빅데이터분석 실습

5주 지도학습 알고리즘 2

데이터 사이언스 전공
담당교수: 곽철완

강의 내용

- k-최근접 이웃 회귀분석
- 선형회귀모델
- 규제화
  - 리지 회귀

▪ 기본 라이브러리

◦ 분석을 위해 기본 라이브러리를 import한다

```
%matplotlib inline
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
import sklearn
```

k-최근접 이웃 회귀분석

▪ 목적

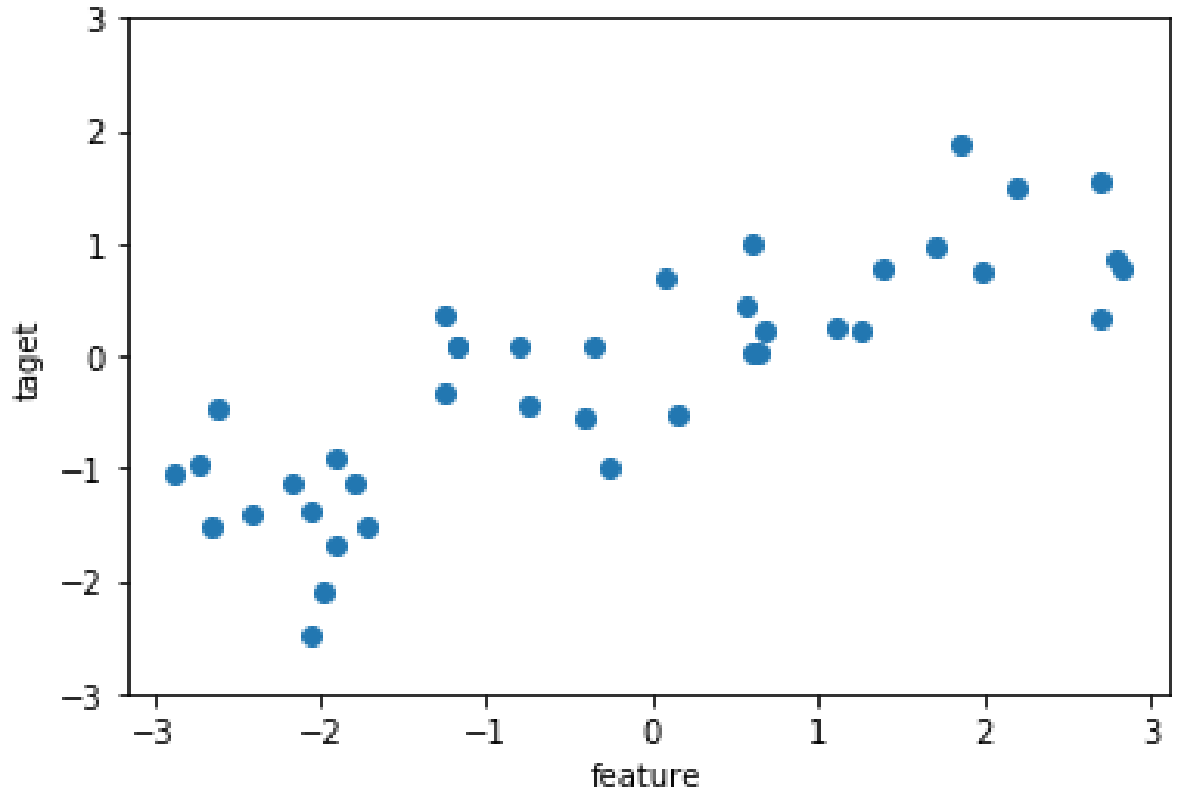
- 일반 회귀분석과 유사하게 인접 값을 통한 예측

▪ 분석할 데이터 세트 적재

- wave 데이터 세트 불러오기(mglearn에 포함)
- wave 데이터 세트 그림으로 보기

```
X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, ‘o’)
plt.ylim(-3, 3)
plt.xlabel(“feature”)
plt.ylabel(“taget”)
plt.show()
```

```
X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("feature")
plt.ylabel("taget")
plt.show()
```

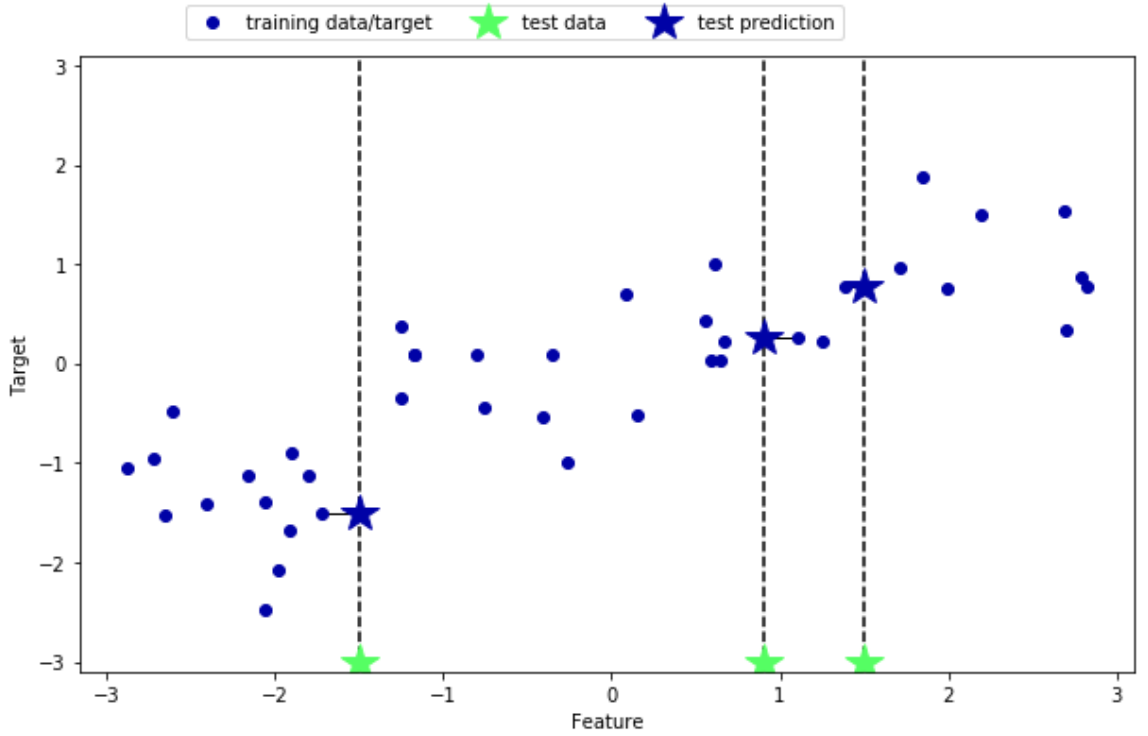


■ 테스트 데이터 예측

◦ ★ 테스트 데이터, ★ 예측 (이웃값: 1)

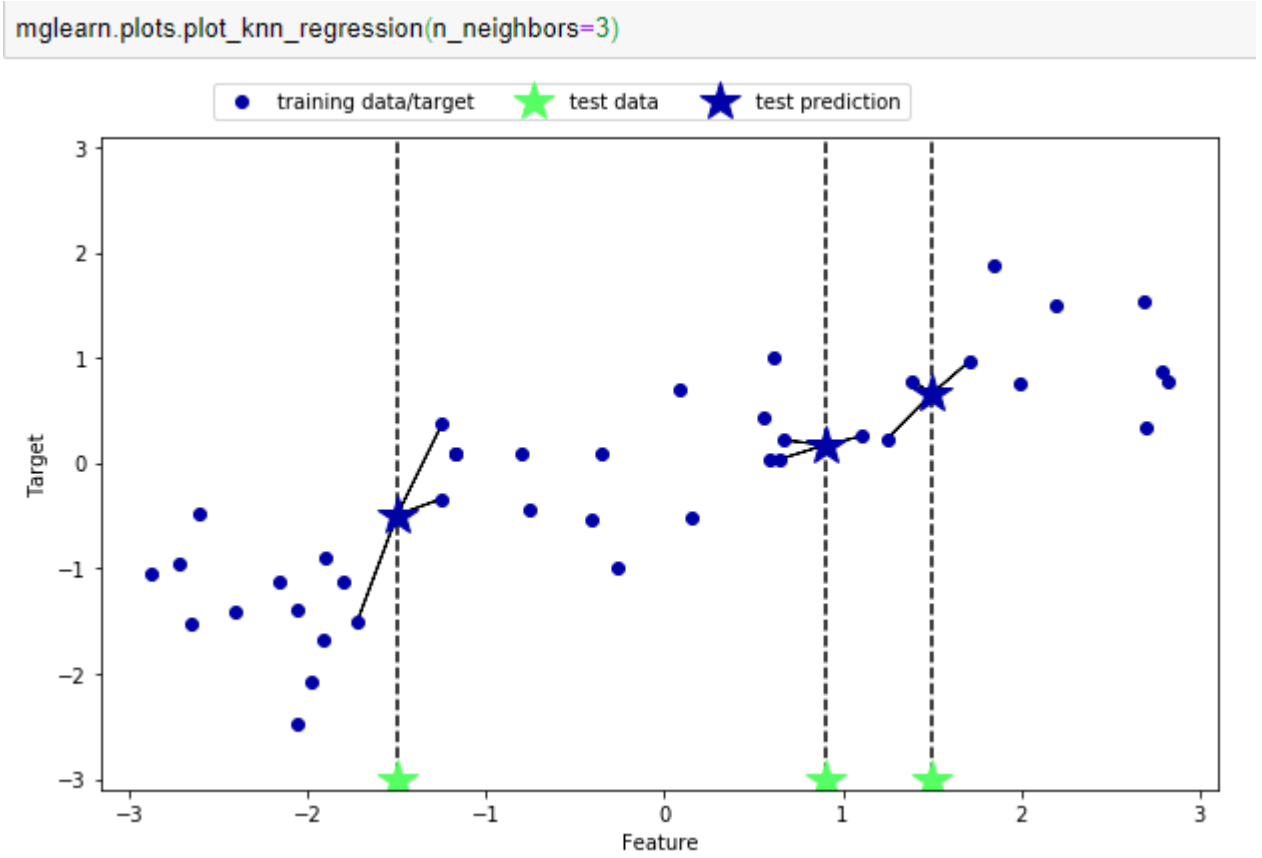
```
mglearn.plots.plot_knn_regression(n_neighbors=1)
```

◦ ★ 표시 값이 예측 값이다



mglearn.plots.plot_knn_regression(n_neighbors=3)

- 이웃값이 3 이 되었을 때,
- ★ 예측 값의 위치는 바뀌었다



▪ 사이킷런의 최근접 이웃 알고리즘

- 학습용 데이터 세트와 평가용 데이터 세트로 구분
- sklearn.model_selection 에서 'train_test_split'를 import한다

```
from sklearn.model_selection import train_test_split
```

- KNeighborsRegressor 클래스는 neighbors 모듈에 포함되어 있다
- 학습용 데이터 세트와 평가용 데이터 세트로 구분한다
- 이웃 값을 3으로 지정
- reg.fit(X_train, y_train):학습 실시

```
from sklearn.neighbors import KNeighborsRegressor

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
reg=KNeighborsRegressor(n_neighbors=3)
reg.fit(X_train, y_train)
```

```
from sklearn.neighbors import KNeighborsRegressor
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
reg=KNeighborsRegressor(n_neighbors=3)
reg.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=3, p=2,
          weights='uniform')
```

◦ 평가용 데이터 세트 예측

```
print("평가용 데이터 세트 예측:\n"), reg.predict(X_test))
```

```
print("평가용 데이터 세트 예측:\n", reg.predict(X_test))
```

```
평가용 데이터 세트 예측:
 [-0.05396539  0.35686046  1.13671923 -1.89415682 -1.13881398 -1.63113382
  0.35686046  0.91241374 -0.44680446 -1.13881398]
```

◦ 예측 값이 잘 맞는지 비교해야 한다(학습용 데이터와 평가용 데이터를 비교)

◦ score 메서드 사용하여 모델 reg 개체에 평가용 데이터 세트를 삽입하여 평가한다

print(“평가용 데이터 세트 R^2: {:.2f}” .format(reg.score(X_test, y_test)))

```
print("평가용 데이터 세트 R^2: {:.2f}" .format(reg.score(X_test, y_test)))
```

```
평가용 데이터 세트 R^2: 0.83
```

$$r^2 = 1 - \frac{\sum_{i=1}^{n}(y - \hat{y})^2}{\sum_{i=1}^{n}(y - \bar{y})^2}$$

- 결정 계수 R^2 값은 회귀모델에서 예측의 적합도 측정이다
- 결정계수 = 1- ∑(타깃값 – 모델의 예측값)² / ∑(타깃값 – 타깃값의 평균값)²

```
print("평가용 데이터 세트 R^2: {:.2f}" .format(reg.score(X_test, y_test)))
```

```
평가용 데이터 세트 R^2: 0.83
```

- 결정계수: 0.83

▪ KNeighborsRegressor 분석

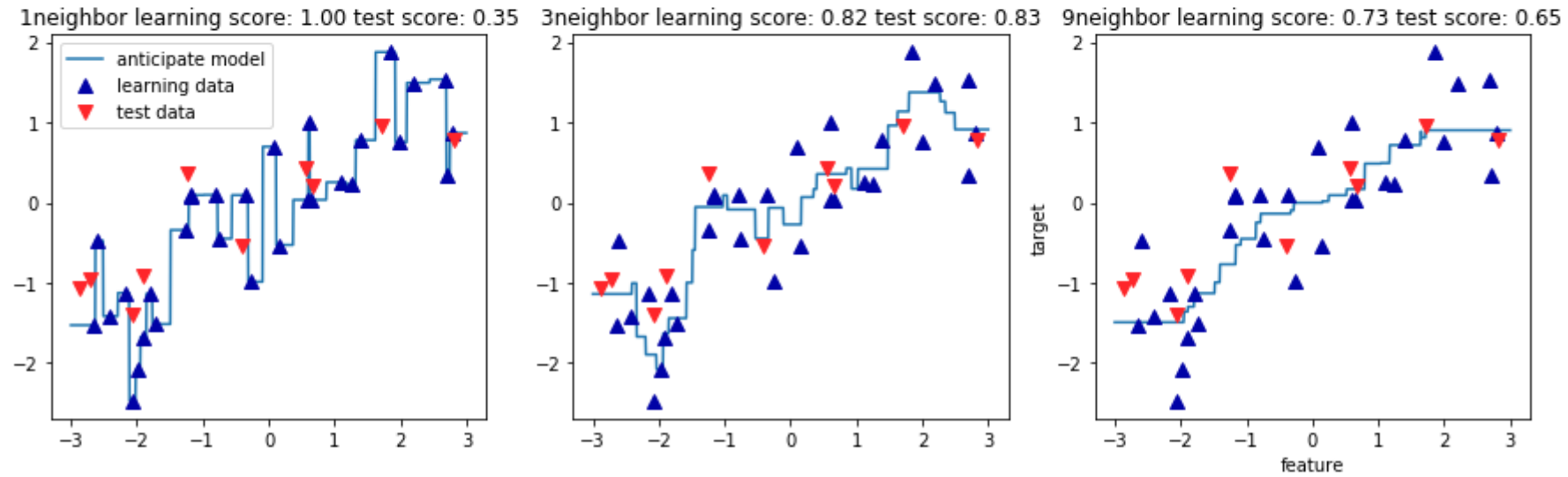
- 평가용 데이터 세트 예측
  - 이웃 값에 따른 예측 비교

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes)
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0),markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
```

```
  ax.set_title(
        "{} neighbor learning score: {:.2f} test scire: {:.2f}".format(
        n_neighbors, reg.score(X_train, y_train),
        reg.score(X_test, y_test)))
ax.set_xlabel("feature")
ax.set_ylabel("target")
axes[0].legend(["anticipate model", "learning data", "test data"], loc="best")
```

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0),markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
    ax.set_title(
    "{}neighbor learning score: {:.2f} test score: {:.2f}".format(
        n_neighbors, reg.score(X_train, y_train),
        reg.score(X_test, y_test)))
ax.set_xlabel("feature")
ax.set_ylabel("target")
axes[0].legend(["anticipate model", "learning data", "test data"], loc="best")
```

```
<matplotlib.legend.Legend at 0x1c8106bc940>
```



■ 장단점과 매개변수

- 매개변수
  - 이웃의 수 결정이 중요한 영향을 미침
  - 거리 측정에는 Euclidean Distance(유클리디안 거리) 적용
- 장점
  - 이해하기 쉬운 모델로, 복잡한 모델을 적용하기 전에 시범적으로 사용한다
- 단점
  - 사전에 전처리 과정이 중요하다
  - 피처가 많은 데이터 세트에서는 잘 작동하지 않는다
  - 속도가 느려 실제로는 잘 사용하지 않는다

선형 회귀 모델

- 일반화된 예측 함수

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \ldots + w[p] \times x[p] + b$$

  - x[0] ~ x[p]: 피처
  - w(기울기)와 b(절편): 모델이 학습할 파라미터
  - ŷ : 모델이 만들어 낸 예측 값
    - 피처가 하나면 ŷ = w[0] + x[0] + b 이다 (x: 피처)

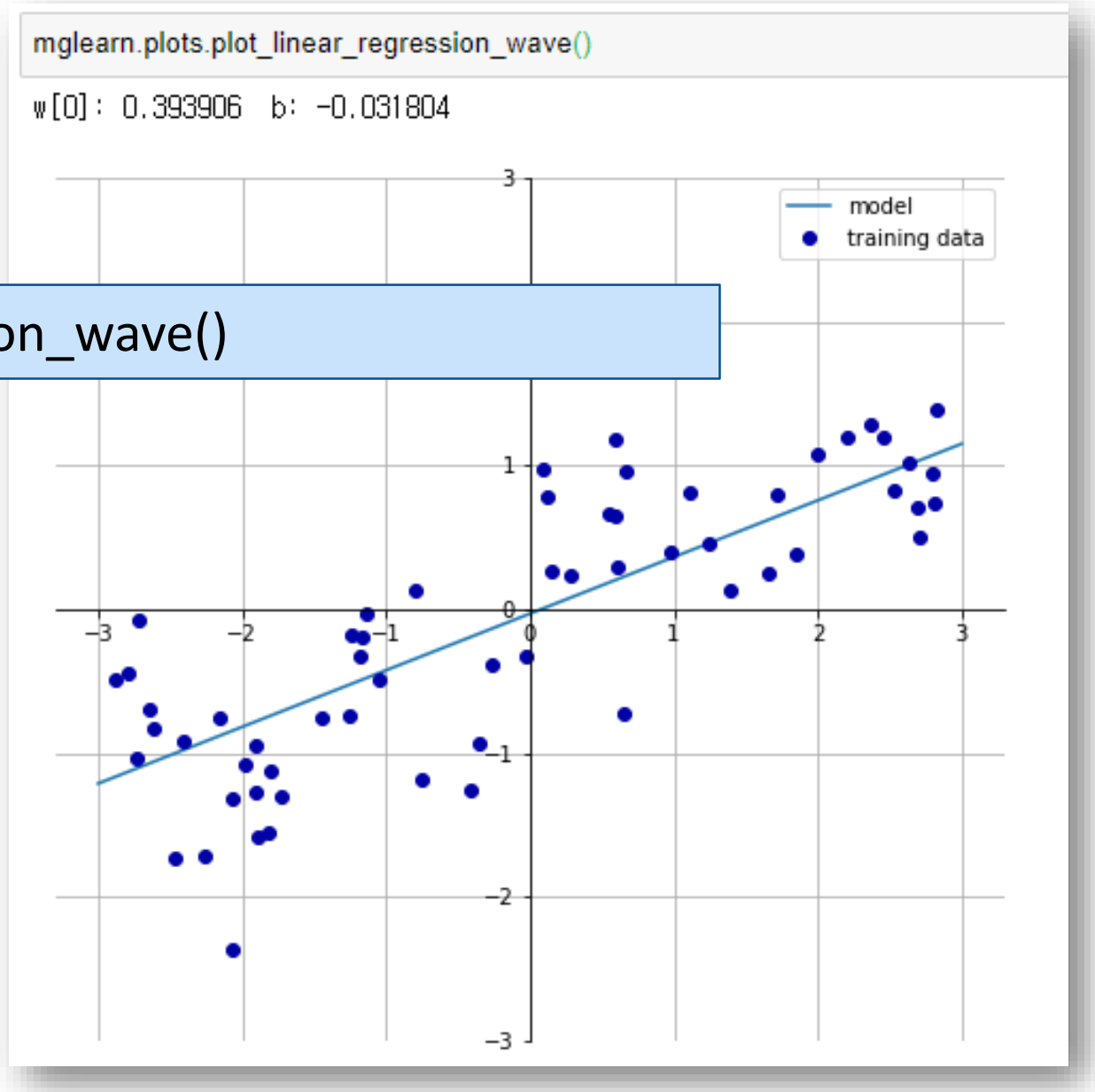
- 사용하는 데이터 세트
  - wave 데이터 세트

- 선형 회귀선 그리기

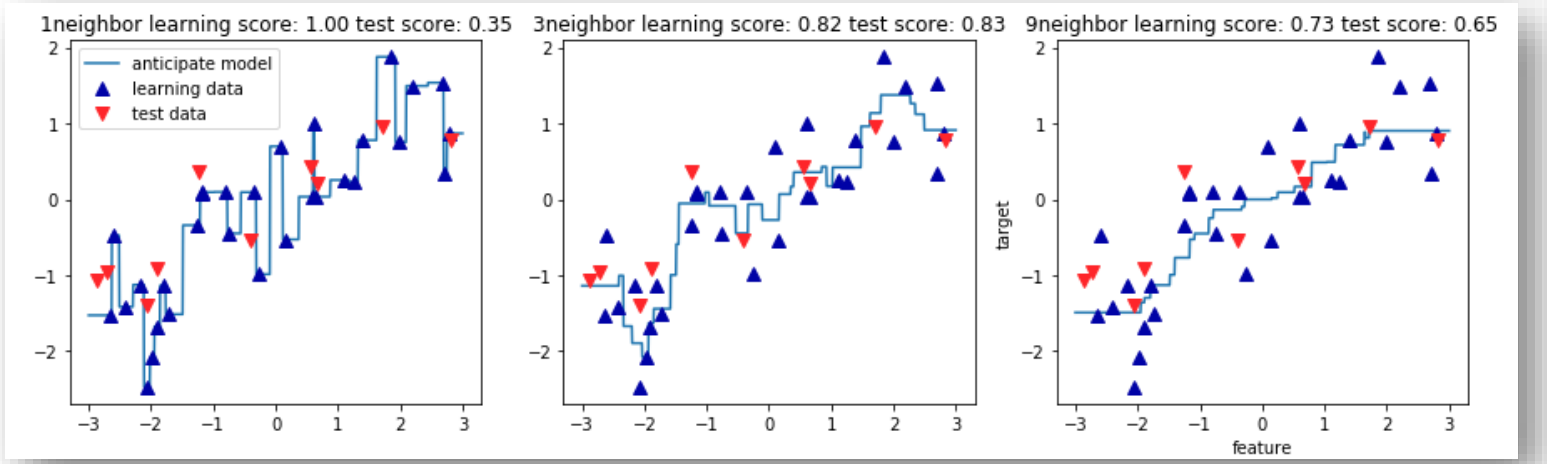
```
mglearn.plots.plot_linear_regression_wave()
```

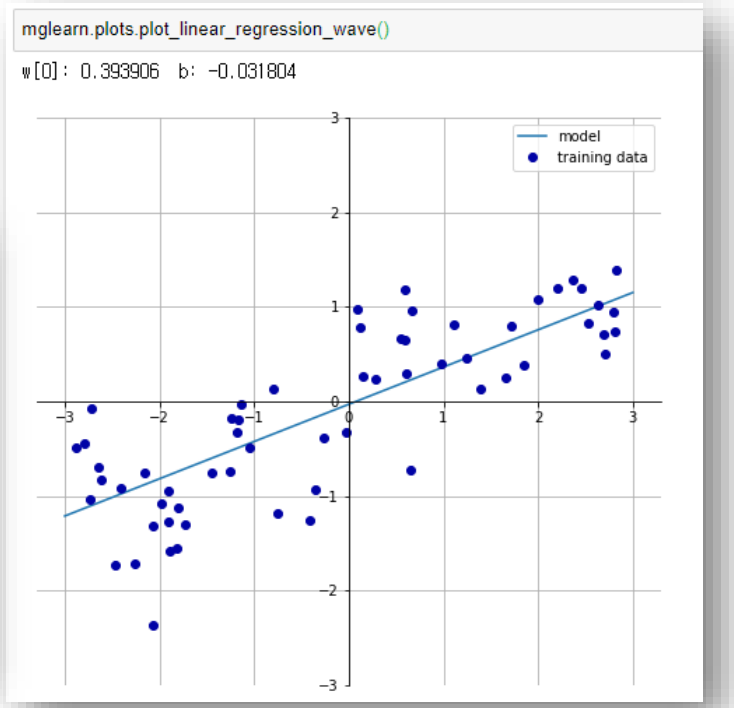
  - w[0]: 0.3939
  - b : -0.0318

- 선형 회귀 모델은 피처가 하나이면 직선
- 피처가 2개면 평면
- 더 많으면 초평면이 된다



- 최근접 이웃 모델과 선형 모델 비교

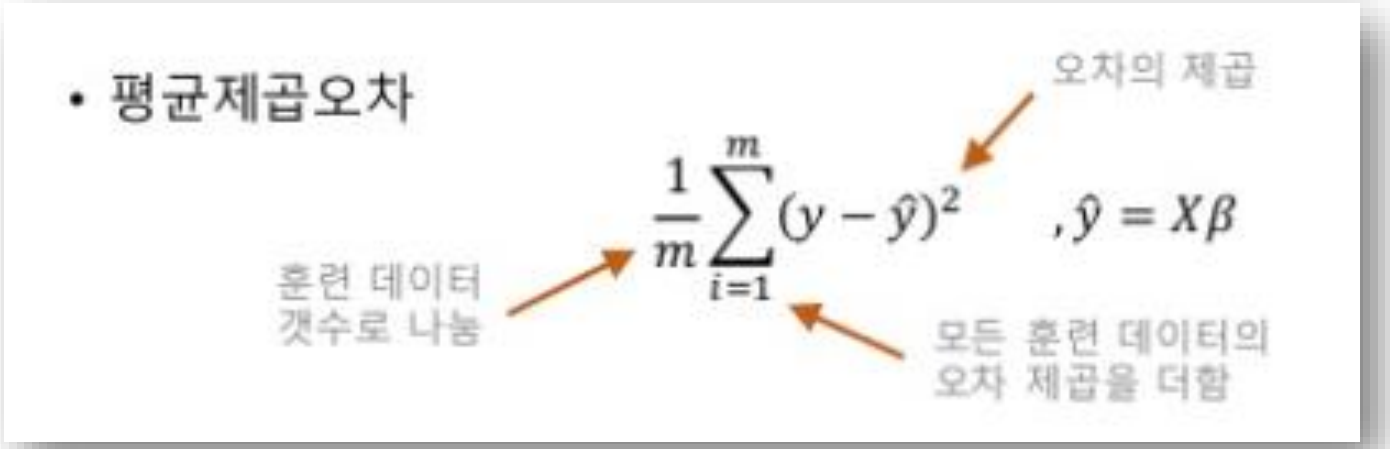




  - 최근접 이웃 모델이 보다 정확하게 보일 수 있으나, 피처가 많은 데이터 세트에서는 선형 모델이 더 훌륭한 성능을 보일 수 있다

▪ 선형 회귀(최소제곱법)

- 특징
  - 가장 간단하고, 오래된 선형 알고리즘이다
  - 학습 데이터 세트에 있는 타깃 y와 예측 사이의 평균제곱오차(mean squared error)를 최소화하는 w(기울기) 값과 b(절편) 값을 찾는다
    - 평균제곱오차: 타깃 값과 예측 값의 차이를 제곱하여 더한 후에 학습용 데이터의 개수로 나눈 것
  - 매개변수가 없다



- 모델 만들기

```
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)
```

  - 기울기(w)는 회귀 계수(coefficient)라 하며, lr 객체의 coef_ 속성에 저장된다
  - 절편(w)은 intercept_ 속성에 저장된다

```
print(“lr.coef_:”, lr.coef_)
print(“lr.intercept_:”, lr.intercept_)
```

- lr.coef_ 와 lr.intercept_ 처럼 단어 끝에 밑줄은 scikit-learn 학습용 데이터에서 유도된 속성에서 붙이는 것으로 사용자가 지정한 매개변수와 구분하기 위함이다

```
print("lr.coef_:", lr.coef_)
print("lr.intercept_:", lr.intercept_)
```

```
lr.coef_: [0.39390555]
lr.intercept_: -0.031804343026759746
```

- intercept_ 속성은 실수이지만, coef_ 속성은 NumPy 배열이다

- 학습용 데이터 세트와 평가용 데이터 세트 비교

```
print(“학습용 데이터 세트 점수: {:.2f}” .format(lr.score(X_train, y_train)))
print(“평가용 데이터 세트 점수: {: .2f}” .format(lr.score(X_test, y_test)))
```

```
print("학습용 데이터 세트 점수: {:.2f}" .format(lr.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}" .format(lr.score(X_test, y_test)))
```

```
학습용 데이터 세트 점수: 0.67
평가용 데이터 세트 점수: 0.66
```

- R^2 값이 0.66이며, 학습용 데이터 세트 점수와 비슷하다 → 과소적합
- 일반적으로 1차원 데이터 세트 모델은 단순하므로 과대적합일 경우가 적다
- 피처가 많은 경우에는 과대적합(overfitting)일 가능성이 높다 → 피처가 많아 복잡도를 제어(피처 수를 줄이는)할 수 있는 모델 필요

▪ 리지 회귀

- 특징
  - 선형 회귀분석에서 피처 수가 많아 과대적합(overfitting)을 방지하기 위해
  - 각 피처의 계수(w)를 가능한 한 작게 만든다
  - 즉, 모든 피처의 회귀계수를 작게 하여 각 피처가 주는 영향력(다중공선성 방지)을 최소화한다
  - 이를 규제(regularization, 혹은 정규화)라 한다
    - 규제란 과대적합이 되지 않도록 모델을 강제로 제한하는 것을 말한다(L2 규제)

- 사용 데이터 세트
  - Boston Housing 데이터 세트
  - 1970년대 미국 보스톤 주변의 주택 평균 가격 예측
  - 피처: 범죄율, 찰스강 인접도, 고속도로 접근성 등 13개

```
from sklearn.datasets import load_boston
boston = load_boston()
print("데이터 세트의 형태:", boston.data.shape)
```

```
# boston housing 데이터 세트
from sklearn.datasets import load_boston
boston = load_boston()
print("데이터 세트의 형태:", boston.data.shape)
```

```
데이터 세트의 형태: (506, 13)
```

- 피처 공학(feature engineering)을 사용하여 피처 간의 곱(예, 범죄율과 고속도로 접근성을 곱하여 피처로 만듦)을 이용하여 유도
  - load_extended_boston 함수 사용(13개 → 104개)
- 학습용 데이터 세트와 평가용 데이터 세트로 구분

```
X, y = mglearn.datasets.load_extended_boston( )
print("X.shape:", X.shape)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
X, y = mglearn.datasets.load_extended_boston()

print("X.shape:", X.shape)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
X.shape: (506, 104)
```

- 선형 모델 분석
  - 리지 회귀 분석과 비교하기 위해 선형 모델 적용

```
lr = LinearResgression().fit(X_train, y_train)


print("학습용 데이터 세트 점수: {:.2f}" .format(lr.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}" .format(lr.score(X_test, y_test)))
```

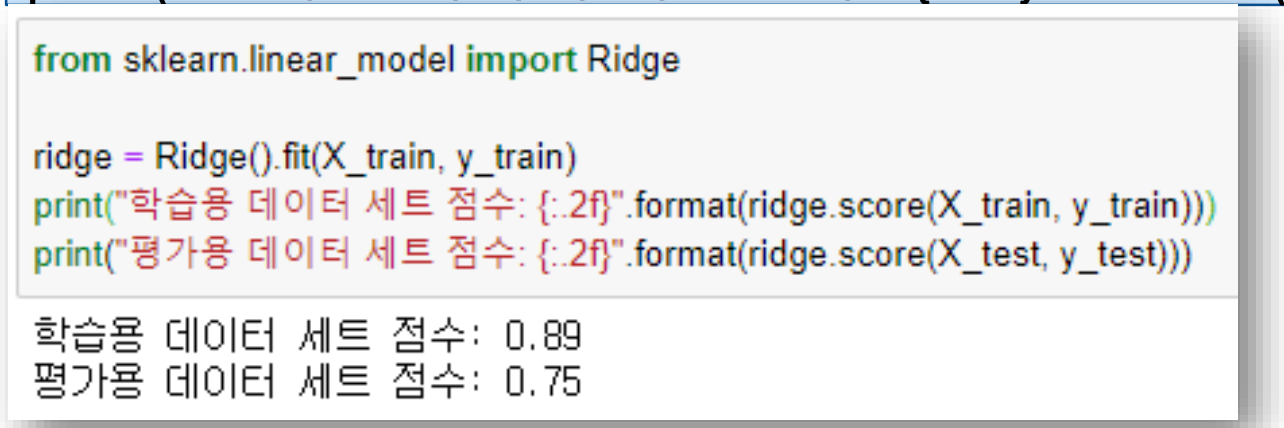
```
lr = LinearRegression().fit(X_train, y_train)

print("학습용 데이터 세트 점수: {:.2f}".format(lr.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}".format(lr.score(X_test, y_test)))
```

```
학습용 데이터 세트 점수: 0.95
평가용 데이터 세트 점수: 0.61
```

  - 학습용 데이터 세트 점수와 평가용 데이터 세트 점수 차이가 큰 것은 과대적합 신호이다

- 리지 회귀 분석

```
from sklearn.linear_model import Ridge
ridge = Ridge( ).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}" .format(ridge.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}" .format(ridge.score(X_test, y_test)))
```



  - 학습용은 선형 모델보다 낮지만(0.95 -> 0.89), 평가용은 더 높다(0.61 -> 0.75)
  - 평가용 데이터 세트 점수가 높아야 되기 때문에 리지(Ridge)를 선택한다

- alpha 값
  - 리지 회귀 모델은 피처를 단순하게 하면서 학습용 데이터 세트의 성능을 조절할 수 있다
  - alpha 값을 매개 변수로 학습용 데이터 세트의 성능을 기준으로 모델의 단순화 정도를 지정할 수 있다
  - alpha 값을 높이면 회귀 계수를 0에 가깝게 만들어 학습용 데이터 세트의 성능은 낮아지지만 일반화를 증가시킬 수 있다
  - 반대로, 아주 작은 alpha 값은 회귀 계수를 거의 제한하지 않아 선형 회귀로 만든 모델과 거의 같아 진다

- alpha 값 조정 결과 모델 성능

```
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}" .format(ridge10.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}" .format(ridge10.score(X_test, y_test)))
```

```
# 알파값을 10으로 조정
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}" .format(ridge10.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}" .format(ridge10.score(X_test, y_test)))
```

```
학습용 데이터 세트 점수: 0.79
평가용 데이터 세트 점수: 0.64
```

◦ alpha 값 0.1

```
ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}" .format(ridge01.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}" .format(ridge01.score(X_test, y_test)))
```

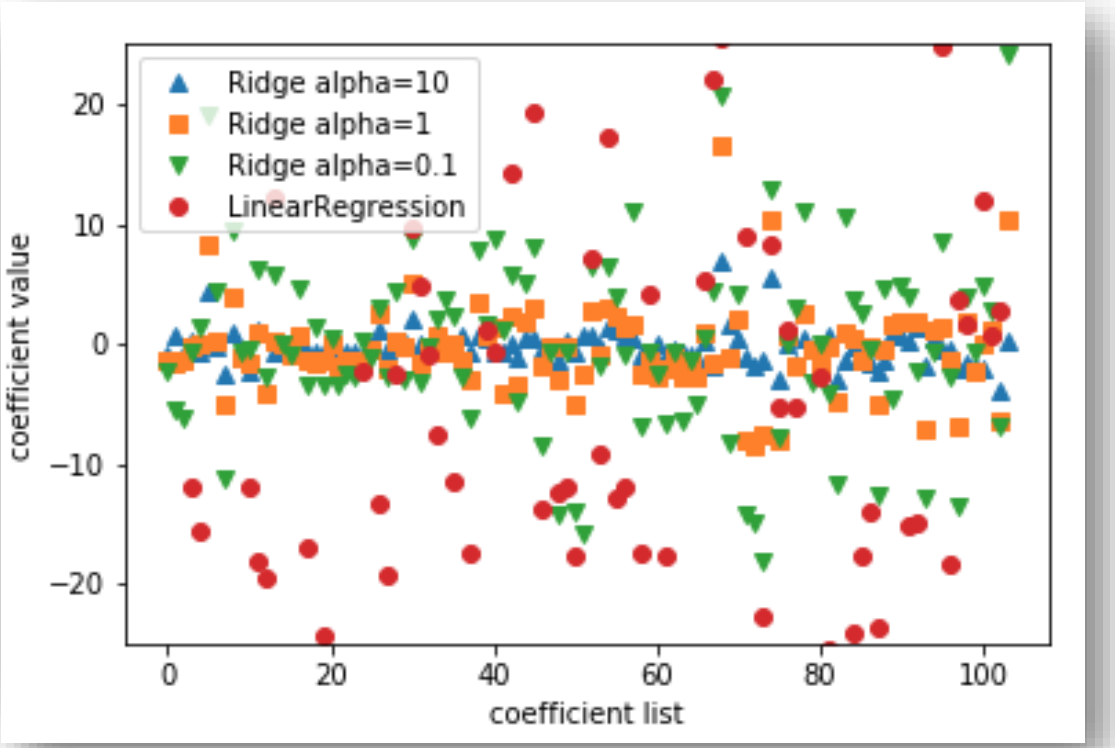
```
학습용 데이터 세트 점수: 0.93
평가용 데이터 세트 점수: 0.77
```

◦ 학습용 및 평가용 성능이 향상되었다(다른 데이터 세트를 사용하면 달라 질 수 있다)

◦ 선형 모델과 alpha 값에 다른 회귀 계수 변화

```
plt.plot(ridge10.coef_, '^', label="Ridge alpha=10")
plt.plot(ridge.coef_, 's', label="Ridge alpha=1")
plt.plot(ridge01.coef_, 'v', label="Ridge alpha=0.1")
plt.plot(lr.coef_, 'o', label="LinearRegression")
plt.xlabel("coefficient list")
plt.ylabel("coefficient value")
xlims = plt.xlim( )
plt.xlim(xlims)
plt.ylim(-25, 25)
plt.legend( )
```

```
plt.plot(ridge10.coef_, '^', label="Ridge alpha=10")
plt.plot(ridge.coef_, 's', label="Ridge alpha=1")
plt.plot(ridge01.coef_, 'v', label="Ridge alpha=0.1")
plt.plot(lr.coef_, 'o', label="LinearRegression")
plt.xlabel("coefficient list")
plt.ylabel("coefficient value")
xlims = plt.xlim()
plt.xlim(xlims)
plt.ylim(-25, 25)
plt.legend( )
```



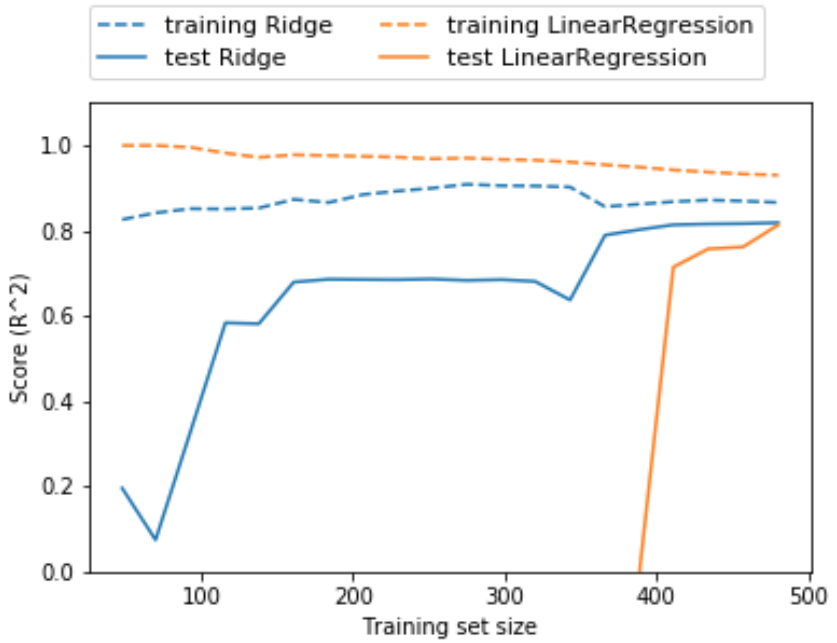
- x 축은 피처(0~104가지) 순으로 나열
- y 축은 회귀 계수
  - alpha = 10 일 때 회귀 계수는 -3에서 3 사이
  - alpha = 0.1 일 때 회귀 계수는 더 넓게 분산된다

- 데이터 세트 크기에 따른 모델 성능 변화
  - 규제 효과를 이해하기 위해 alpha 값을 고정하고 학습용 데이터 세트 크기를 변화시켰을 때, 모델의 성능 변화 → 학습 곡선 learning curve

```
mglearn.plots.plot_ridge_n_samples( )
```

  - 결과 해석
    - 데이터 세트 크기가 충분히 크면 리지와 선형 회귀 성능이 같아질 것이다
    - 데이터 세트 크기가 커질 수록 선형 회귀 학습용 성능이 감소한다

```
mglearn.plots.plot_ridge_n_samples()
```



요약

- k- 최근접 이웃 회귀분석
- 선형 회귀모델
- 리지 회귀

다음 시간

- 라소 회귀
- 분류용 선형 모델
- 나이브 베이즈 분류기