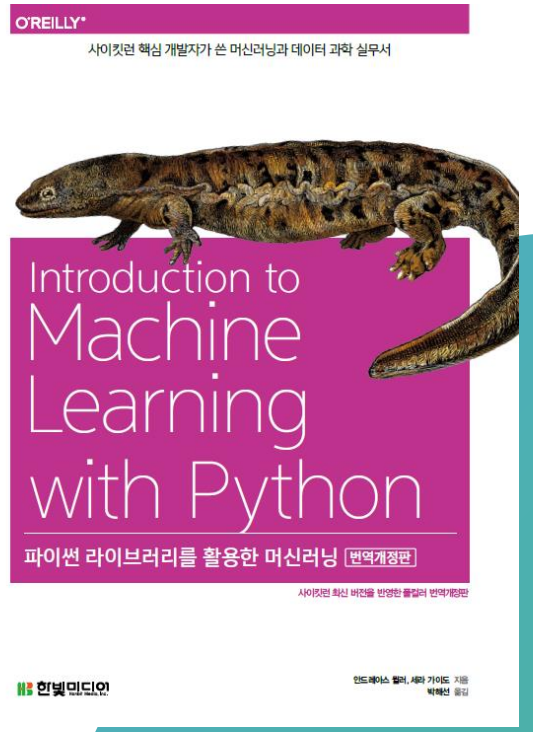
Chapter 02 : 지도 학습

파이썬라이브러리를 활용한머신러닝 (개정판)



시작하기전에

- **책에서 사용하는 소프트웨어 버전**
  - Python 버전: 3.7.2 (default, Dec 29 2018, 06:19:36)
  - [GCC 7.3.0]
  - pandas 버전: 0.23.4
  - matplotlib 버전: 3.0.2
  - NumPy 버전: 1.15.4
  - SciPy 버전: 1.1.0
  - IPython 버전: 7.2.0
  - scikit-learn 버전: 0.20.2
- **예제 다운로드 링크**
  - https://github.com/rickiepark/introduction_to_ml_with_python
  - https://nbviewer.jupyter.org/github/rickiepark/introduction_to_ml_with_python/tree/master/

이 책의 학습 목표

- 1장: 머신러닝과 머신러닝 애플리케이션의 기초 개념을 소개 및 사용 환경
- 2장: 지도 학습 알고리즘
- 3장: 비지도 학습 알고리즘
- 4장: 머신러닝에서 데이터를 표현하는 방법
- 5장: 모델 평가와 매개변수 튜닝을 위한 교차 검증과 그리드 서치
- 6장: 모델을 연결하고 워크플로를 캡슐화하는 파이프라인 개념
- 7장: 텍스트 데이터에 적용하는 방법과 텍스트에 특화된 처리 기법
- 8장: 개괄적인 정리와 어려운 주제에 대한 참고 자료 안내

Contents

CHAPTER 02 지도 학습

CHAPTER 02 지도 학습

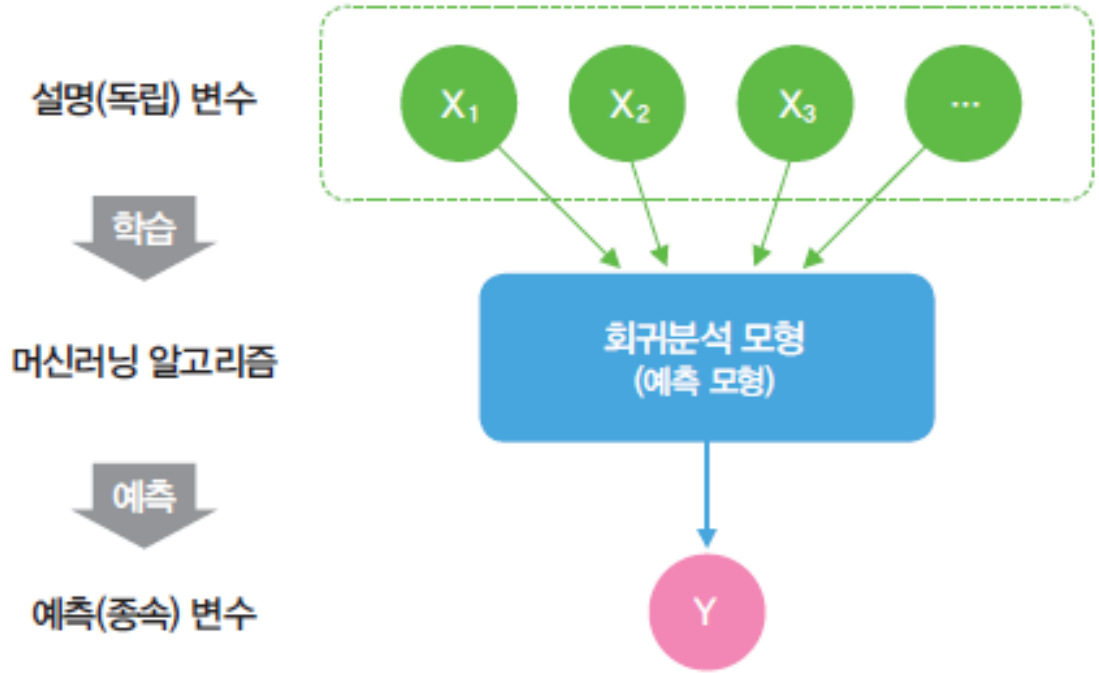
머신러닝의 지도 학습 알고리즘

SECTION 2.1 분류와 회귀

- 지도 학습에는 분류(classification)와 회귀(regression)가 있음
  - 분류: 분류는 미리 정의된, 가능성 있는 여러 **클래스 레이블**class label 중 하나를 예측하는 것
    - 딱 두 개의 클래스로 분류하는 **이진 분류**binary classification
      - 이진 분류는 질문의 답이 예/아니오만 나올 수 있도록 하는 것
      - 이메일에서 스팸을 분류하는 것이 이진 분류 문제 → 예/아니오 대답에 대한 질문은 "이 이메일이 스팸인가요?"
      - *이진 분류에서 한 클래스를 **양성**positive 클래스(좋은 값이나 장점을 나타내는 것이 아니라 학습하고자 하는 대상)*
        *다른 하나를 **음성**negative 클래스*
    - 셋 이상의 클래스로 분류하는 **다중 분류**multiclass classification
      - 붓꽃의 분류(붓꽃이 종류가 여러개), 웹사이트의 글로부터 어떤 언어의 웹사이트인지를 예측 (한국어 or 영어 or 일본어 등)

SECTION 2.1 분류와 회귀

- 지도 학습에는 분류(classification)와 회귀(regression)가 있음
  - 회귀: 연속적인 숫자, 또는 프로그래밍 용어로 말하면 **부동소수점수**(수학 용어로는 **실수**)를 예측하는 것
    - 어떤 사람의 교육 수준, 나이, 주거지를 바탕으로 연간 소득을 예측
    - 옥수수 농장에서 전년도 수확량과 날씨, 고용 인원수 등으로 올해 수확량을 예측



  - **출력값에 연속성**이 있는지 질문해보면 회귀와 분류 문제를 쉽게 구분할 수 있음
    - 출력값 사이에 연속성이 있다면 회귀 문제, 연속성이 없으면 분류 문제

SECTION 2.2 일반화, 과대적합, 과소적합

- 일반화 성능이 최대가 되는 모델이 최적임
  - 지도 학습에서는 훈련 데이터로 학습한 모델이 훈련 데이터와 특성이 같다면 처음 보는 새로운 데이터가 주어져도 정확히 예측할 거라 기대함
  - 모델이 처음 보는 데이터에 대해 정확하게 예측할 수 있으면 → 훈련 세트에서 테스트 세트로 **일반화** generalization 되었다고 함
  - 보통 훈련 세트에 대해 정확히 예측하도록 모델을 구축 → 훈련 세트와 테스트 세트가 매우 비슷하다면 그 모델이 테스트 세트에서도 정확히 예측하리라 기대할 수 있음 but

표 2-1 고객 샘플 데이터

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

▲표 2-1 고객 샘플 데이터

문제 정의: 요트를 구매한 고객과 구매 의사가 없는 고객의 데이터를 이용해 누가 요트를 살지 예측
관심없는 고객들을 성가시게 하지 않고 실제 구매할 것 같은 고객에게만 홍보 메일을 보내는 것이 목표

"45세 이상이고 자녀가 셋 미만이며 이혼하지 않은 고객은 요트를 살 것입니다."

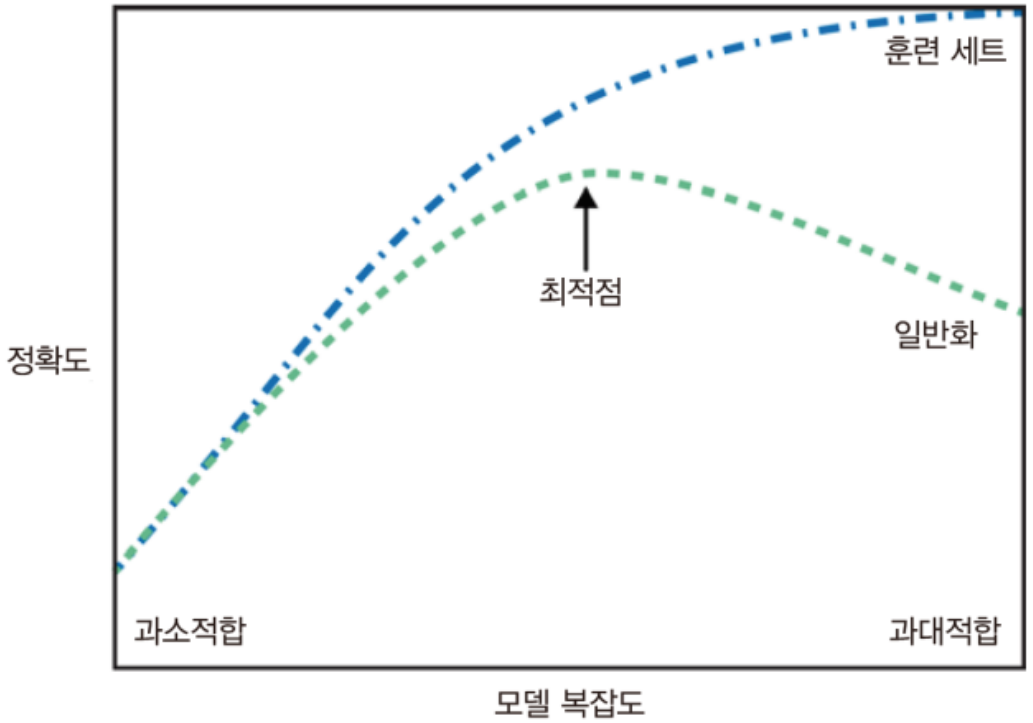
가진 정보를 모두 사용해서 너무 복잡한 모델을 만드는 것 **과대적합** overfitting

너무 간단한 모델이 선택되는 것을 **과소적합** underfitting

SECTION 2.2 일반화, 과대적합, 과소적합

◦ 모델 복잡도와 데이터셋 크기의 관계

- 모델을 복잡하게 할 수록 훈련 데이터에 대해서는 더 정확히 예측할 수 있음 but 너무 복잡해지면 훈련 세트의 각 데이터 포인트에 너무 민감해져 새로운 데이터에 잘 일반화되지 못함 → **과대적합** overfitting
- 우리가 찾으려는 모델은 일반화 성능이 최대가 되는 최적점에 있는 모델



▲그림 2-1 모델 복잡도에 따른 훈련과 테스트 정확도의 변화

모델의 복잡도는 훈련 데이터셋에 담긴 입력 데이터의 다양성과 관련있음
→ 데이터셋에 다양한 데이터 포인트가 많을수록 과대적합 없이 더 복잡한 모델을 만들 수 있음

보통 데이터 포인트를 더 많이 모으는 것이 다양성을 키워주므로
큰 데이터셋은 더 복잡한 모델을 만들 수 있게 해줌

but 같은 데이터 포인트를 중복하거나 매우 비슷한 데이터를 모으는 것은 도움이 되지 않음

데이터를 더 많이 수집하고 적절하게 더 복잡한 모델을 만들면 지도 학습 문제에서 종종 놀라운 결과를 얻을 수 있음

SECTION 2.3 지도 학습 알고리즘

- 머신러닝 알고리즘의 작동 방식 학습

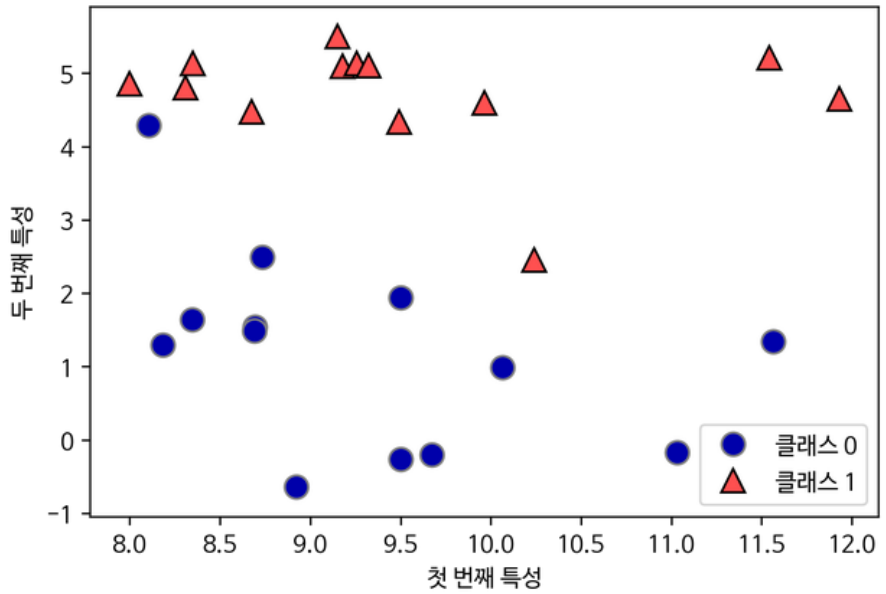
  - 데이터로부터 어떻게 학습하고 예측하는가?
  - 모델의 복잡도가 어떤 역할을 하는가?
  - 알고리즘이 모델을 어떻게 만드는가?
  - 모델들의 장단점을 평가하고 어떤 데이터가 잘 들어맞을지 살펴보기
  - 매개변수와 옵션의 의미 학습

SECTION 2.3 지도 학습 알고리즘

◦ 예제에 사용할 데이터셋

- 여러 알고리즘을 설명하기 위해 데이터셋도 여러 개 사용 → 데이터 이해가 가장 중요
  - 어떤 데이터셋은 작고 인위적으로 만든 것,
    알고리즘의 특징을 부각하기 위해 만든 것, 실제 샘플로 만든 큰 데이터셋 등

- 두 개의 특성을 가진 forge 데이터셋(인위적으로 만든 이진 분류 데이터셋)



▲**그림 2-2** forge 데이터셋의 산점도

데이터 확인하기

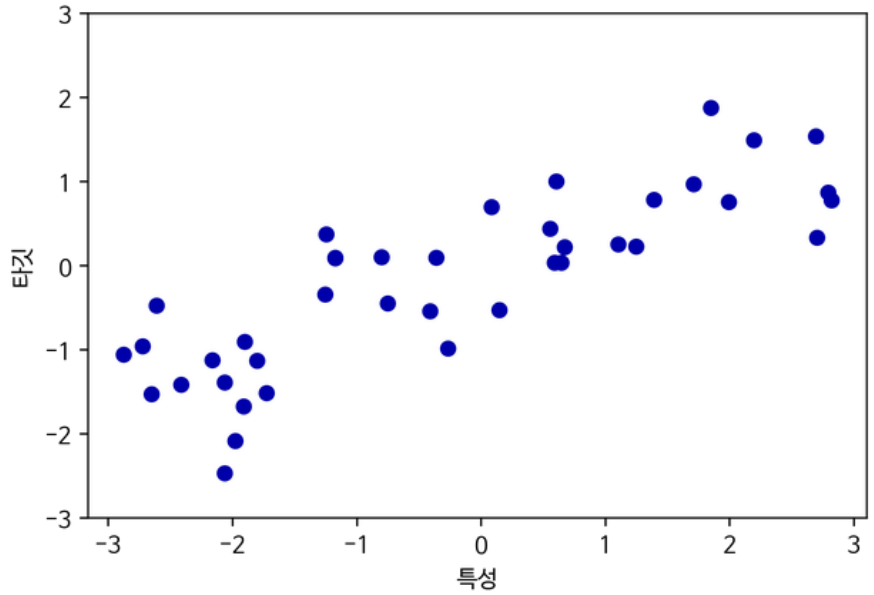
```
[ ]  print("X.shape:", X.shape)
     print("y.shape:", y.shape)

     X.shape: (26, 2)
     y.shape: (26,)
```

X.shape을 통해
데이터셋은 데이터 포인트 26개와 특성 2개
y.shape을 통해
레이블(label, y, 정답)은 [0,1] 이진 분류 형태

SECTION 2.3 지도 학습 알고리즘

◦ 예제에 사용할 데이터셋

- 회귀 알고리즘 설명을 위해 인위적으로 만든 wave 데이터셋
  - wave 데이터셋은 입력 특성 하나(feature)와 모델링할 타깃 변수(label, y, 정답, 출력)
  - 회귀 알고리즘을 위해 타킷 변수는 연속적인 값
  - 특성이 적은 데이터셋 (저차원 데이터셋) 얻은 직관 → 특성이 많은 데이터셋 (고차원 데이터셋) 발전 가능



▲**그림 2-3** x 축을 특성, y 축을 타깃으로 한 wave 데이터셋의 그래프

데이터 확인하기

```
[ ]  print("X.shape:", X.shape)
     print("y.shape:", y.shape)

     X.shape: (40, 1)
     y.shape: (40,)
```

X.shape을 통해
데이터셋은 데이터 포인트 40개와 특성 1개
y.shape을 통해
레이블(label, y, 정답) 연속적인 실수 형태

SECTION 2.3 지도 학습 알고리즘

◦ 예제에 사용할 데이터셋

- 인위적인 소규모 데이터셋 외에 scikit-learn에 들어 있는 실제 데이터셋도 두 개를 사용
- 유방암 종양의 임상 데이터를 기록해놓은 위스콘신 유방암Wisconsin Breast Cancer 데이터셋 (줄여서 cancer)
  - 각 종양은 양성benign(해롭지 않은 종양)과 악성malignant(암 종양)으로 레이블
  - 문제정의: 조직 데이터를 기반으로 **종양이 악성인지를 예측**(이진 분류 시 1로 코딩)할 수 있도록 학습하는 것
  - 데이터 다운로드: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

데이터 확인하기

```
[ ]  print("유방암 데이터의 형태:", cancer.data.shape) #569건
```

```
유방암 데이터의 형태: (569, 30)
```

```
[ ]  import numpy as np

     print("클래스별 샘플 갯수:₩n",
           {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))})
```

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

cancer.data.shape을 통해
데이터셋은 데이터 포인트 569개와 특성 30개

569개의 데이터 포인트 중
각 종양은 212개 악성(37%), 357개 양성(63%)

SECTION 2.3 지도 학습 알고리즘

◦ 예제에 사용할 데이터셋

- 보스턴 주택가격Boston Housing 회귀 분석용 실제 데이터셋
  - 문제정의: 범죄율, 찰스강 인접도, 고속도로 접근성 등의 정보를 이용해 1970년대 **보스턴 주변의 주택 평균 가격을 예측**
  - 데이터셋 다운로드: https://www.kaggle.com/c/boston-housing

데이터 확인하기

```
[ ]  print("boston.keys():₩n", boston.keys()) #데이터셋 키 확인

     boston.keys():
      dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])

[ ]  print("특성 이름:₩n", boston.feature_names) # 데이터의 특성 (feature) 확인

     특성 이름:
      ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
      'B' 'LSTAT']

[ ]  print("보스톤 주택가격 데이터의 형태:", boston.data.shape)

     보스톤 주택가격 데이터의 형태: (506, 13)
```

데이터 특성(feature) 파악하기 위해 각 컬럼 정보 확인하기

crim
per capita crime rate by town.

zn
proportion of residential land zoned for lots over 25,000 sq.ft.

indus
proportion of non-retail business acres per town.

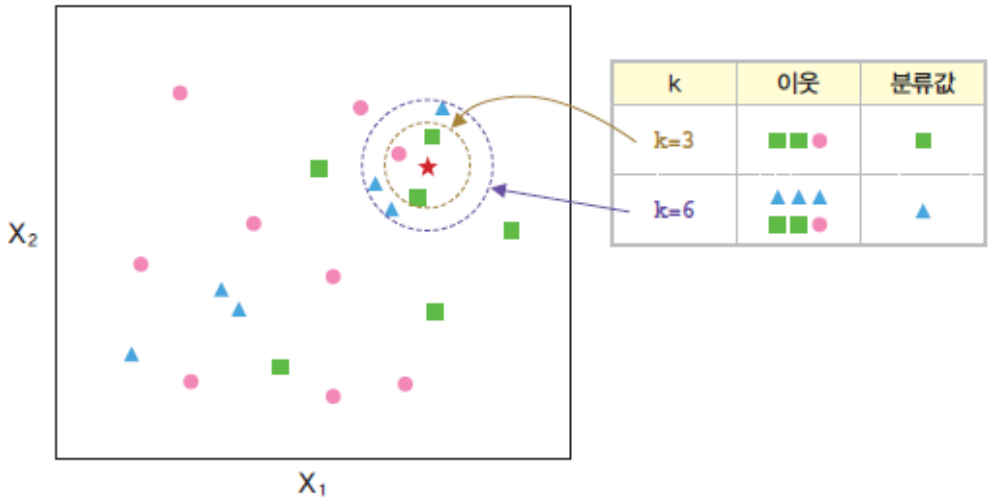
chas
Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).

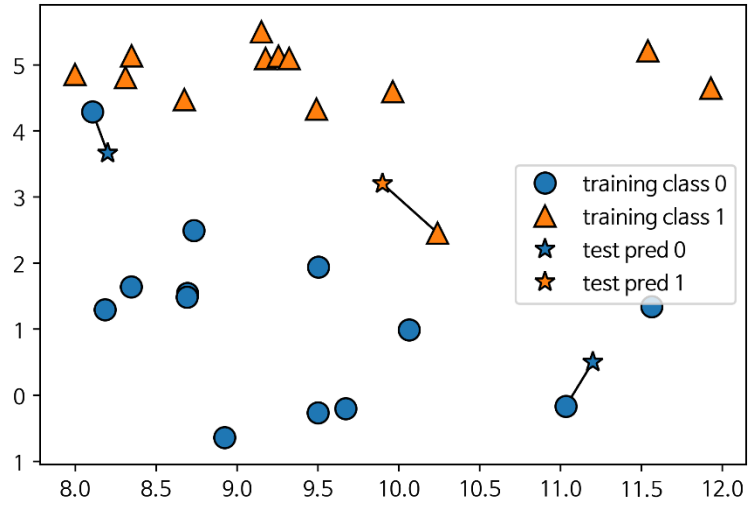
cancer.data.shape을 통해
데이터셋은 데이터 포인트 506개와 특성 13개

SECTION 2.3 지도 학습 알고리즘

◦ k-최근접 이웃 (k-NN k-Nearest Neighbors)

- k-NN k-Nearest Neighbors 알고리즘은 가장 간단한 머신러닝 알고리즘
  - 훈련 데이터셋을 그냥 저장하는 것이 모델을 만드는 과정의 전부
  - 새로운 데이터 포인트에 대해 예측할 땐 알고리즘이 훈련 데이터셋에서 가장 가까운 데이터 포인트, 즉 '최근접 이웃'을 찾음
- k-최근접 이웃 분류
  - 훈련 데이터셋에서 가장 가까운 데이터 포인트를 찾음 -> 최근접 이웃 (Nearest Neighbors)
  - 가장 간단한 k-NN 알고리즘은 가장 가까운 훈련 데이터 포인트 하나를 최근접 이웃으로 찾아 예측에 사용





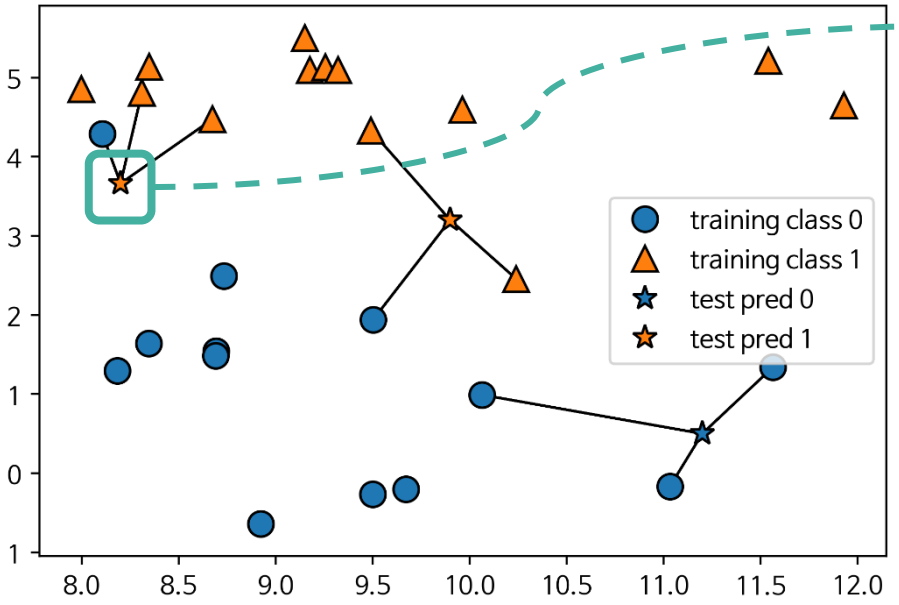
▲**그림 2-4** forge 데이터셋에 대한 1-최근접 이웃 모델의 예측

SECTION 2.3 지도 학습 알고리즘

◦ k-최근접 이웃 (k-NN k-Nearest Neighbors)

▪ k-최근접 이웃 분류

- 가장 가까운 이웃 하나가 아니라 임의의 k개를 선택 가능 → k-최근접 이웃 알고리즘
- 둘 이상의 이웃을 선택할 때는 레이블을 정하기 위해 투표
  → 즉 테스트 포인트 하나에 대해 클래스 0에 속한 이웃이 몇 개, 클래스 1에 속한 이웃이 몇 개인지 확인
  → 이웃이 더 많은 클래스를 레이블로 지정
  → k-최근접 이웃 중 다수의 클래스가 레이블



이웃을 하나만 사용했을 때와 예측이 달라짐

forge 데이터셋 분류 예 -> forge 데이터셋에 대한 3-최근접 이웃 모델의 예측

```
[ ]  plt.figure(dpi = 200)
     plt.rc('font', family ='NanumBarunGothic')

     mglearn.plots.plot_knn_classification(n_neighbors=3) # 데이터 포인트 3개 추가 (제일 근접한 3개)
```

▲ **그림 2-5** forge 데이터셋에 대한 3-최근접 이웃 모델의 예측

SECTION 2.3 지도 학습 알고리즘

- scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 분류
  - 분류 문제 정의
    - forge 데이터 셋을 사용한 이진 분류(Label, 1) 예측하기
    - k-최근접 이웃 알고리즘 적용 하여 예측하고 평가하기
  - 데이터 준비하기
    - 일반화 성능을 평가할 수 있도록 데이터 분리 -> 훈련 세트(train set) 테스트 세트(test set)

데이터 준비하기

```
[ ]  from sklearn.model_selection import train_test_split

     X, y = mglearn.datasets.make_forge() # 데이터(feature), 레이블(label, 정답)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:
  warnings.warn(msg, category=FutureWarning)
```

일반화 성능을 평가할 수 있도록 데이터 분리 -> 훈련 세트(train set) 테스트 세트(test set)

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=7)
```

X : 데이터는 대문자 X로 표현 (입력)
y : 레이블은 소문자 y로 표현 (출력)
* 2차원 배열 대문자 X, 1차원 배열(벡터) 소문자 y

train_test_split 함수 default로 분리 비율(75% : 25%)
random_state: 랜덤 시드 고정

SECTION 2.3 지도 학습 알고리즘

- scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 분류
  - Scikit-learn k-최근접 이웃 알고리즘 KNeighborsClassifier 객체 생성
    - KNeighborsClassifier를 임포트import 하고 객체 생성
    - 이웃의 수 같은 매개변수 지정 → 이웃의 수를 3으로 지정
  - 분류 모델 학습
    - 훈련 세트를 사용하여 분류 모델을 학습
    - KNeighborsClassifier에서의 학습은 예측할 때 이웃을 찾을 수 있도록 데이터를 저장하는 것

KNeighborsClassifier를 임포트import하고 객체 생성

```
[ ]  from sklearn.neighbors import KNeighborsClassifier

     clf = KNeighborsClassifier(n_neighbors=3)
```

n_neighbors=3 : 이웃의 수 3 지정

훈련 세트를 사용하여 분류 모델 학습

```
[ ]  clf.fit(X_train, y_train)

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

KNeighborsClassifier 설정된 매개변수 추후 설명

SECTION 2.3 지도 학습 알고리즘

- scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 분류
  - 예측
    - 테스트 데이터에 대해 predict 메서드를 호출해서 예측
    - 테스트 세트의 각 데이터 포인트에 대해 훈련 세트에서 가장 가까운 이웃을 계산 → 가장 많은 클래스를 찾음
  - 평가
    - 모델이 얼마나 잘 일반화되었는지 평가
    - score 메서드에 테스트 데이터와 테스트 레이블을 넣어 호출하여 모델 정확도 평가

```
테스트 데이터에 대해 predict 메서드를 호출해서 예측
테스트 세트의 각 데이터 포인트에 대해 훈련 세트에서 가장 가까운 이웃을 계산 -> 다음 가장 많은 클래스를 찾기

[ ]  print("테스트 세트 예측:", clf.predict(X_test))

     테스트 세트 예측: [0 1 0 0 0 1 1]

모델이 얼마나 잘 일반화되었는지 평가 -> 정확도
score() 함수 사용 -> 테스트 데이터와 테스트 레이블을 넣어 호출

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

     테스트 세트 정확도: 0.86

[ ]  print("테스트 세트 정확도: {:.2f}%".format(clf.score(X_test, y_test)*100))

     테스트 세트 정확도: 85.71%
```

예측은 한번도 학습하지 않은 테스트 세트 사용
X_test를 주고, 결과 예측

Score 메소드를 사용하여 모델 정확도 평가
X_test(학습하지 않은 데이터)
y_test(실제 결과)를 넣어 정확도 측정

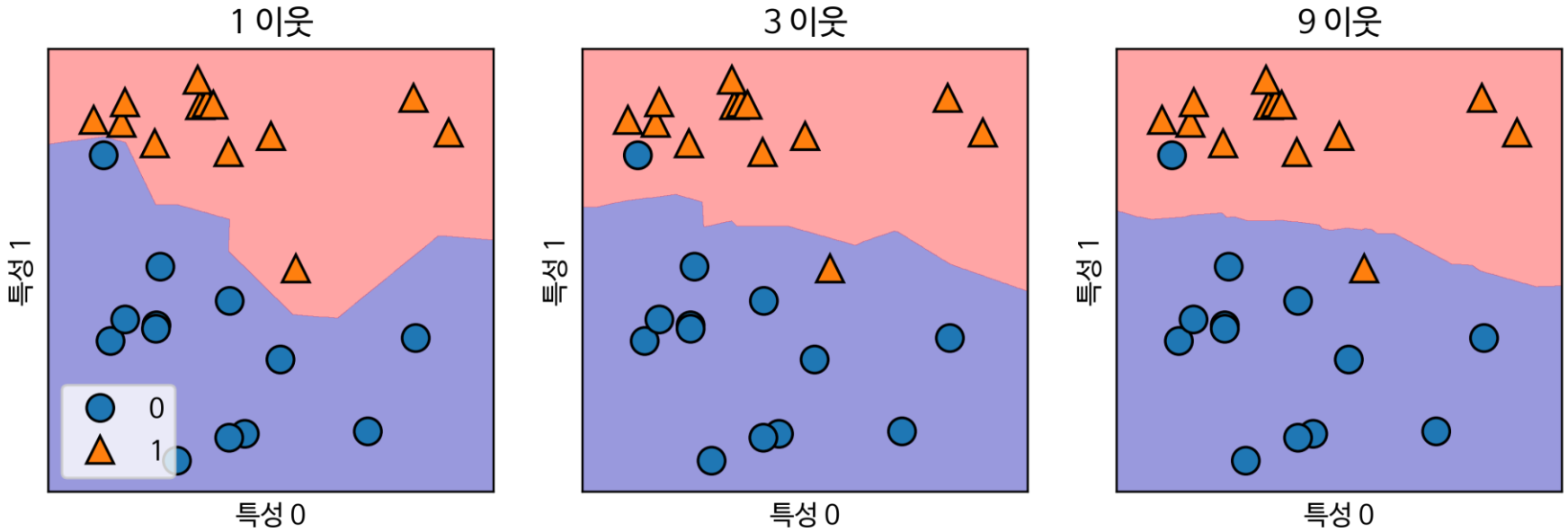
SECTION 2.3 지도 학습 알고리즘

◦ scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 분류

▪ KNeighborsClassifier 분석

- 알고리즘이 클래스 0과 클래스 1로 지정한 영역으로 나뉘는 **결정 경계**decision boundary 확인 가능
- 이웃을 하나 선택했을 때는 결정 경계가 훈련 데이터에 가깝게 따라가고 있음
- 이웃의 수를 늘릴수록 결정 경계는 더 부드러워짐
- 부드러운 경계는 더 단순한 모델을 의미

이웃을 적게 사용하면 모델의 복잡도 ↑
이웃을 많이 사용하면 모델의 복잡도 ↓

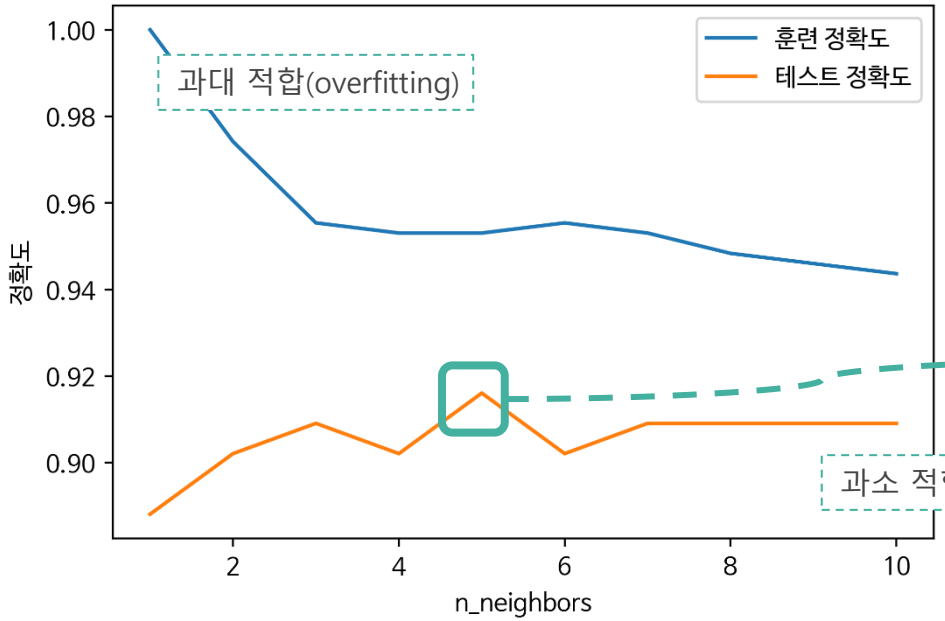


▲**그림 2-6** n_neighbors 값이 각기 다른 최근접 이웃 모델이 만든 결정 경계

SECTION 2.3 지도 학습 알고리즘

◦ scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 분류

- 유방암 데이터셋을 사용하여 이웃의 수(결정경계)에 따른 성능 평가
  - 모델의 복잡도와 일반화 사이의 관계 확인을 위해 유방암 데이터셋을 사용하여 성능 평가
  - 데이터 셋 분리(훈련셋, 테스트셋)
    → KNeighborsClassifier 메소드 1 에서 10 까지 n_neighbors 를 적용하여 모델 생성
    → 훈련 세트 정확도 저장, 테스트 세트(일반화) 정확도 저장 → 정확도 그래프로 비교



n_neighbors 수(x 축)에 따른 훈련 세트와 테스트 세트 정확도(y 축)

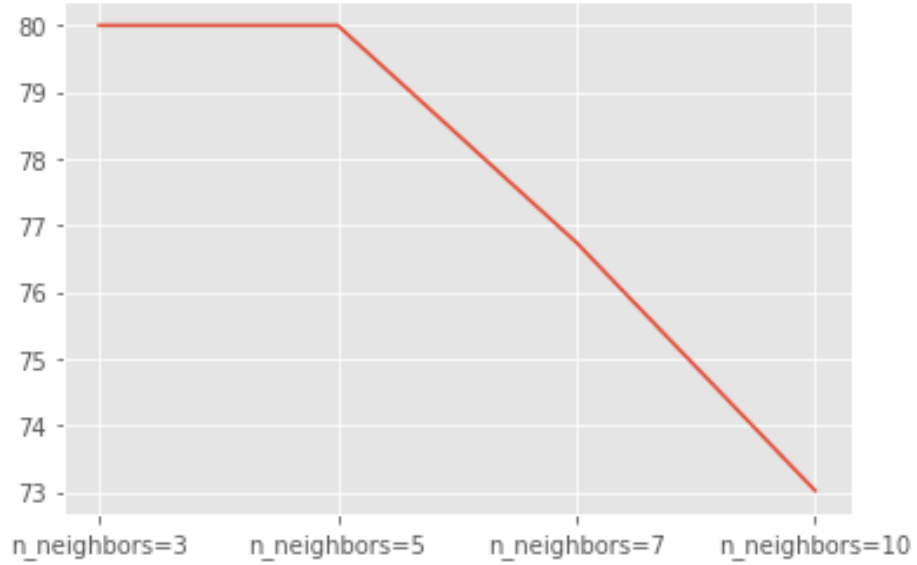
최근접 이웃의 수가 하나일 때는 훈련 데이터에 대한 예측이 완벽
but, 이웃을 하나 사용한 테스트 세트의 정확도 낮음 → 일반화 x

테스트 세트의 정확도가 가장 좋을 때
이웃 개수가 중간 정도인 다섯 개를 사용한 경우

▲ **그림 2-7** n_neighbors 변화에 따른 훈련 정확도와 테스트 정확도

SECTION 2.3 지도 학습 알고리즘

- scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 – 분류 예제
  - KNeighborsClassifier 타이타닉 생존자 예측 따른 성능 평가
    - Seaborn에서 제공하는 titanic 데이터셋 사용
    - 타이타닉 데이터 전처리 (원핫인코딩 - 범주형 데이터를 모형이 인식할 수 있도록 숫자형으로 변환)
    - 데이터 셋 분리(훈련셋, 테스트셋)
      → KNeighborsClassifier 메소드 n_neighbors 5를 적용하여 모델 생성
      → 테스트 세트(일반화) 예측 저장 → 모형 성능 평가 - Confusion Matrix 계산, 평가지표 계산
    - n_neighbors 3, n_neighbors 7, n_neighbors 10 모델 변경 하여 성능 평가 진행하여 모델 설명하기

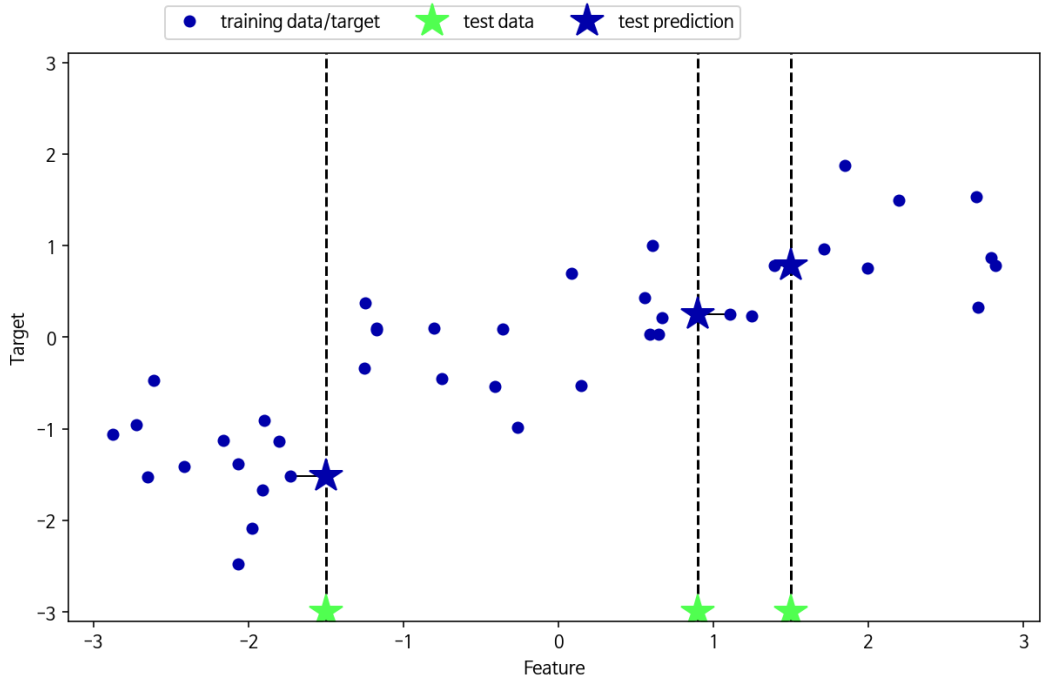


테스트 평가지표 n_neighbors=5

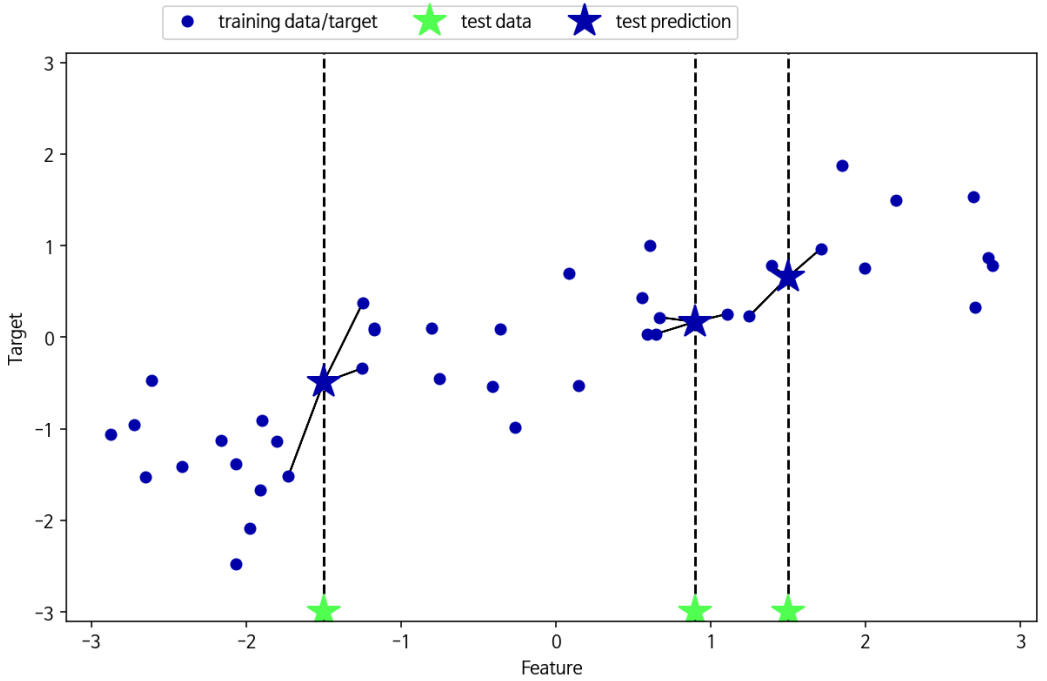
	precision	recall	f1-score	support
0	0.79	0.89	0.84	125
1	0.81	0.68	0.74	90
accuracy			0.80	215
macro avg	0.80	0.78	0.79	215
weighted avg	0.80	0.80	0.80	215

SECTION 2.3 지도 학습 알고리즘

- scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 회귀
  - k-최근접 이웃 회귀 (k-Neighbors Regression)
    - k-최근접 이웃 알고리즘은 회귀 분석에도 쓰임
    - k=1 경우 그냥 가장 가까운 이웃의 타켓값
    - k >=2 경우 회귀분석 → 여러 개의 최근접 이웃 간의 평균(average or mean)이 예측 값



▲ **그림 2-8** wave 데이터셋에 대한 1-최근접 이웃 회귀 모델의 예측



▲ **그림 2-9** wave 데이터셋에 대한 3-최근접 이웃 회귀 모델의 예측

SECTION 2.3 지도 학습 알고리즘

- scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 회귀
  - 분류 문제 정의
    - Wave 데이터 셋을 사용한 연속적인 값 예측하기
    - k-최근접 이웃 회귀 알고리즘 적용 하여 예측하고 평가하기
  - 데이터 준비하기
    - 일반화 성능을 평가할 수 있도록 데이터 분리 -> 훈련 세트(train set) 테스트 세트(test set)

데이터 셋 분리하기(훈련셋, 테스트셋)

```
[ ]  from sklearn.neighbors import KNeighborsRegressor
     from sklearn.model_selection import train_test_split

     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)
```

X : 데이터는 대문자 X로 표현 (입력)
y : 레이블은 소문자 y로 표현 (출력)
* 2차원 배열 대문자 X, 1차원 배열(벡터) 소문자 y

make_wave(n_samples=40) 데이터 사이즈 40개 설정

train_test_split 함수 default로 분리 비율(75% : 25%)
random_state: 랜덤 시드 고정

SECTION 2.3 지도 학습 알고리즘

◦ scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 회귀

- Scikit-learn k-최근접 이웃 회귀 알고리즘 KNeighborsRegressor 객체 생성
  - KNeighborsRegressor를 임포트import 하고 객체 생성
  - 이웃의 수 같은 매개변수 지정 → 이웃의 수를 3으로 지정
- 회귀 모델 학습
  - 훈련 세트를 사용하여 회귀 모델을 학습
  - KNeighborsRegressor에서의 학습은 예측할 때 이웃을 찾을 수 있도록 데이터를 저장하는 것

KNeighborsRegressor를 임포트import하고 객체 생성

```
[ ]  # 이웃의 수를 3으로 하여 모델의 객체를 만듭니다
     reg = KNeighborsRegressor(n_neighbors=3)
```

n_neighbors=3 : 이웃의 수 3 지정

훈련 세트를 사용하여 회귀 모델 학습

```
[ ]  # 훈련 데이터와 타깃을 사용하여 모델을 학습시킵니다
     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')
```

weights='uniform' np.mean 함수 사용하여 단순 평균 계산
Distance 일 경우 거리를 고려한 가중치 평균(average) 계산

SECTION 2.3 지도 학습 알고리즘

◦ scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 회귀

▪ 예측

- 테스트 데이터에 대해 predict 메서드를 호출해서 예측
- 테스트 세트의 각 데이터 포인트에 대해 훈련 세트에서 가장 가까운 이웃을 계산
  → 여러 개의 최근접 이웃 간의 평균(average or mean)이 예측 값

▪ 평가

- score 메서드를 사용해 모델 평가 하는데, 회귀일 땐 R2(결정계수) 값을 반환
- R2 값은 회귀 모델에서 예측의 적합도를 측정한 것

예측하기

```
[ ]  print("테스트 세트 예측:₩n", reg.predict(X_test))
```

```
테스트 세트 예측:
 [-0.05396539  0.35686046  1.136
  0.35686046  0.91241374 -0.4468
```

R2(결정계수) 0~1사이의 값
1은 예측이 완벽한 경우
0은 훈련 세트의 출력값이 y_train의 평균으로만 예측하는 모델일 경우

성능평가 하기

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

```
테스트 세트 R^2: 0.83
```

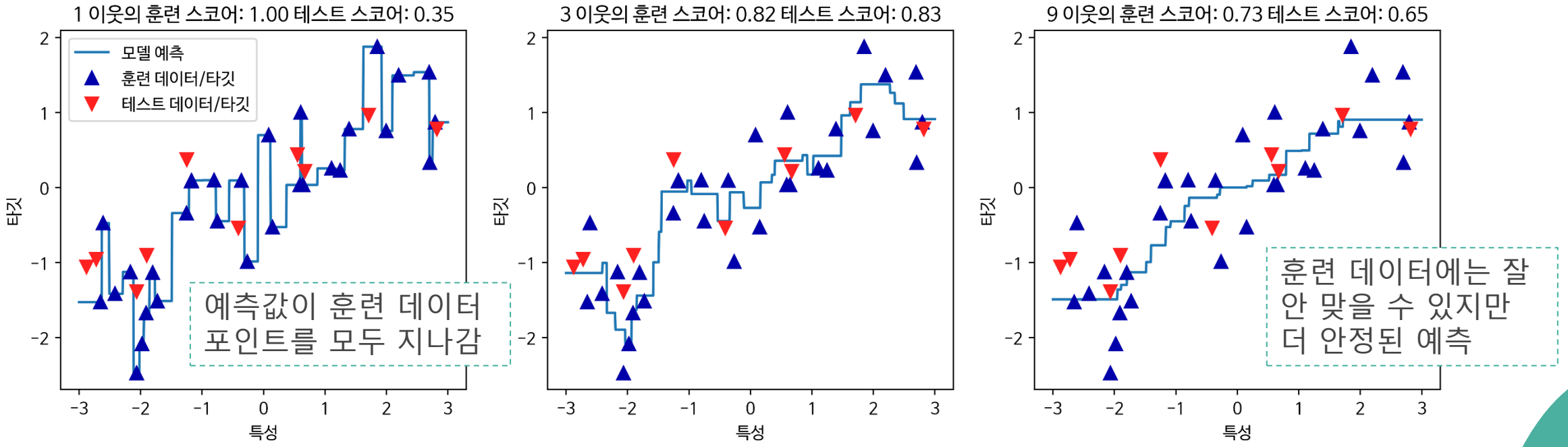
테스트 세트의 R2 점수 0.83 → 비교적 잘 맞는 모델

SECTION 2.3 지도 학습 알고리즘

◦ scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현 - 회귀

▪ KNeighborsRegressor 분석

- 1차원 데이터셋에 대해 가능한 모든 특성 값을 만들어 예측
- -3과 3 사이에 1,000개의 데이터 포인트 생성 → KNeighborsRegressor 1, 3, 9 이웃을 사용한 예측
- 훈련 세트 적합도 저장, 테스트 세트(일반화) 적합도 저장 → 적합도 그래프로 비교



▲ **2-10** n_neighbors 값에 따라 최근접 이웃 회귀로 만들어진 예측 비교

SECTION 2.3 지도 학습 알고리즘

◦ scikit-learn을 사용해서 k-최근접 이웃 알고리즘 구현

▪ 장단점과 매개변수

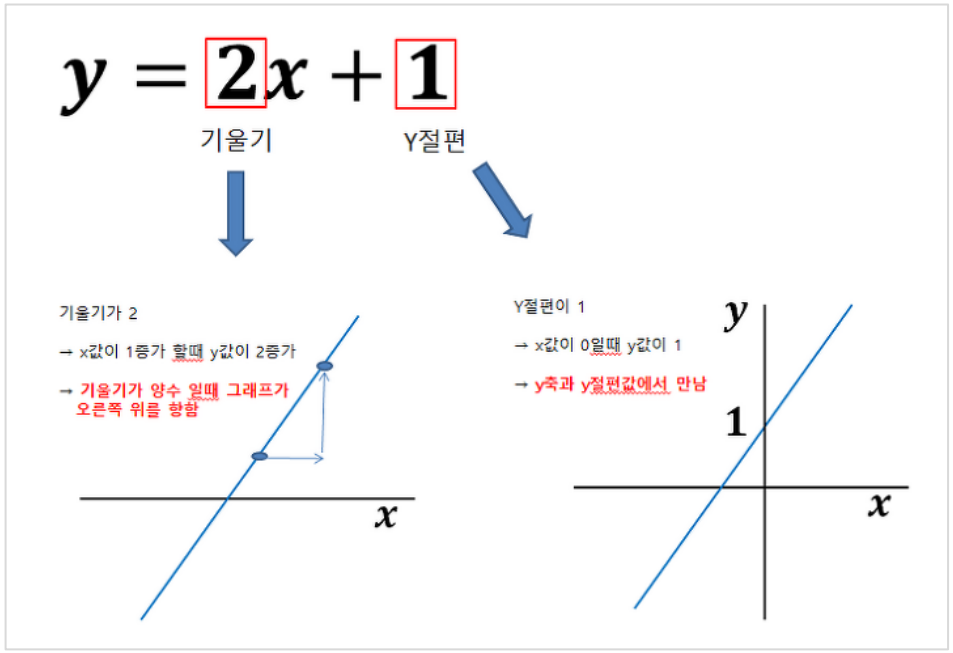
- 일반적으로 KNeighbors 분류기에 중요한 매개변수 : **데이터 포인트 사이의 거리를 재는 방법, 이웃의 수**
  - 실제로 이웃의 수는 3개나 5개 정도로 적을 때 잘 작동하지만, 이 매개변수는 잘 조정해야함
- k-NN의 장점
  - 이해하기 매우 쉬운 모델
  - 많이 조정하지 않아도 자주 좋은 성능을 발휘 더 복잡한 알고리즘을 적용해보기 전에 시도해볼 수 있는 좋은 시작점
- K-NN의 단점
  - 보통 최근접 이웃 모델은 매우 빠르게 만들 수 있지만, 훈련 세트가 매우 크면 예측이 느려짐
  - k-NN 알고리즘을 사용할 땐 데이터를 전처리하는 과정이 중요
  - (수백 개 이상의) 많은 특성을 가진 데이터셋에는 잘 동작하지 않음
  - 특성 값 대부분이 0인 (즉 희소한) 데이터셋과는 특히 잘 작동하지 않음
- k-최근접 이웃 알고리즘이 이해하긴 쉽지만,
  예측이 느리고 많은 특성을 처리하는 능력이 부족해 현업에서는 잘 쓰지 않음
- 이런 단점이 없는 알고리즘 → 선형 모델

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델

▪ 선형 모델linear model

- 100여 년 전에 개발
- 지난 몇십 년 동안 폭넓게 연구되고 현재도 널리 쓰임
- 선형 모델은 입력 특성에 대한 **선형 함수**를 만들어 예측을 수행



▪ 회귀의 선형 모델

- 회귀의 경우 선형 모델을 위한 일반화된 예측 함수
- $\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \ldots + w[p] \times x[p] + b$
- x[0]부터 x[p]까지는 하나의 데이터 포인트에 대한 특성(특성의 개수는 p + 1)
- w(기울기)와 b(절편)는 모델이 학습할 파라미터
- 1 그리고 $\hat{y}$은 모델이 만들어낸 예측값
- 특성이 하나인 데이터셋이라면 → $\hat{y} = w[0] \times x[0] + b$
- 특성이 많아지면 w는 각 특성에 해당하는 기울기를 모두 가짐
- 다르게 생각하면 예측값은 입력 특성에 w의 각 가중치(음수일 수도 있음)를 곱해서 더한 가중치 합

SECTION 2.3 지도 학습 알고리즘

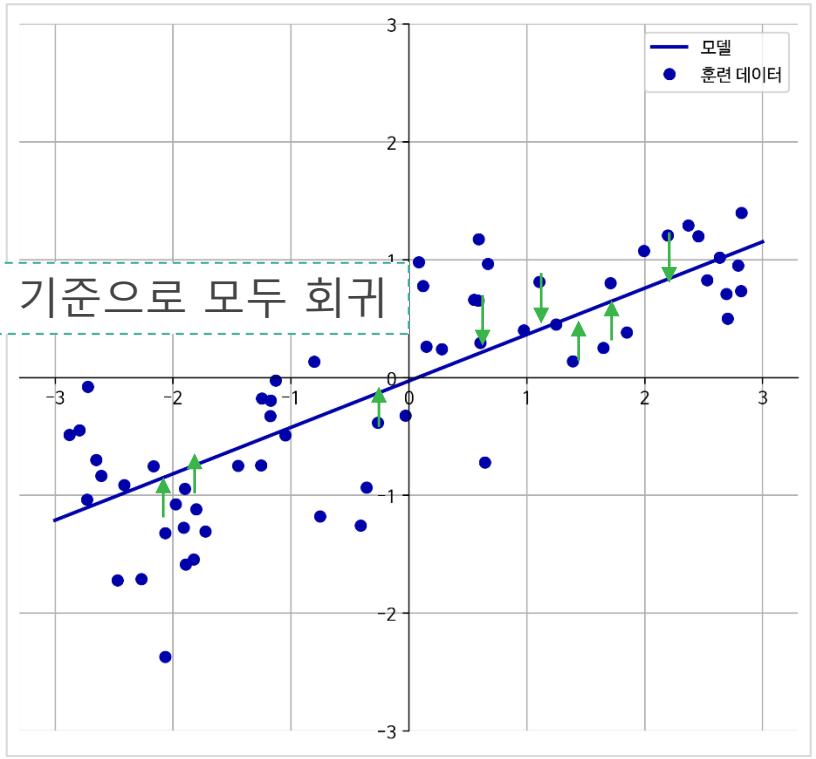
◦ 선형 모델

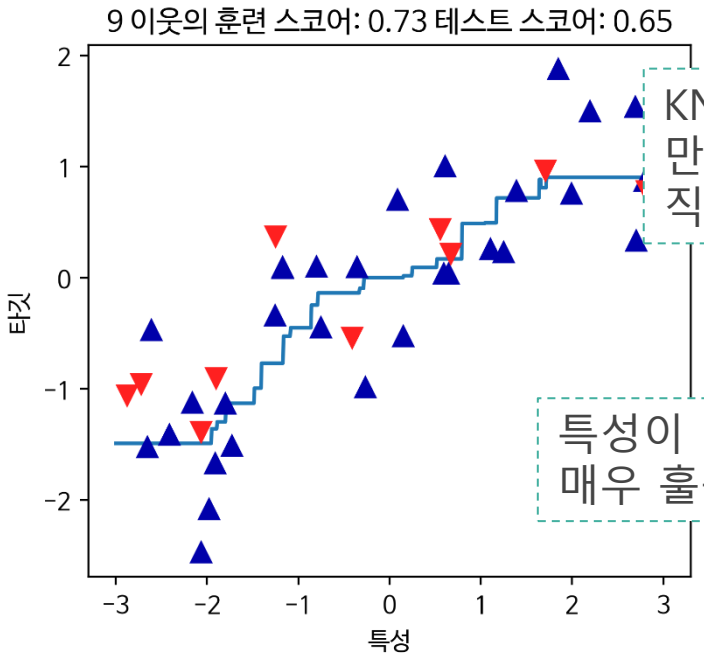
▪ 회귀의 선형 모델 예측

- 1차원 wave 데이터셋으로 파라미터 w[0]와 b를 직선처럼 되도록 학습
- 직선 방정식을 이해하기 쉽도록 그래프의 중앙을 가로질러서 x, y 축을 그림

회귀에서 가장 인기 있는 선형 모델
뒤에서 살펴봄

이 직선 기준으로 모두 회귀





KNeighborsRegressor를 사용하여
만든 선과 비교해보면
직선을 사용한 예측이 더 제약이 많아 보임

특성이 많은 데이터셋이라면 선형 모델은
매우 훌륭한 성능을 낼 수 있음

▲ **그림 2-11** wave 데이터셋에 대한 선형 모델의 예측

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 선형 회귀(최소제곱법)

▪ 선형 회귀(최소제곱법)

- **선형 회귀**linear regression 또는 **최소제곱법**OLS, ordinary least squares은 가장 간단하고 오래된 회귀용 선형 알고리즘
- 선형 회귀는 예측과 훈련 세트에 있는 타깃 y 사이의 **평균제곱오차**mean squared error를 최소화하는 파라미터 w와 b를 찾는 것
- 평균제곱오차는 예측값과 타깃값의 차이를 제곱하여 더한 후에 샘플의 개수로 나눈 것
- 선형 회귀는 매개변수가 없는 것이 장점 but 모델의 복잡도를 제어할 방법도 없음

선형모델 만든 후 학습시키기

```
[6]  from sklearn.linear_model import LinearRegression

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

선형 모델을 만드는 코드 (데이터 준비하기 생략)

lr 객체 확인하기

```
[7]  print("lr.coef_:", lr.coef_)
     print("lr.intercept_:", lr.intercept_)

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

기울기 파라미터(w)는 **가중치**weight 또는 **계수**coefficient
lr 객체의 coef_ 속성에 저장

편향offset or 절편intercept 파라미터(b)
intercept_ 속성에 저장

scikit-learn은 훈련 데이터에서 유도된 속성은 항상 끝에 밑줄을 붙임
그 이유는 사용자가 지정한 매개변수와 구분하기 위해서

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 선형 회귀(최소제곱법)

▪ 선형 회귀(최소제곱법) 성능 확인

- 훈련 세트와 테스트 세트의 성능을 확인
- score 메서드를 사용해 모델 평가 하는데, 회귀일 땐 R2(결정계수) 값을 반환
- 값이 0.66인 것은 그리 좋은 결과는 아님 but 훈련 세트와 테스트 세트의 점수가 매우 비슷함

성능 평가하기

```
[8]  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: 0.67
     테스트 세트 점수: 0.66
```

과대적합이 아니라 과소적합인 상태를 의미

- 1 차원 데이터셋에서는 모델이 매우 단순하므로 (혹은 제한적이므로) 과대적합을 걱정할 필요가 없음
- 그러나 (특성이 많은) 고차원 데이터셋에서는 선형 모델의 성능이 매우 높아져서 과대적합될 가능성이 높음

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 선형 회귀(최소제곱법)

▪ 보스턴 주택가격 데이터셋을 사용한 선형 회귀 성능 평가

- LinearRegression 모델이 보스턴 주택가격 데이터셋 같은 복잡한 데이터셋에서 어떻게 동작하는지 확인
- 데이터셋에는 샘플이 506개가 있고 특성은 유도된 것을 합쳐 105개
- 데이터셋 준비 → 훈련 세트와 테스트 세트로 나누기 → 선형 모델 생성
- 훈련 세트와 테스트 세트의 점수를 비교 → 훈련 세트에서는 예측이 매우 정확함
- 테스트 세트에서는 R2 값이 매우 낮음
  → 과대적합 → 확실한 신호이므로 복잡도를 제어할 수 있는 모델을 사용해야 함
  → 릿지 회귀

보스턴 주택가격 데이터셋을 사용한 선형 회귀 성능 평가

```
[9]  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)
```

```
[10] 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
```

과대적합(overfitting) 인 상태를 의미

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 릿지 회귀

▪ 릿지Ridge 회귀

- 회귀를 위한 선형 모델이므로 최소적합법에서 사용한 것과 같은 예측 함수를 사용
- 릿지 회귀에서의 가중치(w) 선택은 훈련 데이터를 잘 예측하기 위해서 뿐만 아니라
  추가 제약 조건을 만족시키기 위한 목적도 있음 → 가중치의 절댓값을 가능한 한 작게 만드는 것
- w의 모든 원소가 0에 가깝게 되길 원함
- 모든 특성이 출력에 주는 영향을 최소한으로 만듦 (기울기를 작게 만듦) → 이런 제약을 규제regularization

▪ 규제regularization

- 규제랑 과대적합이 되지 않도록 모델을 강제로 제한한다는 의미
- 릿지 회귀에 사용하는 규제 방식을 L2 규제라고 함
- 수학적으로 릿지는 계수의 L2 노름norm의 제곱을 페널티로 적용
- 릿지 회귀는 linear_model.Ridge에 구현

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 릿지 회귀

▪ 보스턴 주택가격 데이터셋을 사용한 릿지Ridge 회귀 성능 평가

- Ridge 모델이 보스턴 주택가격 데이터셋 같은 복잡한 데이터셋에서 어떻게 동작하는지 확인

보스턴 주택가격 데이터셋을 사용한 릿자 회귀 성능 평가

```
[11]  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
```

훈련 세트에서의 점수는 LinearRegression보다 낮지만 테스트 세트에 대한 점수는 더 높음

모델의 복잡도가 낮아지면 훈련 세트에서의 성능은 나빠지지만 더 일반화된 모델

- Ridge는 모델을 단순하게(계수를 0에 가깝게) 해주고 훈련 세트에 대한 성능 사이를 절충할 수 있는 방법을 제공

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 릿지 회귀

▪ 보스턴 주택가격 데이터셋을 사용한 릿지Ridge 회귀 성능 평가

- alpha 매개변수로 훈련 세트의 성능 대비 모델을 얼마나 단순화할지를 지정
- 같은 데이터셋과 릿지 회귀 모델에서 alpha 값만 조정하여 성능 평가
- alpha 값을 높이면 계수를 0에 더 가깝게 만들어서 훈련 세트의 성능은 나빠지지만 일반화에는 도움을 줌

alpha 매개변수로 훈련 세트의 성능 대비 모델 단순화 지정

릿지 회귀 모델 기본값 alpha=1.0

```
[12]  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
```

alpha=10로 지정하여 학습 후 성능 평가

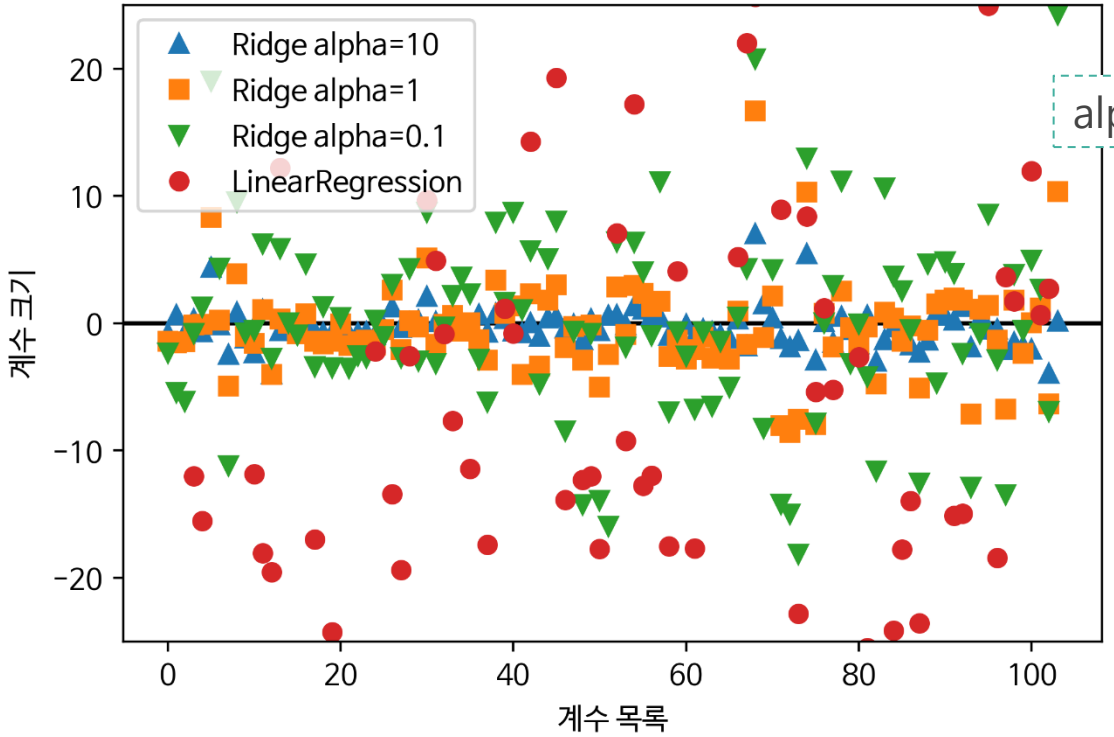
```
[13]  ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
      print("훈련 세트 점수: {:.2f}".format(ridge01.score(X_train, y_train)))
      print("테스트 세트 점수: {:.2f}".format(ridge01.score(X_test, y_test)))

      훈련 세트 점수: 0.93
      테스트 세트 점수: 0.77
```

alpha=0.1로 지정하여 학습 후 성능 평가

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 릿지 회귀

- alpha 값에 따라 모델의 coef_ 속성이 어떻게 달라지는지를 조사
  - alpha 매개변수가 모델을 어떻게 변경시키는지 더 깊게 이해할 수 있음
  - 높은 alpha 값은 제약이 더 많은 모델이므로 작은 alpha 값일 때보다 coef_의 절댓값 크기가 작을 것이라고 예상



alpha=10일 때 대부분의 계수는 -3과 3 사이에 위치

alpha=1일 때 Ridge 모델의 계수는 좀 더 커짐
alpha=0.1일 때 계수는 더 커짐

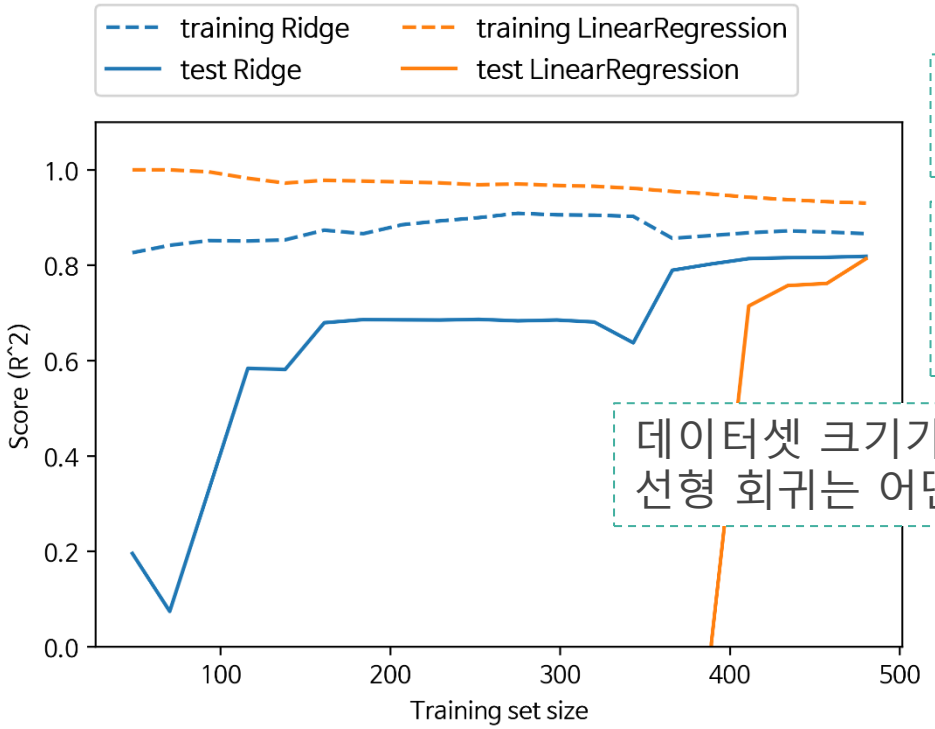
아무런 규제가 없는(alpha=0) 선형 회귀의 계수는 값이 더 커져 그림 밖으로 넘어감

▲ **그림 2-12** 선형 회귀와 몇 가지 alpha 값을 가진 릿지 회귀의 계수 크기 비교

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 릿지 회귀

- 데이터셋의 크기에 따른 **학습 곡선**learning curve
  - 규제의 효과를 이해하는 또 다른 방법은 alpha 값을 고정하고 훈련 데이터의 크기를 변화시켜 보는 것
  - 데이터셋의 크기에 따른 모델의 성능 변화를 나타낸 그래프를 **학습 곡선**learning curve이라고 함
  - 보스턴 주택가격 데이터셋에서 여러 가지 크기로 샘플링하여 LinearRegression과 Ridge(alpha=1)을 적용한 것



모든 데이터셋에 대해 릿지와 선형 회귀 모두 훈련 세트의 점수가 테스트 세트의 점수보다 높음

릿지에는 규제가 적용되므로 릿지의 훈련 데이터 점수가 전체적으로 선형 회귀의 훈련 데이터 점수보다 낮음
But 테스트 데이터에서는 릿지의 점수가 더 높음

데이터셋 크기가 400 미만에서는
선형 회귀는 어떤 것도 학습하지 못함

마지막에는 선형 회귀가 릿지 회귀를 따라잡음
데이터를 충분히 주면 규제 항은 덜 중요해져서
릿지 회귀와 선형 회귀의 성능이 같아질 것이라는 점

▲ **12-13** 보스턴 주택가격 데이터셋에 대한 릿지 회귀와 선형 회귀의 학습 곡선

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 라쏘

▪ 라쏘lasso

- 선형 회귀에 규제를 적용하는 데 Ridge의 대안으로 Lasso 사용
- 릿지 회귀에서와 같이 라쏘lasso도 계수를 0에 가깝게 만들려고 함
- 방식이 조금 다르며 이를 L1 규제 → L1 규제의 결과로 라쏘를 사용할 때 어떤 계수는 정말 0이 됨
- 모델에서 완전히 제외되는 특성이 생긴다는 뜻 → 특성 선택feature selection이 자동으로 이뤄진다고 볼 수 있음
- 일부 계수를 0으로 만들면 모델을 이해하기 쉬워지고 이 모델의 가장 중요한 특성이 무엇인지 드러내줌
- 확장된 보스턴 주택가격 데이터셋에 라쏘를 적용

확장된 보스턴 주택가격 데이터셋에 라쏘를 적용

```
[20]  from sklearn.linear_model import Lasso
      import numpy as np

      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:", np.sum(lasso.coef_ != 0))

      훈련 세트 점수: 0.29
      테스트 세트 점수: 0.21
      사용한 특성의 개수: 4
```

훈련 세트, 테스트 세트 점수 둘 다 좋지 않음

과소적합(underfitting)
105개의 특성 중 4개만 사용

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 라쏘

- 라쏘lasso 과소적합을 줄이기 위해서 alpha 값 설정
  - lasso도 계수를 얼마나 강하게 0으로 보낼지를 조절하는 alpha 매개변수를 지원
  - 기본값인 alpha=1.0을 사용하니 과소적합 발생
  - 과소적합 줄이기 위해서 alpha 값을 줄이기 → max_iter(반복 실행하는 최대 횟수)의 기본값을 늘려야 함

과소적합을 줄이기 위해서 alpha 값 설정

```
[21]  # max_iter 기본 값을 증가시키지 않으면 max_iter 값을 늘이라는 경고가 발생합니다
      lasso001 = Lasso(alpha=0.01, max_iter=50000).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:", np.sum(lasso001.coef_ != 0))

      훈련 세트 점수: 0.90
      테스트 세트 점수: 0.77
      사용한 특성의 개수: 33

[22]  lasso00001 = Lasso(alpha=0.0001, max_iter=50000).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:", np.sum(lasso00001.coef_ != 0))

      훈련 세트 점수: 0.95
      테스트 세트 점수: 0.64
      사용한 특성의 개수: 96
```

alpha 값을 낮추면 모델의 복잡도는 증가하여
훈련 세트와 테스트 세트에서의 성능이 좋아짐

성능은 Ridge보다 조금 나은데 사용된 특성은 105개 중 33개 사용
→ 모델을 분석하기가 조금 더 쉬움

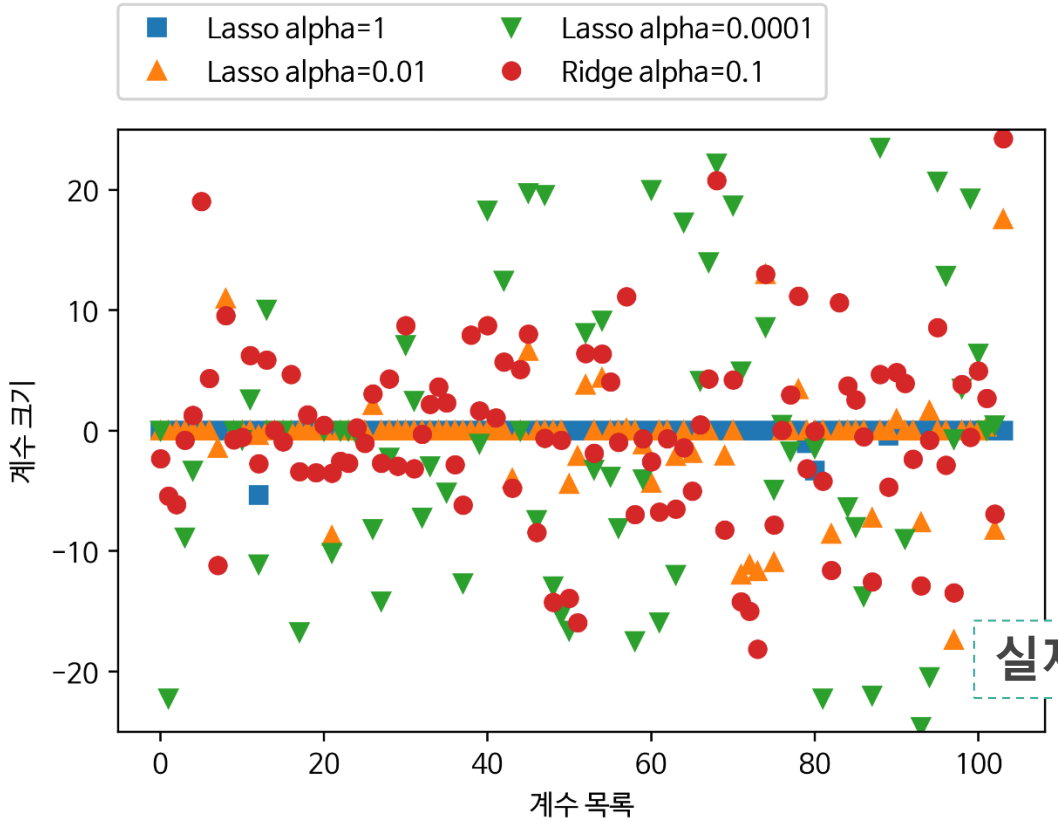
but alpha 값을 너무 낮추면 규제의 효과가 없어져
과대적합이 되므로 LinearRegression의 결과와 비슷해짐

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 - 라쏘

▪ 라쏘lasso alpha 값이 다른 모델들의 계수 비교 그래프

- alpha 값이 다른 모델(릿지, 라쏘)들의 계수를 그래프로 표현하여 비교



alpha=1일 때 (이미 알고 있듯)
계수 대부분이 0일 뿐만 아니라 나머지 계수들도 크기가 작음

alpha를 0.01로 줄이면 대부분의 특성이 0이 되는
(정삼각형 모양으로 나타낸) 분포를 얻게 됨

alpha=0.0001이 되면 계수 대부분이 0이 아니고
값도 커져 꽤 규제받지 않은 모델

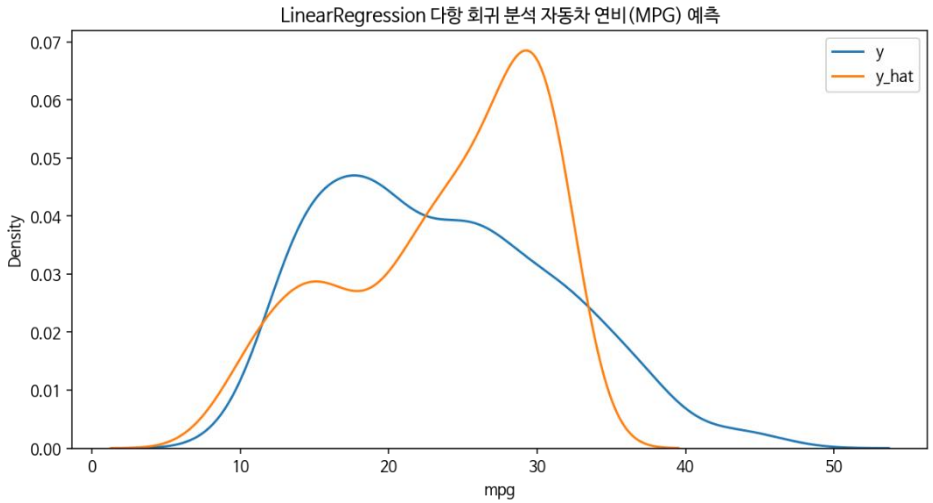
실제 이 두 모델 중 보통은 릿지 회귀를 선호

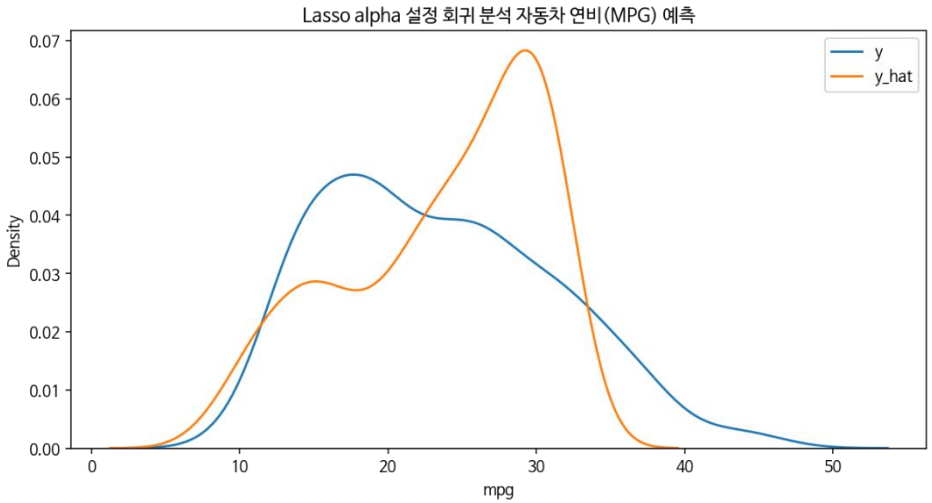
▲ **그림 2-14** 릿지 회귀와 alpha 값이 다른 라쏘 회귀의 계수 크기 비교

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 – 예제

- LinearRegression 자동차 연비(MPG) 예측 따른 성능 평가
  - https://archive.ics.uci.edu/ml/datasets/auto+mpg 에서 제공하는 자동차 연비 데이터셋 사용
  - 자동자 연비 데이터 전처리 (데이터 타입 변환 및 데이터 클랜징)
  - 데이터 셋 분리(훈련셋, 테스트셋)
    → 선형회귀분석 모듈 LinearRegression 메소드 적용하여 모델 생성
    → 테스트 세트(일반화) 예측 저장 → 모형 성능 평가
  - 단순 회귀 분석, 다항회귀 분석,라쏘 회귀 분석 모델 변경하며 성능 평가 진행 → 모델 설명하기
  - **실습 : 릿지Ridge 회귀 모델을 사용한 학습 및 성능 평가 → 모델 설명하기**





SECTION 2.3 지도 학습 알고리즘

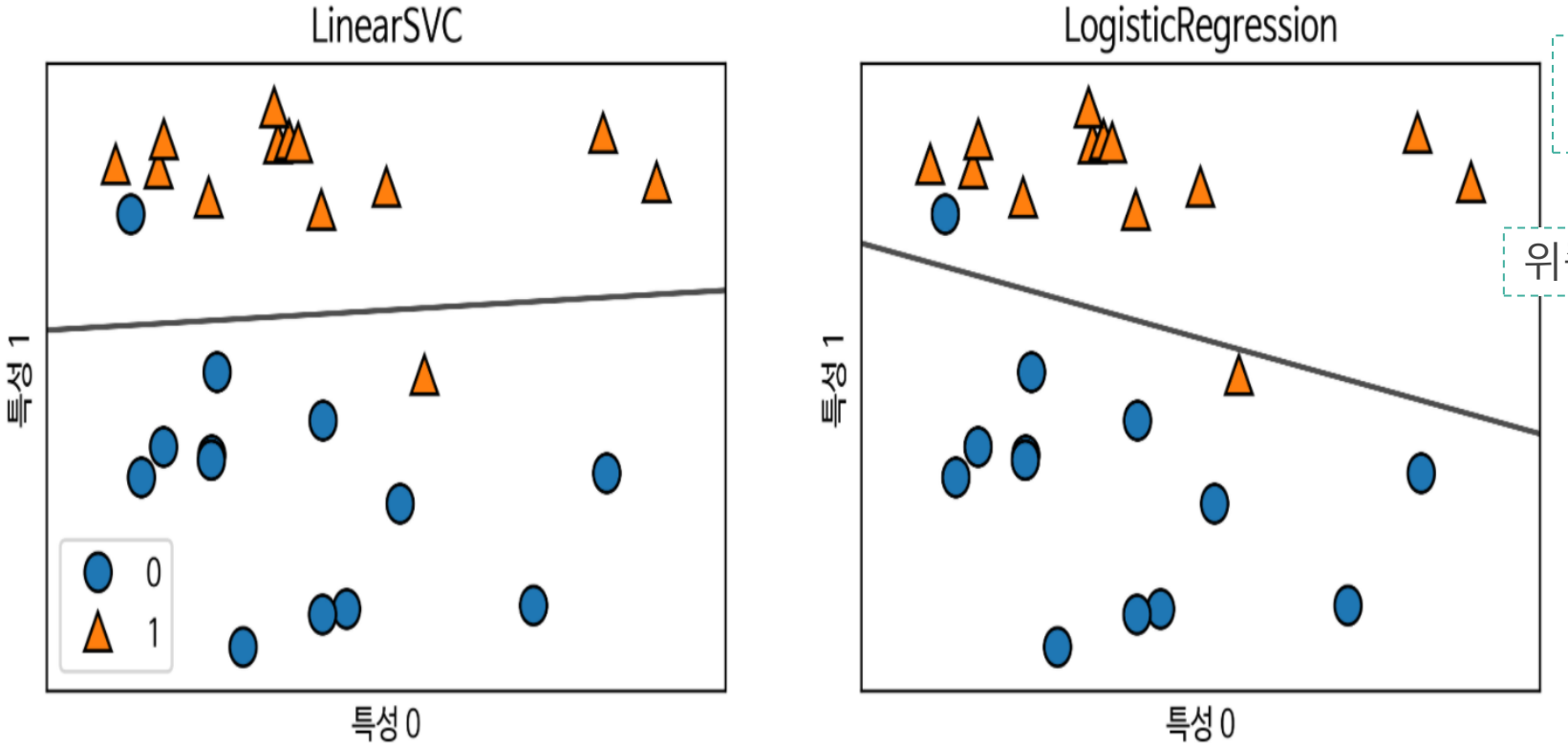
◦ 분류형 선형 모델

- 선형 모델은 분류에도 널리 사용 → ex) 이진 분류binary classification
  - $\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \ldots + w[p] \times x[p] + b > 0$
  - 분류용 선형 모델에서는 **결정 경계**가 입력의 선형 함수
  - (이진) 선형 분류기는 선, 평면, 초평면을 사용해서 두 개의 클래스를 구분하는 분류기
- 선형 모델을 학습시키는 알고리즘 두가지 방법
  - 특정 계수와 절편의 조합이 훈련 데이터에 얼마나 잘 맞는지 측정하는 방법
  - 사용할 수 있는 규제가 있는지, 있다면 어떤 방식인지
  - 불행하게도 수학적이고 기술적인 이유로, 알고리즘들이 만드는 잘못된 분류의 수를 최소화하도록 w와 b를 조정하는 것은 불가능
- 가장 널리 알려진 두 개의 선형 분류 알고리즘
  - linear_model.LogisticRegression 구현된 로지스틱 회귀logistic regression
  - svm.LinearSVC(SVC; support vector classifier)구현된 선형 서포트 벡터 머신support vector machine

SECTION 2.3 지도 학습 알고리즘

◦ 분류형 선형 모델 – 로지스틱 회귀, 서포트 벡터 머신

- forge 데이터셋을 사용하여 LogisticRegression과 LinearSVC 모델들이 만들어낸 결정 경계를 그래프 비교
  - forge 데이터셋의 첫 번째 특성을 x 축, 두 번째 특성을 y 축
  - 회귀에서 본 Ridge와 마찬가지로 이 두 모델은 기본적으로 L2 규제를 사용



linearSVC와 LogisticRegression으로 만든 결정 경계가 직선으로 표현

위쪽은 클래스 1, 아래쪽은 클래스 0으로 나눔

새로운 데이터가
직선 위쪽에 놓이면 클래스 1로 분류
직선 아래쪽에 놓이면 클래스 0으로 분류

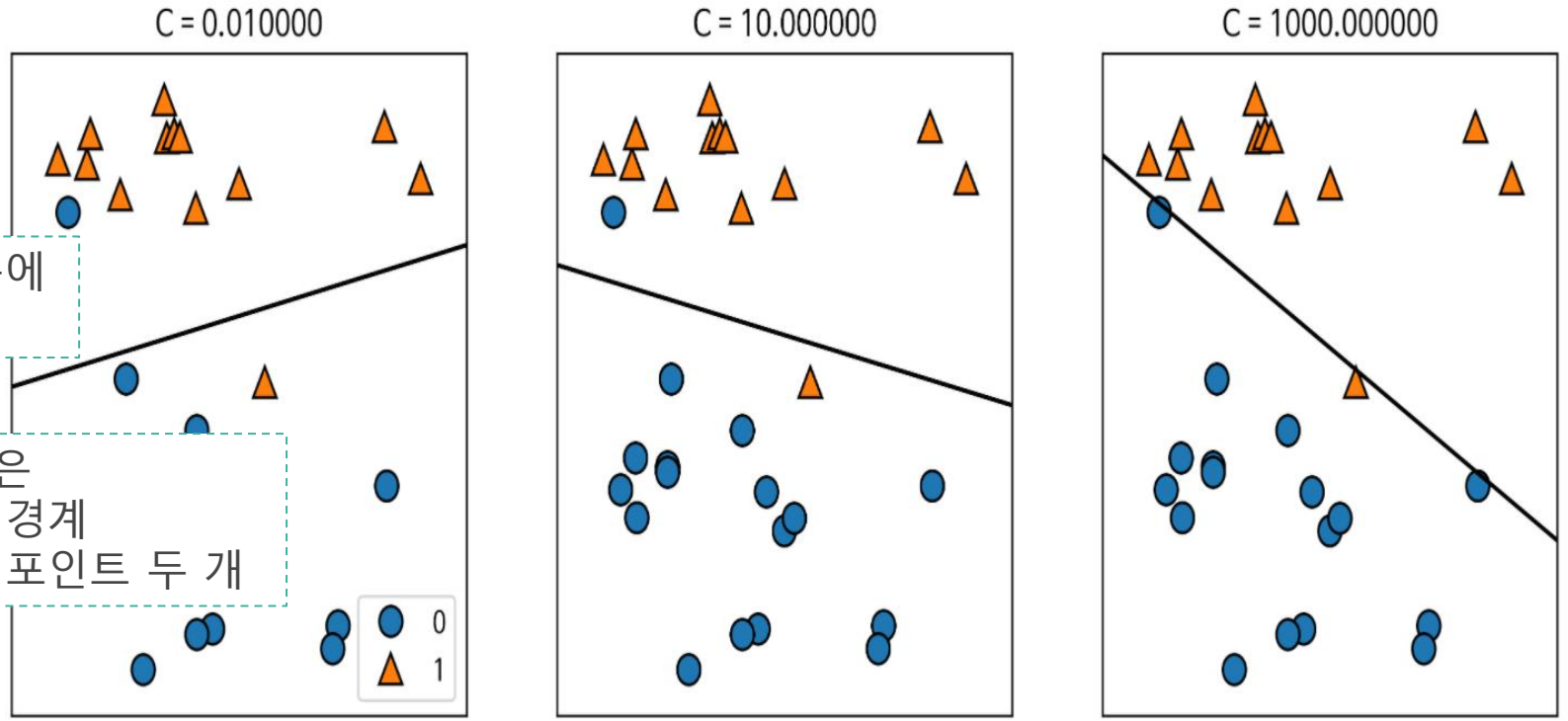
규제의 강도를 결정하는 매개변수는 C
→ C의 값이 높아지면 규제가 감소

▲ **그림 2-15** forge 데이터셋에 기본 매개변수를 사용해 만든 선형 SVM과 로지스틱 회귀 모델의 결정 경계

SECTION 2.3 지도 학습 알고리즘

◦ 분류형 선형 모델 – 로지스틱 회귀, 서포트 벡터 머신

- LogitsticRegression과 LinearSVC에서 규제의 강도를 결정하는 매개변수는 C 설정에 따른 그래프 비교
  - 알고리즘은 C의 값이 낮아지면 데이터 포인트 중 다수에 맞추려고 함
  - C의 값을 높이면 개개의 데이터 포인트를 정확히 분류하려고 노력할 것



아주 작은 C 값 때문에
규제가 많이 적용

규제가 강해진 모델은
수평에 가까운 결정 경계
잘못 분류한 데이터 포인트 두 개

모든 데이터 포인트를
정확하게 분류하려 함
클래스의 전체적인
배치를 잘 파악 못함
과대적합

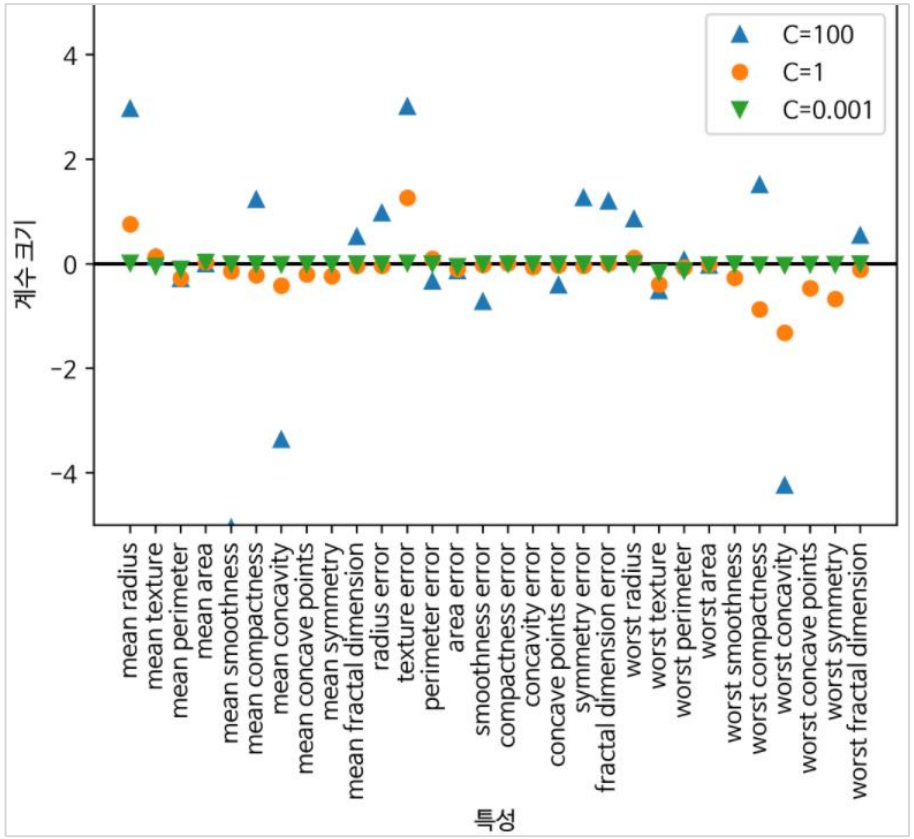
▲ **그림 2-16** forge 데이터셋에 각기 다른 C 값으로 만든 선형 SVM 모델의 결정 경계

SECTION 2.3 지도 학습 알고리즘

◦ 분류형 선형 모델 – 로지스틱 회귀

▪ 유방암 데이터셋을 사용한 로지스틱 회귀LogisticRegression 성능 평가

- 규제의 강도를 결정하는 매개변수 C 값 설정에 따른 유방암 데이터셋을 사용한 성능 평가 비교




```
logreg = LogisticRegression(max_iter=5000).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: 0.958
테스트 세트 점수: 0.958
```

훈련 세트와 테스트 세트의 성능이 매우 비슷
→ 과소적합

```
logreg100 = LogisticRegression(C=100, max_iter=5000).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: 0.984
테스트 세트 점수: 0.972
```

복잡도가 높은 모델일수록 성능이 좋음

```
logreg001 = LogisticRegression(C=0.01, max_iter=5000).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: 0.953
테스트 세트 점수: 0.951
```

이미 과소적합된 모델 → 강한 규제
→ 성능 낮아짐

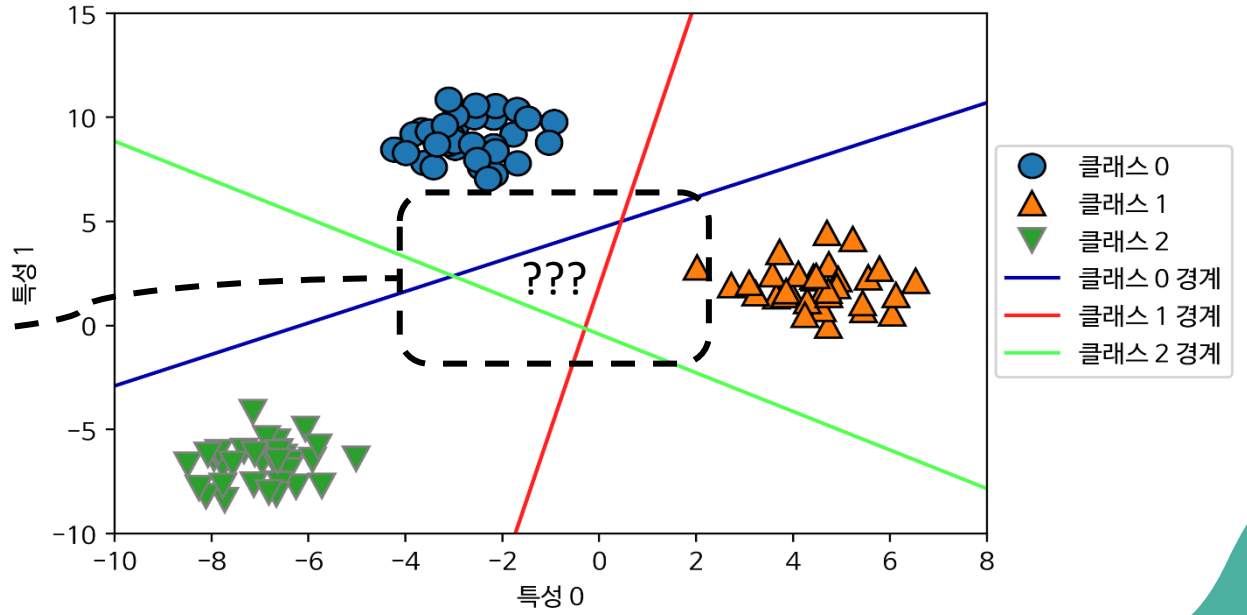
▲ **그림 2-17** 유방암 데이터셋에 각기 다른 C 값을 사용하여 만든 로지스틱 회귀의 계수

SECTION 2.3 지도 학습 알고리즘

◦ 분류형 선형 모델 – 다중 클래스 분류용 선형 모델

- (로지스틱 회귀만 제외하고) 많은 선형 분류 모델은 태생적으로 이진 분류만을 지원
  - 다중 클래스multiclass를 지원하지 않음
  - 이진 분류 알고리즘을 다중 클래스 분류 알고리즘으로 확장하는 보편적인 기법은 일대다one-vs.-rest 방법
  - 일대다 방식은 각 클래스를 다른 모든 클래스와 구분하도록 이진 분류 모델을 학습
    → 결국 클래스의 수만큼 이진 분류 모델이 만들어짐
    → 예측을 할 때 이렇게 만들어진 모든 이진 분류기가 작동
    → 가장 높은 점수를 내는 분류기의 클래스를 예측값으로 선택
  - LinearSVC 분류기를 활용하여
    세 개의 이진 분류기가 만드는 경계를 시각화

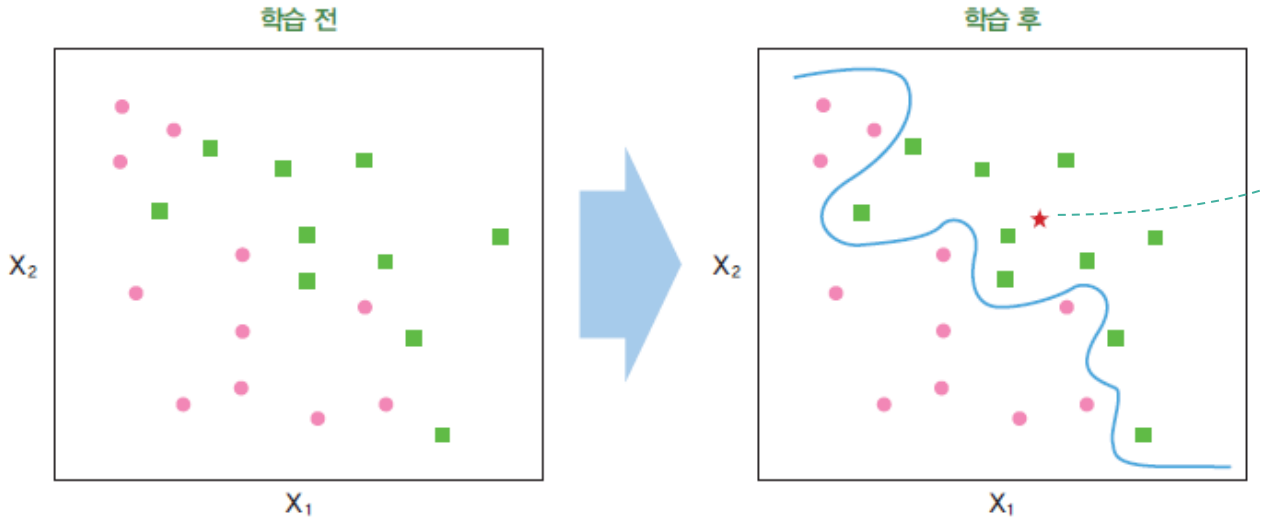
분류 공식의 결과가 가장 높은 클래스
즉 가장 가까운 직선의 클래스



SECTION 2.3 지도 학습 알고리즘

◦ 분류형 선형 모델 – 서포트 벡터 머신(support vector machine, SVM)

- 데이터셋의 여러 속성을 나타내는 데이터프레임의 각 열은 열 백터 형태로 구현 됨
  - 열 벡터들이 각각 고유의 축을 갖는 벡터 공간 만듬 → 분석 대상이 되는 개별 관측값은 모든 속성(열벡터) 관한 값을 해당 축의 좌표로 표시하여 벡터 공간에서 위치를 나타냄
  - 속성이 2개 존재하는 데이터 셋은 2차원 평면 공간 좌표, 속성 3개이면 3차원, 속성 4새이면 고차원 벡터 공간
  - SVM 모형은 벡터 공간에 위치한 훈련 데이터의 좌표와 각 데이터가 어떤 분류 값을 가져야 하는지 정답을 입력 받아 학습 → 같은 분류 값을 갖는 데이터끼리 간은 공간에 위치하도록 함



새로운 데이터에 대해서도
어느 공간에 위치 하는지 분류 가능

SVM 모형은 학습을 통해 벡터 공간을
나누는 경계를 찾음

새로운 데이터는 백터 공간에서
녹색 사각형 영역에 존재 함

SECTION 2.3 지도 학습 알고리즘

◦ 분류형 선형 모델 – 서포트 벡터 머신 (Support Vector Machine, SVM) 예제

▪ KNeighborsClassifier 타이타닉 생존자 예측 따른 성능 평가

- Seaborn에서 제공하는 titanic 데이터셋 사용
- 타이타닉 데이터 전처리 (원핫인코딩 - 범주형 데이터를 모형이 인식할 수 있도록 숫자형으로 변환)
- 데이터 셋 분리(훈련셋, 테스트셋)
  → sklearn SVC(Support Vector Classification) 메소드 radial basis function (RBF) 커널 사용하여 모델 생성

모형 성능 평가

```
[20] from sklearn import metrics
     svm_matrix = metrics.confusion_matrix(y_test, y_hat)
     print(svm_matrix)
```

```
[[120   5]
 [ 35  55]]
```

	precision	recall	f1-score	support
0	0.77	0.96	0.86	125
1	0.92	0.61	0.73	90
accuracy			0.81	215
macro avg	0.85	0.79	0.80	215
weighted avg	0.83	0.81	0.81	215

```
[28] print("테스트 세트 정확도: {:.2f}%".format(svm_model.score(X_test, y_test)*100))
```

```
테스트 세트 정확도: 81.40%
```

타이타닉 생존자 예측 모델 중 제일 높은 예측 정확도 확인

SECTION 2.3 지도 학습 알고리즘

◦ 선형 모델 – 장단점과 매개변수

- 선형 모델의 주요 매개변수는 회귀 모델에서는 alpha였고 LinearSVC와 LogisticRegression에서는 C
  - alpha 값이 클수록, C 값이 작을수록 모델이 단순해짐 (특별히 회귀 모델에서 이 매개변수를 조정하는 일이 매우 중요)
  - 보통 C와 alpha는 로그 스케일로 최적치를 정함
- L1 규제를 사용할지 L2 규제를 사용할지를 정해야 함
  - 중요한 특성이 많지 않다고 생각하면 L1 규제를 사용
  - 그렇지 않으면 기본적으로 L2 규제를 사용
  - L1 규제는 모델의 해석이 중요한 요소일 때도 사용할 수 있음
  - L1 규제는 몇 가지 특성만 사용하므로 해당 모델에 중요한 특성이 무엇이고 효과가 어느 정도인지 설명하기 쉬움
- 선형 모델은 학습 속도가 빠르고 예측
- 회귀와 분류에서 본 공식을 사용해 예측이 어떻게 만들어지는지 비교적 쉽게 이해할 수 있다는 것
  - 계수의 값들이 왜 그런지 명확하지 않을 때가 종종 있음
  - 특히 데이터셋의 특성들이 서로 깊게 연관되어 있을 때 그렇고, 이럴 땐 계수를 분석하기가 매우 어려움
  - 선형 모델은 샘플에 비해 특성이 많을 때 잘 작동
  - 다른 모델로 학습하기 어려운 매우 큰 데이터셋에도 선형 모델을 많이 사용 but 저차원 데이터셋 x

SECTION 2.3 지도 학습 알고리즘

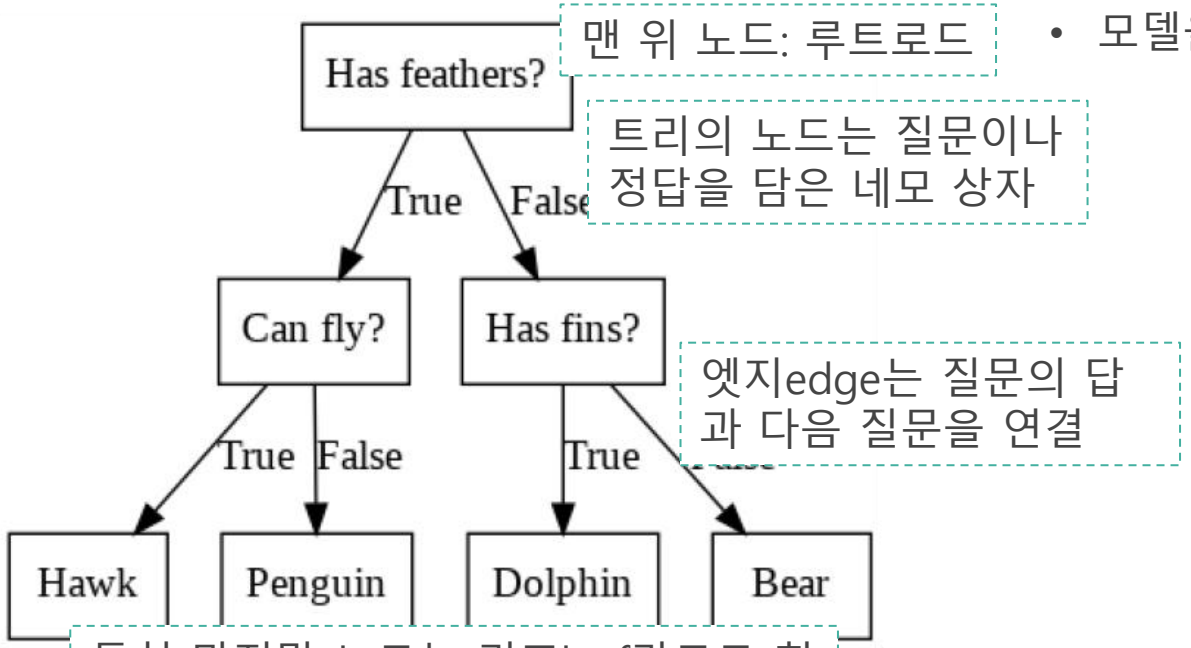
◦ 나이브 베이즈 분류기

- 나이브 베이즈naive bayes 분류기는 앞 절의 선형 모델과 매우 유사
  - LogisticRegression이나 LinearSVC 같은 선형 분류기보다 훈련 속도가 빠른 편
  - 일반화 성능이 조금 뒤쳐짐
- 나이브 베이즈 분류기가 효과적인 이유
  - 특성을 개별로 취급해 파라미터를 학습하고 각 특성에서 클래스별 통계를 단순하게 취합
  - scikit-learn에 구현된 나이브 베이즈 분류기는 GaussianNB, BernoulliNB, MultinomialNB 총 3가지
  - BernoulliNB 분류기는 각 클래스의 특성 중 0이 아닌 것이 몇 개인지 셈
  - MultinomialNB는 클래스별로 특성의 평균을 계산, GaussianNB는 클래스별로 각 특성의 표준편차와 평균을 저장
- 장단점과 매개변수
  - GaussianNB는 대부분 매우 고차원인 데이터셋에 사용, 다른 두 나이브 베이즈 모델은 텍스트 같은 희소한 데이터를 카운트하는 데 사용
  - 훈련과 예측 속도가 빠르며 훈련 과정을 이해하기 쉽고, 희소한 고차원 데이터에서 잘 작동
  - 비교적 매개변수에 민감하지 않음
  - 학습 시간이 너무 오래 걸리는 매우 큰 데이터셋에 적용

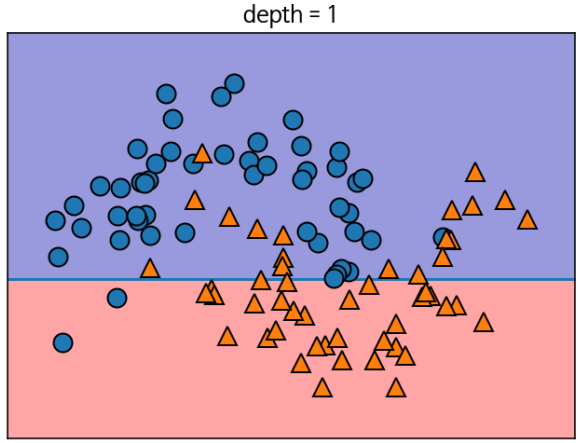
SECTION 2.3 지도 학습 알고리즘

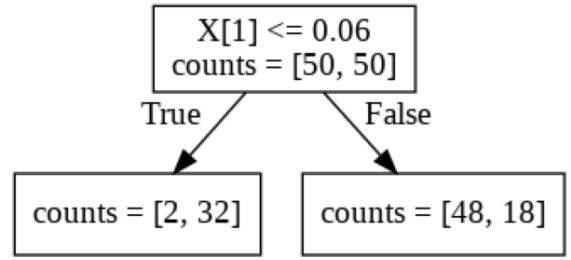
◦ 결정 트리

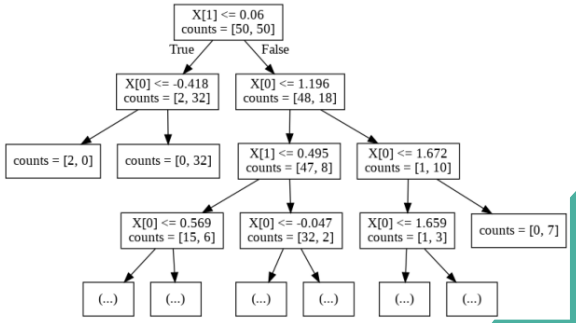
- **결정 트리**decision tree는 분류와 회귀 문제에 널리 사용하는 모델
  - 기본적으로 결정 트리는 결정에 다다르기 위해 예/아니오 질문을 이어 나가면서 학습
  - 모델을 직접 만드는 대신 지도 학습 방식으로 데이터로부터 학습



▲ **그림 2-22** 몇 가지 동물들을 구분하기 위한 결정 트리







SECTION 2.3 지도 학습 알고리즘

◦ 결정 트리 - 결정 트리의 복잡도 제어하기

▪ 결정 트리의 복잡도 제어하기

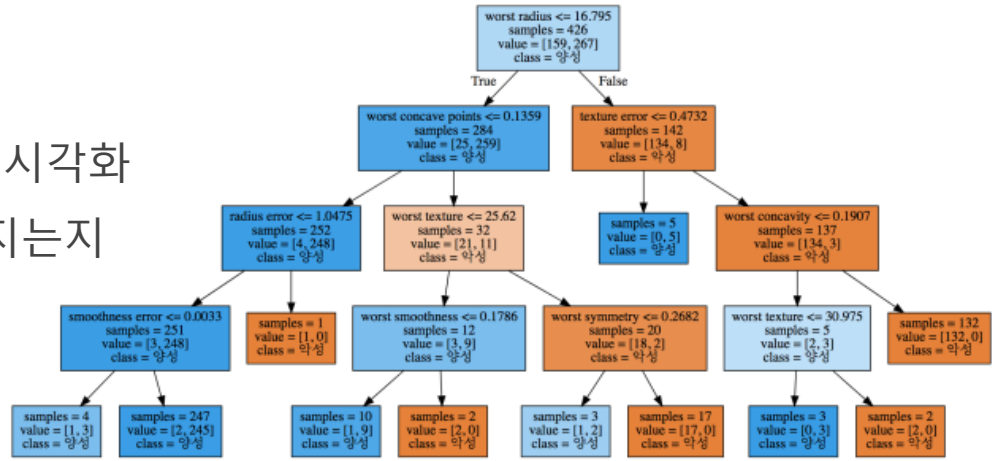
- 일반적으로 트리 만들기를 모든 리프 노드가 순수 노드가 될 때까지 진행하면 모델이 매우 복잡해지고 훈련 데이터에 과대적합 됨 → 순수 노드로 이루어진 트리는 훈련 세트에 100% 정확하게 맞는다는 의미

▪ 과대적합을 막는 전략은 크게 두 가지

- 트리 생성을 일찍 중단하는 전략(**사전 가지치기**pre-pruning)
- 트리를 만든 후 데이터 포인트가 적은 노드를 삭제하거나 병합하는 전략 (**사후 가지치기**post-pruning 또는 그냥 **가지치기**pruning)
- scikit-learn에서 결정 트리는 DecisionTreeRegressor와 DecisionTreeClassifier에 구현
- scikit-learn은 사전 가지치기만 지원

▪ 결정 트리 분석

- 트리 모듈의 export_graphviz 함수를 이용해 트리를 시각화
- 트리를 시각화하면 알고리즘의 예측이 어떻게 이뤄지는지 잘 이해할 수 있으며 비전문가에게 머신러닝 알고리즘을 설명하기에 좋음
- 단점 사전 가지치기를 해도 과대적합 경향 있음

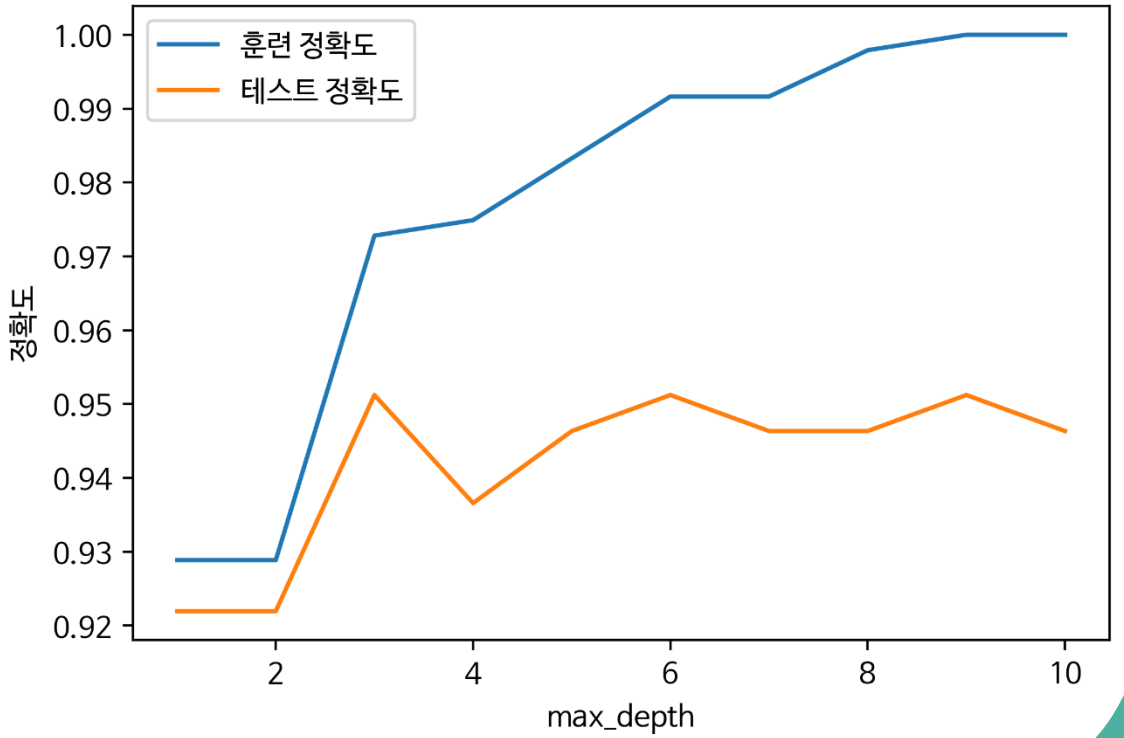


SECTION 2.3 지도 학습 알고리즘

◦ 결정 트리 - DecisionTreeClassifier 예제

- DecisionTreeClassifier 사용하여 Breast Cancer 데이터셋 사용하여 유방암 (1), 정상(0) 예측 및 성능평가
  - Breast Cancer 데이터셋 (출처: UCI ML Repository)
  - https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
  - Breast Cancer 데이터 전처리 (데이터 타입 변환 및 결측치 제거, 설명 변수 데이터를 정규화)
  - 데이터 셋 분리(훈련셋, 테스트셋)
    → Decision Tree 분류 모형 - sklearn 사용
    → 속성으로 criterion='entropy', max_depth=5 적용
  - max_depth 설정 값 변경하며, 최적의 모델 찾기

	precision	recall	f1-score	support
2	0.98	0.94	0.96	131
4	0.90	0.96	0.93	74
accuracy			0.95	205
macro avg	0.94	0.95	0.94	205
weighted avg	0.95	0.95	0.95	205



SECTION 2.3 지도 학습 알고리즘

◦ 결정 트리의 앙상블 - 랜덤 포레스트

- **앙상블**ensemble은 여러 머신러닝 모델을 연결하여 더 강력한 모델을 만드는 기법
  - **랜덤 포레스트**random forest , **그래디언트 부스팅**gradient boosting 결정 트리는 둘 다 모델을 구성하는 기본 요소로 결정 트리를 사용
  - 결정 트리의 주요 단점은 훈련 데이터에 과대적합되는 경향 → 랜덤 포레스트는 이 문제를 회피할 수 있는 방법
- 랜덤 포레스트 아이디어
  - 각 트리는 비교적 예측을 잘 할 수 있지만 데이터의 일부에 과대적합하는 경향을 가진다는 데 기초
  - 잘 작동하되 서로 다른 방향으로 과대적합된 트리를 많이 만들면 그 결과를 평균냄으로써 과대적합된 양을 줄임
  - 트리 모델의 예측 성능이 유지되면서 과대적합이 줄어드는 것이 수학적으로 증명
  - 랜덤 포레스트는 이름에서 알 수 있듯이 트리들이 달라지도록 트리 생성 시 무작위성을 주입
- 랜덤 포레스트에서 트리를 랜덤하게 만드는 방법은 두 가지
  - 트리를 만들 때 사용하는 데이터 포인트를 무작위로 선택하는 방법
  - 분할 테스트에서 특성을 무작위로 선택하는 방법

SECTION 2.3 지도 학습 알고리즘

◦ 결정 트리의 앙상블 - 랜덤 포레스트 구축

- 랜덤 포레스트 모델을 만들려면 생성할 트리의 개수를 정해야함
  - RandomForestRegressor나 RandomForestClassifier의 n_estimators 매개변수 설정
  - 트리를 만들기 위해 먼저 데이터의 **부트스트랩 샘플**bootstrap sample을 생성
  - n_samples개의 데이터 포인트 중에서 무작위로 데이터를 n_samples 횟수만큼 반복 추출
  - 이렇게 만든 데이터셋으로 결정 트리를 만듬
  - 각 노드에서 특성의 일부만 사용하기 때문에 트리의 각 분기는 각기 다른 특성 부분 집합을 사용
  - 핵심 매개변수는 max_features - max_features=1로 설정하면 트리의 분기는 테스트할 특성을 고를 필요가 없게 되며 그냥 무작위로 선택한 특성의 임계값을 찾음
  - 결국 max_features 값을 크게 하면 랜덤 포레스트의 트리들은 매우 비슷해지고 가장 두드러진 특성을 이용해 데이터에 잘 맞춰질 것
- 회귀와 분류에 있어서 랜덤 포레스트는 현재 가장 널리 사용되는 머신러닝 알고리즘
  - 유념할 점은 랜덤 포레스트는 이름 그대로 랜덤
  - 랜덤 포레스트는 텍스트 데이터 같이 매우 차원이 높고 희소한 데이터에는 잘 작동하지 않음

SECTION 2.3 지도 학습 알고리즘

◦ 결정 트리의 앙상블 - 그래디언트 부스팅 회귀 트리

- 그래디언트 부스팅 회귀 트리는 여러 개의 결정 트리를 묶어 강력한 모델을 만드는 또 다른 앙상블 방법
  - 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리를 만듦
  - 기본적으로 그래디언트 부스팅 회귀 트리에는 무작위성이 없음 → 대신 강력한 사전 가지치기가 사용
  - 그래디언트 부스팅 트리는 보통 하나에서 다섯 정도의 깊지 않은 트리를 사용하므로 메모리를 적게 사용하고 예측도 빠름
  - scikit-learn에 구현된 GradientBoostingClassifier 사용
- 그래디언트 부스팅의 근본 아이디어
  - 얕은 트리 같은 간단한 모델(**약한 학습기**weak learner라고도 합니다)을 많이 연결하는 것
  - 각각의 트리는 데이터의 일부에 대해서만 예측을 잘 수행할 수 있어서 트리가 많이 추가될수록 성능이 좋아짐
- 그래디언트 부스팅 트리는 머신러닝 경연 대회에서 우승을 많이 차지하였고 업계에서도 널리 사용
  - 랜덤 포레스트보다는 매개변수 설정에 조금 더 민감하지만 잘 조정하면 더 높은 정확도를 제공
  - 중요한 매개변수는 이전 트리의 오차를 얼마나 강하게 보정할 것인지를 제어하는 learning_rate
  - 학습률이 크면 트리는 보정을 강하게 하기 때문에 복잡한 모델을 만듦
  - n_estimators 값을 키우면 앙상블에 트리가 더 많이 추가되어 모델의 복잡도가 커지고 훈련 세트에서의 실수를 바로잡을 기회가 더 많아짐

SECTION 2.3 지도 학습 알고리즘

- 머신러닝 알고리즘의 작동 방식 학습
  - 데이터로부터 어떻게 학습하고 예측하는가?
  - 모델의 복잡도가 어떤 역할을 하는가?
  - 알고리즘이 모델을 어떻게 만드는가?
  - 모델들의 장단점을 평가하고 어떤 데이터가 잘 들어맞을지 살펴보기
  - 매개변수와 옵션의 의미 학습
- 예제에 사용할 데이터셋
- k-최근접 이웃
- 선형 모델
- 나이브 베이즈 분류기
- 결정 트리
- 결정 트리의 앙상블

SECTION 2.5 요약 및 정리

- 각 데이터 모델의 특징
  - 최근접 이웃: 작은 데이터셋일 경우, 기본 모델로서 좋고 설명하기 쉬움
  - 선형 모델: 첫 번째로 시도할 알고리즘. 대용량 데이터셋 가능. 고차원 데이터에 가능
  - 나이브 베이즈: 분류만 가능. 선형 모델보다 훨씬 빠름. 대용량 데이터셋과 고차원 데이터에 가능. 선형 모델보다 덜 정확함
  - 결정 트리: 매우 빠름. 데이터 스케일 조정이 필요 없음. 시각화하기 좋고 설명하기 쉬움
  - 랜덤 포레스트: 결정 트리 하나보다 거의 항상 좋은 성능을 냄. 매우 안정적이고 강력함. 데이터 스케일 조정 필요 없음. 고차원 희소 데이터에는 잘 안 맞음
  - 그레이디언트 부스팅 결정 트리: 랜덤 포레스트보다 조금 더 성능이 좋음. 랜덤 포레스트보다 학습은 느리나 예측은 빠르고 메모리를 조금 사용. 랜덤 포레스트보다 매개변수 튜닝이 많이 필요함
  - 서포트 벡터 머신: 비슷한 의미의 특성으로 이뤄진 중간 규모 데이터셋에 잘 맞음. 데이터 스케일 조정 필요. 매개변수에 민감
  - 신경망: 특별히 대용량 데이터셋에서 매우 복잡한 모델을 만들 수 있음. 매개변수 선택과 데이터 스케일에 민감. 큰 모델은 학습이 오래 걸림