: ridge10 = Ridge(alpha=10).fit(X_train, y_train)  
print("훈련 세트 점수: {:.2f}".format(ridge10.score(X_train, y_train)))  
print("테스트 세트 점수: {:.2f}".format(ridge10.score(X_test, y_test)))
```

훈련 세트 점수: 0.79
테스트 세트 점수: 0.64

제약이 너무 커짐
과소적합

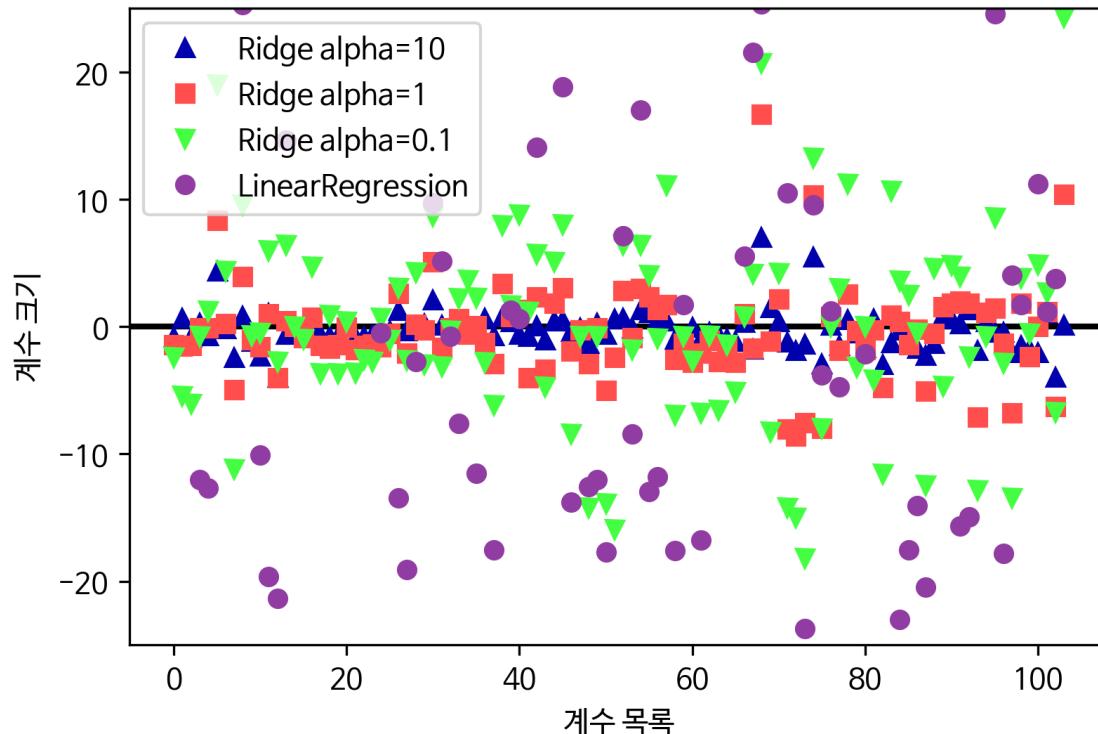
alpha=0.00001
로 하면 비슷해짐

```
In [34]: ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)  
print("훈련 세트 점수: {:.2f}".format(ridge01.score(X_train, y_train)))  
print("테스트 세트 점수: {:.2f}".format(ridge01.score(X_test, y_test)))
```

훈련 세트 점수: 0.93
테스트 세트 점수: 0.77

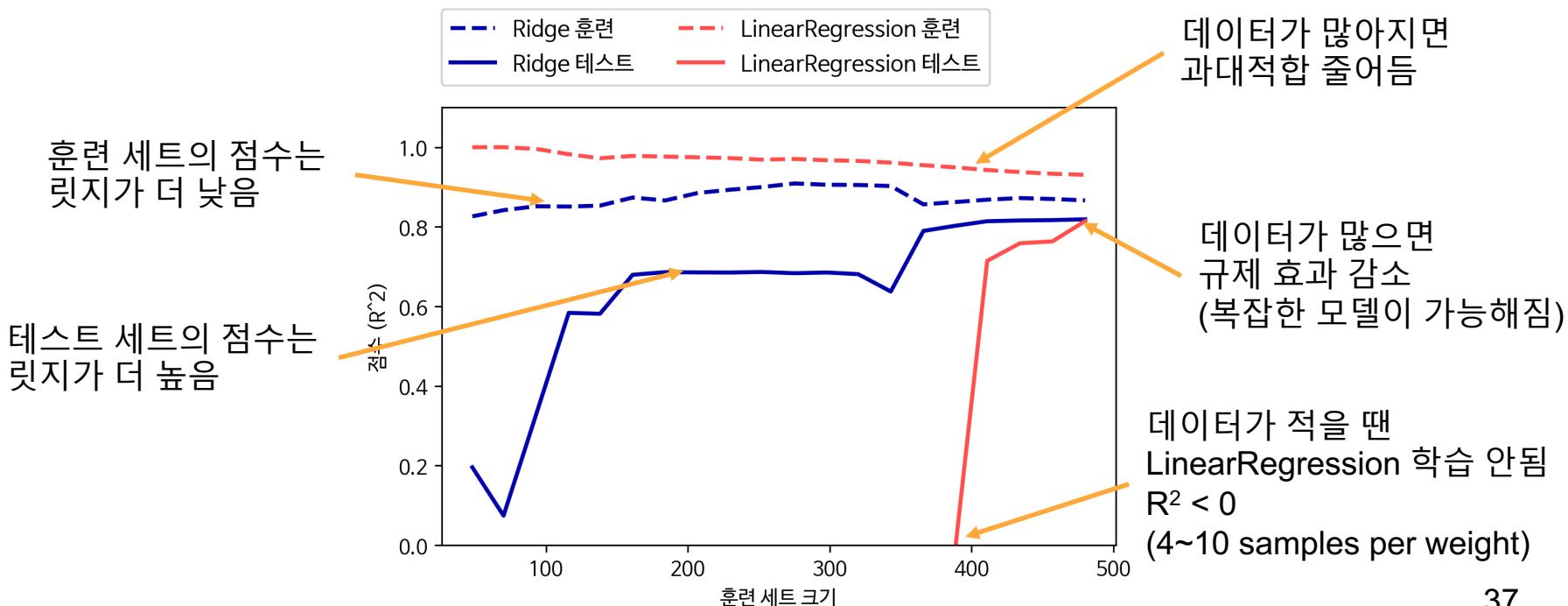
Ridge.coef_

alpha 값에 따른 coef_ 값의 변화



규제와 훈련 데이터의 관계

보스턴 주택가격 데이터셋의 학습곡선 : LinearRegression vs Ridge(alpha=1)



라쏘 Lasso

선형 모델(MSE 최소화) + 가중치 최소화(가능한 0으로)

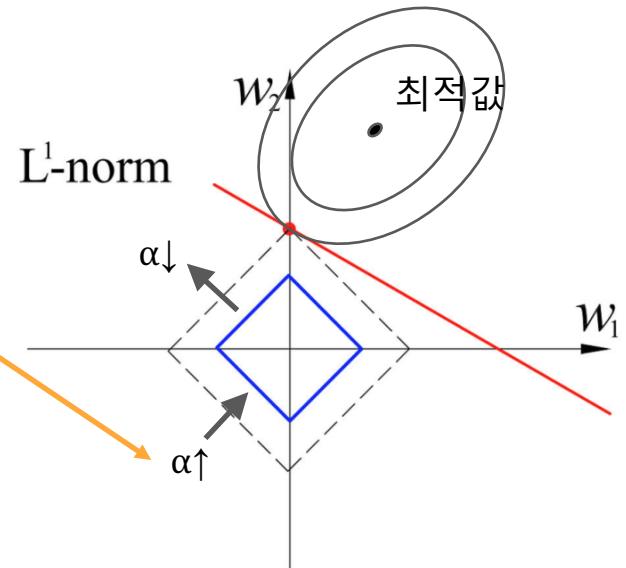
L1 규제 : L1 노름 $\|w\|_1 = \sum_{j=1}^m |w_j|$

비용 함수^{cost function} : $MSE + \alpha \sum_{j=1}^m |w_j|$

α 가 크면 페널티가 커져 w_j 가 더 작아져야 함

w_j 가 0이 될 수 있음(특성 선택의 효과)

일부 계수가 0이 되면 모델을 이해하기 쉽고 중요한 특성을 파악하기 쉽습니다.



Lasso

```
In [37]: from sklearn.linear_model import Lasso
```

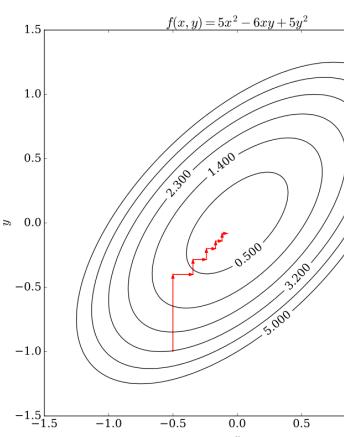
```
lasso = Lasso().fit(X_train, y_train)
print("훈련 세트 점수: {:.2f}".format(lasso.score(X_train, y_train)))
print("테스트 세트 점수: {:.2f}".format(lasso.score(X_test, y_test)))
print("사용한 특성의 개수: {}".format(np.sum(lasso.coef_ != 0)))
```

훈련 세트 점수: 0.29

테스트 세트 점수: 0.21

사용한 특성의 개수: 4

alpha=1.0, max_iter=1000



```
In [38]:
```

```
# "max_iter" 기본 값을 증가시키지 않으면 max_iter 값을 늘이라는 경고가 발생합니다
lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
print("훈련 세트 점수: {:.2f}".format(lasso001.score(X_train, y_train)))
print("테스트 세트 점수: {:.2f}".format(lasso001.score(X_test, y_test)))
print("사용한 특성의 개수: {}".format(np.sum(lasso001.coef_ != 0)))
```

훈련 세트 점수: 0.90

테스트 세트 점수: 0.77

사용한 특성의 개수: 33

과소적합(규제가 너무 큼), 4개의 특성만 활용됨

릿지와 비슷

```
In [39]:
```

```
lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print("훈련 세트 점수: {:.2f}".format(lasso00001.score(X_train, y_train)))
print("테스트 세트 점수: {:.2f}".format(lasso00001.score(X_test, y_test)))
print("사용한 특성의 개수: {}".format(np.sum(lasso00001.coef_ != 0)))
```

훈련 세트 점수: 0.95

테스트 세트 점수: 0.64

사용한 특성의 개수: 94

규제를 너무 낮추면 LinearRegression과 비슷

최소제곱법

```
x, y = mglearn.dataset
```

```
X_train, X_test, y_train = LinearRegression()
```

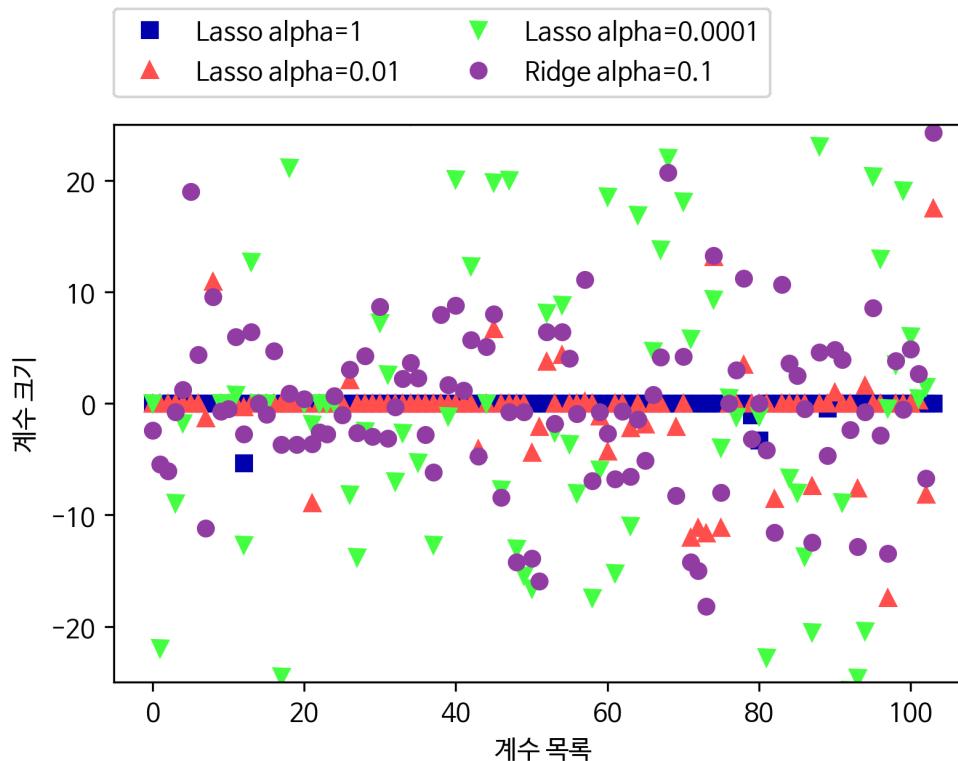
```
print("훈련 세트 점수: {:.2f}")
print("테스트 세트 점수: {:.2f}")
```

훈련 세트 점수: 0.95

테스트 세트 점수: 0.61

Lasso.coef_

alpha 값에 따른 coef_ 값의 변화



Ridge vs Lasso

일반적으로 릿지가 라쏘보다 선호됨

L2 페널티가 L1 페널티보다 선호됨 (SGD에서 강한 수렴)

많은 특성 중 일부만 중요하다고 판단되면 라쏘

분석하고 이해하기 쉬운 모델을 원할 때는 라쏘

ElasticNet

릿지와 라쏘 페널티 결합 (R의 glmnet)

alpha, l1_ratio 매개변수로 L1 규제와 L2 규제의 양을 조절

$$MSE + \alpha \times l1_ratio \sum_{j=1}^m |x_j| + \frac{1}{2} \alpha \times (1 - l1_ratio) \sum_{j=1}^m w_j^2$$

$$l_1 = \alpha \times l1_ratio \quad l_2 = \alpha \times (1 - l1_ratio)$$

$$\alpha = l_1 + l_2 \quad l1_ratio = \frac{l_1}{l_1 + l_2}$$

l₁과 l₂에 맞추어
α와 l1_ratio를 조절

사실 Lasso는 ElasticNet(l1_ratio=1.0)와 동일