

A faint, abstract network graph background consisting of numerous small, semi-transparent blue dots connected by thin lines, forming a complex web-like structure.

# 3장

# 머신 러닝

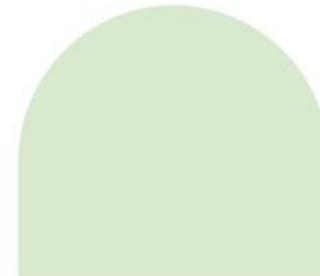
# 핵심 알고리즘

# 3장 머신 러닝 핵심 알고리즘

---

3.1 지도 학습

3.2 비지도 학습



## 3.1 지도 학습

---

## 3.1 지도 학습

### ● 지도 학습

- 지도 학습은 정답(레이블(label))을 컴퓨터에 미리 알려 주고 데이터를 학습시키는 방법
- 지도 학습에는 분류와 회귀가 있음
- 분류(classification)는 주어진 데이터를 정해진 범주에 따라 분류
- 회귀(regression)는 데이터들의 특성(feature)을 기준으로 연속된 값을 그래프로 표현하여 패턴이나 트렌드를 예측할 때 사용

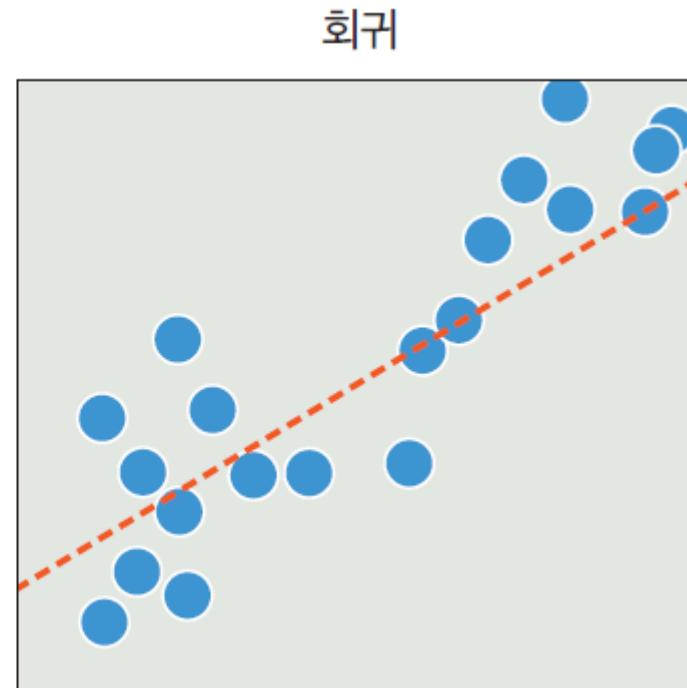
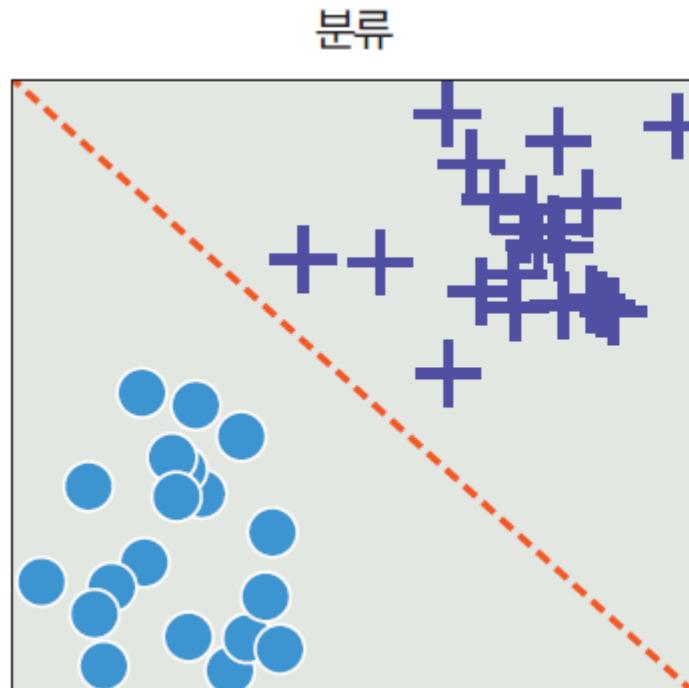
## 3.1 지도 학습

### ▼ 표 3-1 분류와 회귀 차이

구분	분류	회귀
데이터 유형	이산형 데이터	연속형 데이터
결과	훈련 데이터의 레이블 중 하나를 예측	연속된 값을 예측
예시	학습 데이터를 A · B · C 그룹 중 하나로 매핑 예 스팸 메일 필터링	결과값이 어떤 값이든 나올 수 있음 예 주가 분석 예측

## 3.1 지도 학습

▼ 그림 3-1 분류와 회귀



## 3.1 지도 학습

### ● K-최근접 이웃

#### ▼ 표 3-2 K-최근접 이웃을 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터에 대한 분류
언제 사용하면 좋을까?	K-최근접 이웃은 직관적이며 사용하기 쉽기 때문에 초보자가 쓰면 좋습니다. 또한, 훈련 데이터를 충분히 확보할 수 있는 환경에서 사용하면 좋습니다.

## 3.1 지도 학습

### ● K-최근접 이웃

- K-최근접 이웃(K-nearest neighbor)은 새로운 입력(분류되지 않은 검증 데이터)을 받았을 때 기존 클러스터에서 모든 데이터와 인스턴스(instance) 기반 거리를 측정한 후 가장 많은 속성을 가진 클러스터에 할당하는 분류 알고리즘
- 즉, 과거 데이터를 사용하여 미리 분류 모형을 만드는 것이 아니라, 과거 데이터를 저장해 두고 필요할 때마다 비교를 수행하는 방식
- K 값의 선택에 따라 새로운 데이터에 대한 분류 결과가 달라질 수 있음에 유의해야 함

## 3.1 지도 학습

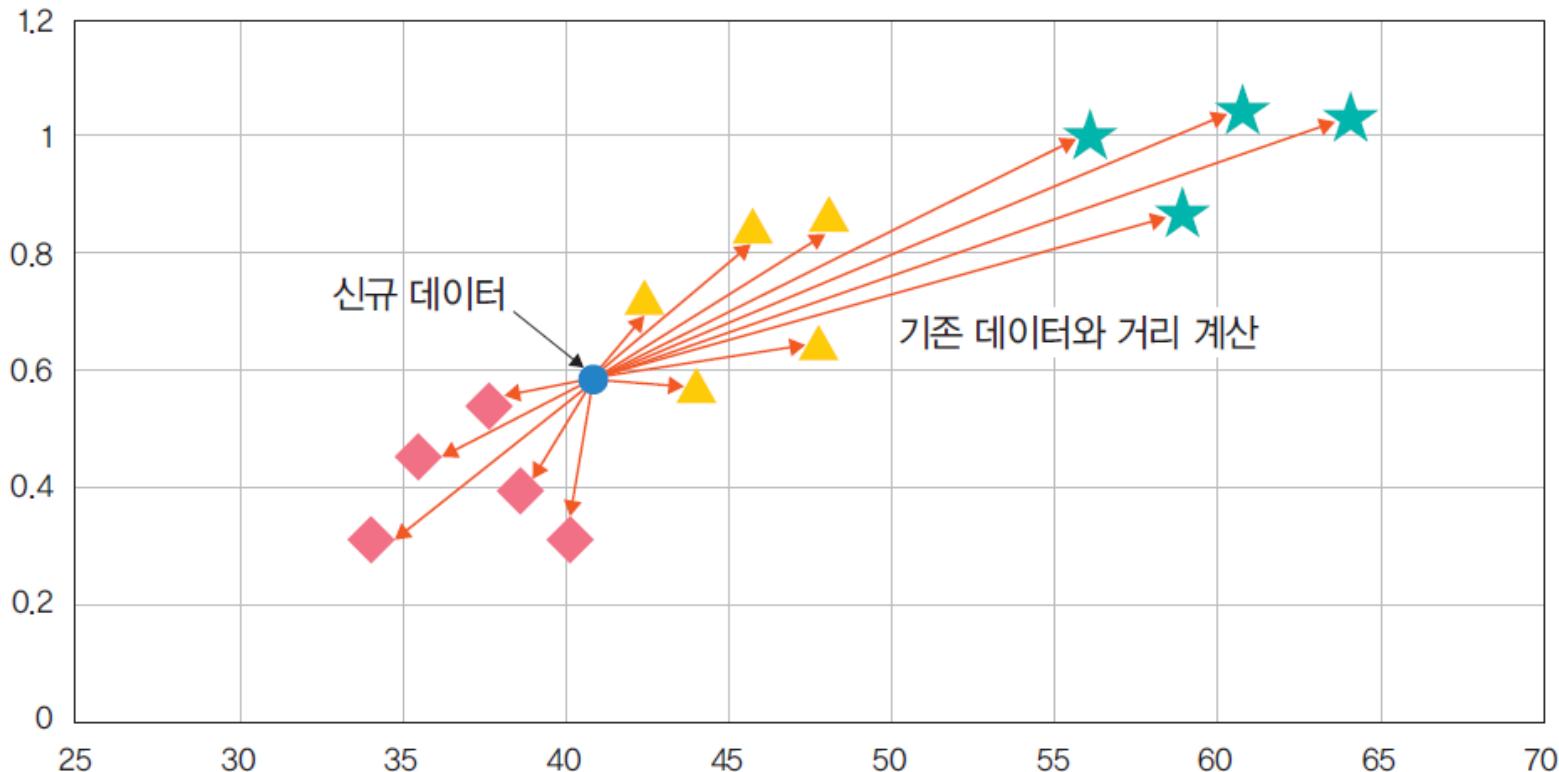
### ● K-최근접 이웃

- 다음 그림과 같이 네모, 세모, 별 모양의 클러스터로 구성된 데이터셋이 있다고 하자
- 신규 데이터인 동그라미가 유입되었다면 기존 데이터들과 하나씩 거리를 계산하고 거리상으로 가장 가까운 데이터 다섯 개( $K=5$ )를 선택하여 해당 클러스터에 할당



## 3.1 지도 학습

▼ 그림 3-2 K-최근접 이웃

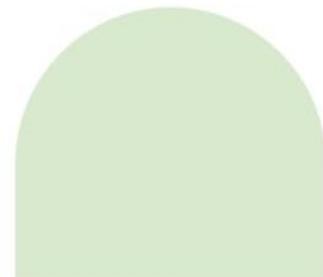


## 3.1 지도 학습

### ● K-최근접 이웃

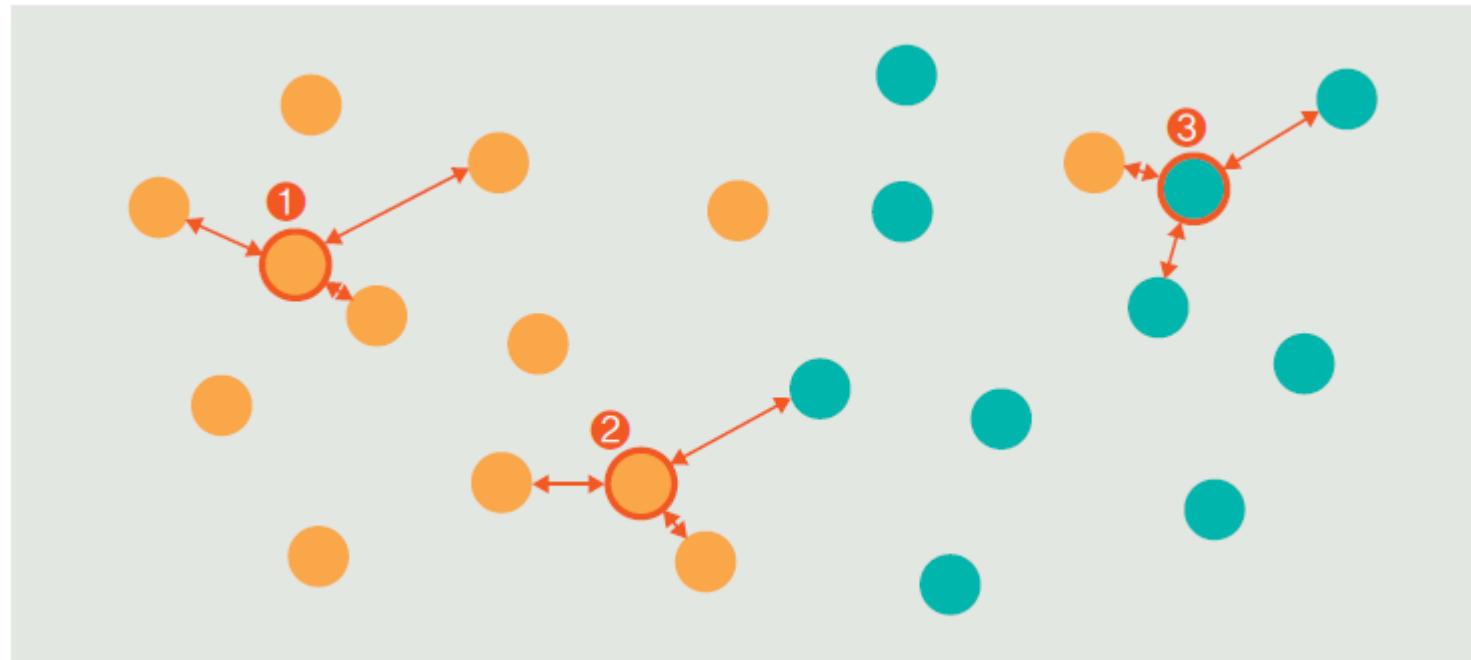
- 예를 들어 다음 그림과 같이 새로운 입력 데이터(빨간색 외각선 원)가 세 개 있을 때 새로운 입력에 대한 분류를 진행해 보자( $K=3$ )

- 새로운 입력 ① : 주변 범주 세 개가 주황색이므로 주황색으로 분류
- 새로운 입력 ② : 주변 범주 두 개가 주황색, 한 개가 녹색이므로 주황색으로 분류
- 새로운 입력 ③ : 주변 범주 두 개가 녹색, 한 개가 주황색이므로 녹색으로 분류



## 3.1 지도 학습

▼ 그림 3-3 K-최근접 이웃 학습 절차

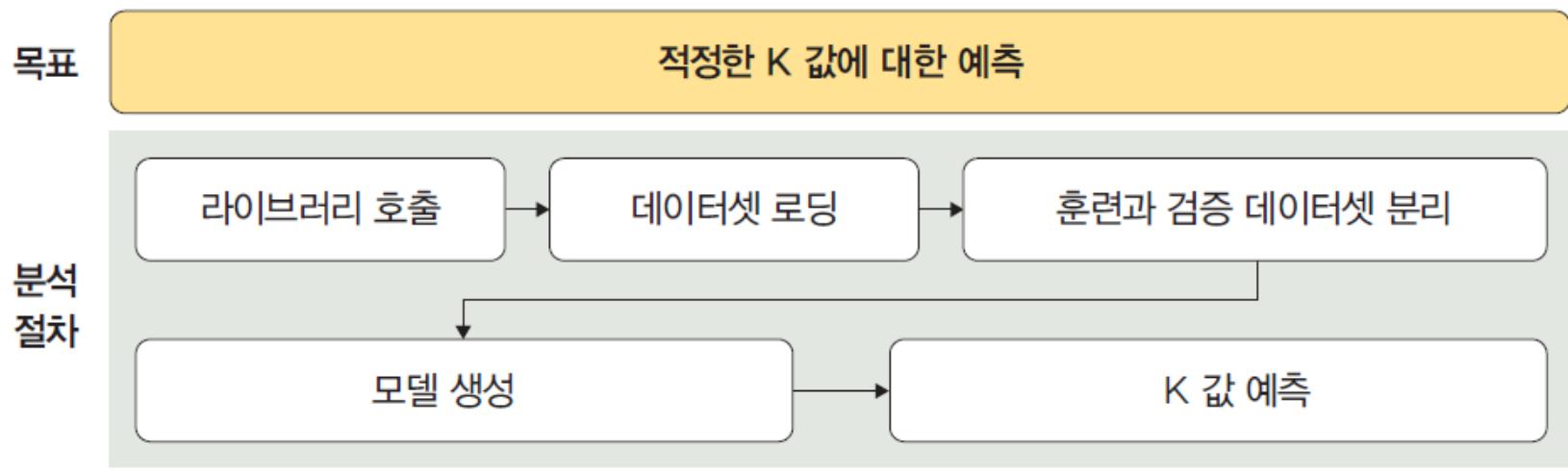


# 3.1 지도 학습

## ● K-최근접 이웃

- 이제 코드에서 구체적으로 확인해 보자
- 예제 목표는 붓꽃에 대한 분류
- 참고로 머신 러닝 코드는 심층 신경망이 필요하지 않기 때문에 사이킷런(scikit-learn)을 이용
- 다음 과정으로 K 값을 예측할 것

### ▼ 그림 3-4 K-최근접 이웃 예제



# 3.1 지도 학습

## ● K-최근접 이웃

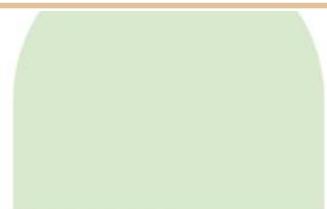
- 먼저 필요한 라이브러리를 호출하고 데이터를 준비
- 데이터는 내려받은 예제 파일의 data 폴더에 있는 iris.data 파일을 사용
- iris.data 데이터 경로는 자신의 실습 환경에 맞게 수정해서 사용할 수 있음

### 코드 3-1 라이브러리 호출 및 데이터 준비

```
import numpy as np ----- 벡터 및 행렬의 연산 처리를 위한 라이브러리
import matplotlib.pyplot as plt ----- 데이터를 차트나 플롯(plot)으로 그려 주는 라이브러리
import pandas as pd ----- 데이터 분석 및 조작을 위한 라이브러리
from sklearn import metrics ----- 모델 성능 평가

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class'] ----- 데이터셋에 열(column) 이름 할당

dataset = pd.read_csv('../chap3/data/iris.data', names=names) ----- 데이터를 판다스 데이터프레임(dataframe)에 저장, 경로는 수정해서 진행
```



# 3.1 지도 학습

## ● K-최근접 이웃

- 준비한 데이터를 전처리하고 훈련과 검증 데이터셋으로 분리

코드 3-2 훈련과 검증 데이터셋 분리

```
X = dataset.iloc[:, :-1].values ----- 모든 행을 사용하지만 열(칼럼)은 뒤에서 하나를 뺀 값을 가져와서 X에 저장  
y = dataset.iloc[:, 4].values ----- 모든 행을 사용하지만 열은 앞에서 다섯 번째 값만 가져와서 y에 저장
```

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20) -----  
X, y를 사용하여 훈련과 검증 데이터셋으로 분리하며, 검증 세트의 비율은 20%만 사용  
from sklearn.preprocessing import StandardScaler  
s = StandardScaler() ----- 특성 스케일링(scaling), 평균이 0, 표준편차가 1이 되도록 변환  
X_train = s.transform(X_train) ----- 훈련 데이터를 스케일링 처리  
X_test = s.transform(X_test) ----- 검증 데이터를 스케일링 처리
```

# 3.1 지도 학습

## ● K-최근접 이웃

- 모델을 생성하고 훈련시킴

코드 3-3 모델 생성 및 훈련

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=50) ----- K=50인 K-최근접 이웃 모델 생성  
knn.fit(X_train, y_train) ----- 모델 훈련
```

- 다음은 모델 생성 및 훈련에 대한 출력 결과

KNeighborsClassifier(n\_neighbors=50)

# 3.1 지도 학습

## ● K-최근접 이웃

- 모델에 대한 정확도를 측정

코드 3-4 모델 정확도

```
from sklearn.metrics import accuracy_score
y_pred = knn.predict(X_test)
print("정확도: {}".format(accuracy_score(y_test, y_pred)))
```

- 그러면 다음 결과가 출력

정확도: 0.9333333333333333

- 참고로 정확도 실행 결과가 책 결과와 다를 수 있음
- train\_test\_split() 메서드는 데이터를 무작위로 분할하므로 코드를 실행할 때마다 정확도에 차이가 있음
- 여러 차례 실행한 후 평균을 찾는 것이 좋음

# 3.1 지도 학습

## ● K-최근접 이웃

- K=50일 때 예측 값이 약 93%로, 수치가 높음
- 그럼 이제 최적의 K 값을 구하고 그것에 대한 정확도를 살펴보자
- for 문을 이용하여 K 값을 1부터 10까지 순환하면서 최적의 K 값과 정확도를 찾음

코드 3-5 최적의 K 찾기

```
k = 10
acc_array = np.zeros(k)
for k in np.arange(1, k+1, 1): -----K는 1에서 10까지 값을 취함
    classifier = KNeighborsClassifier(n_neighbors=k).fit(X_train, y_train) -----
    y_pred = classifier.predict(X_test)                                     for 문을 반복하면서 K 값 변경
    acc = metrics.accuracy_score(y_test, y_pred)
    acc_array[k-1] = acc

max_acc = npamax(acc_array)
acc_list = list(acc_array)
k = acc_list.index(max_acc)
print("정확도", max_acc, "으로 최적의 k는", k+1, "입니다.")
```

## 3.1 지도 학습

### ● K-최근접 이웃

- 다음은 최적의 K와 그에 대한 정확도 결과

정확도 1.0 으로 최적의 k는 1 입니다.

- K 값이 50일 때 정확도가 93%였다면 K 값이 1일 때는 정확도가 100%로 높아졌음
- 이와 같이 K-최근접 이웃 알고리즘은 K 값에 따라 성능이 달라질 수 있으므로 초기 설정이 매우 중요함

# 3.1 지도 학습

## ● 서포트 벡터 머신

### ▼ 표 3-3 서포트 벡터 머신을 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터에 대한 분류
언제 사용하면 좋을까?	서포트 벡터 머신은 커널만 적절히 선택한다면 정확도가 상당히 좋기 때문에 정확도를 요구하는 분류 문제를 다룰 때 사용하면 좋습니다. 또한, 텍스트를 분류할 때도 많이 사용합니다.

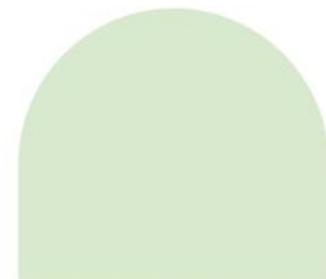
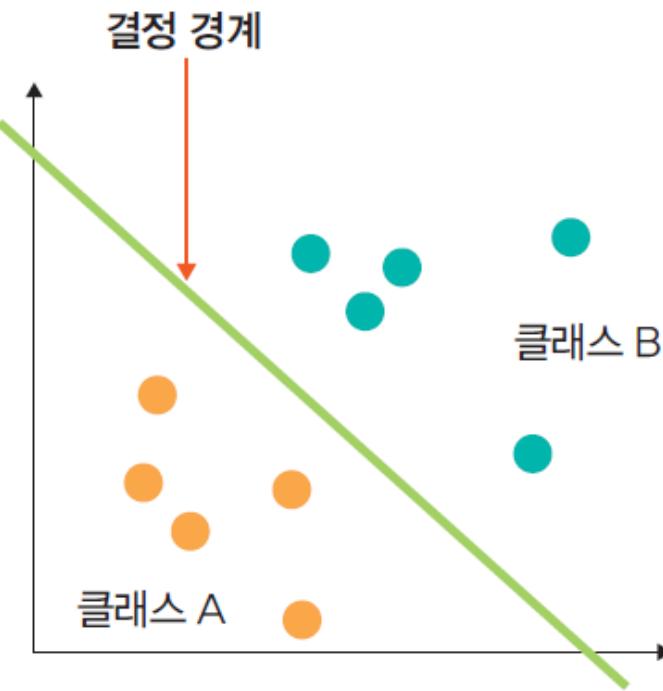
- 서포트 벡터 머신(Support Vector Machine, SVM)은 분류를 위한 기준선을 정의하는 모델
- 즉, 분류되지 않은 새로운 데이터가 나타나면 결정 경계(기준선)를 기준으로 경계의 어느 쪽에 속하는지 분류하는 모델
- 서포트 벡터 머신에서는 결정 경계를 이해하는 것이 중요함

## 3.1 지도 학습

### ● 서포트 벡터 머신

- 결정 경계는 데이터를 분류하기 위한 기준선
- 다음 그림과 같이 주황색 공과 녹색 공이 있을 때 이 공들을 색상별로 분류하기 위한 기준선이 결정 경계

▼ 그림 3-5 서포트 벡터 머신 결정 경계

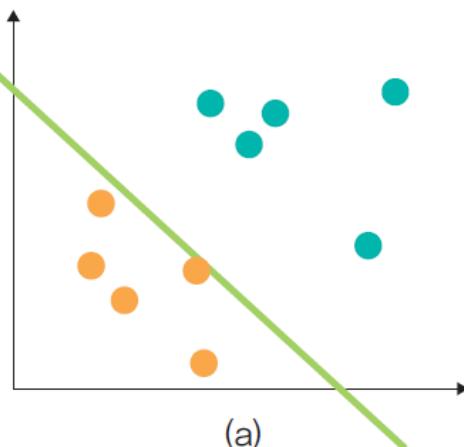


## 3.1 지도 학습

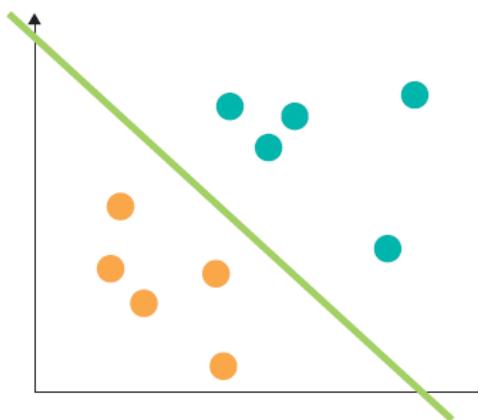
### ● 서포트 벡터 머신

- 그렇다면 결정 경계는 어디에 위치하면 가장 좋을까?
- 다음 그림의 (a)~(c) 중에서 (b)가 가장 안정적으로 보이지 않나?

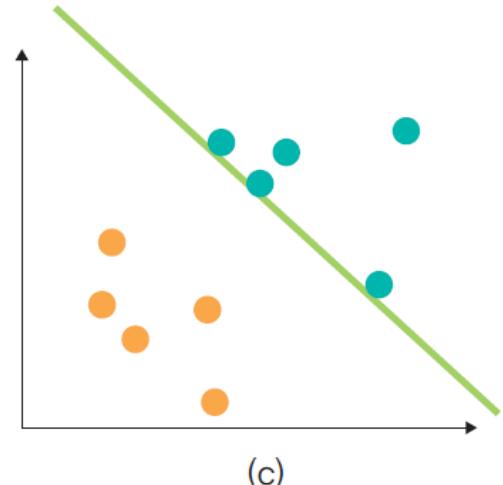
▼ 그림 3-6 서포트 벡터 머신 결정 경계의 위치 결정



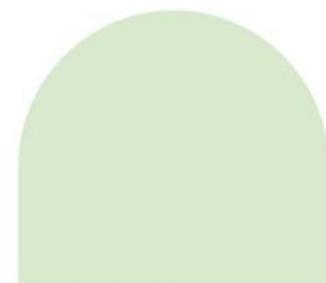
(a)



(b)



(c)



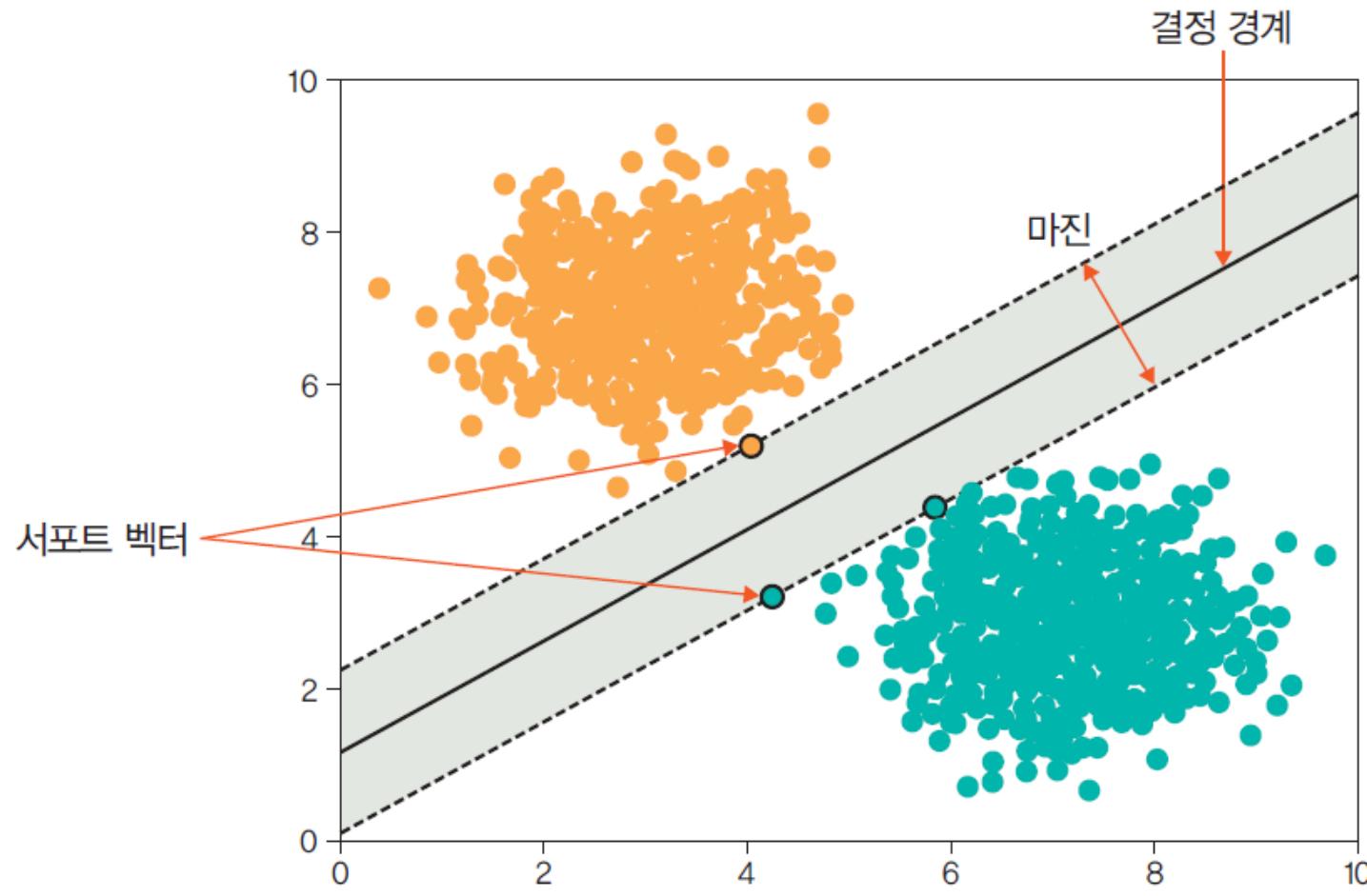
## 3.1 지도 학습

### ● 서포트 벡터 머신

- 결정 경계는 데이터가 분류된 클래스에서 최대한 멀리 떨어져 있을 때 성능이 가장 좋음
- 서포트 벡터 머신을 이해하려면 결정 경계 외에도 마진이라는 개념을 이해해야 함
- 마진(margin)은 결정 경계와 서포트 벡터 사이의 거리를 의미
- 서포트 벡터(support vector)는 결정 경계와 가까이 있는 데이터들을 의미
- 이 데이터들이 경계를 정의하는 결정적인 역할을 한다고 할 수 있음
- 즉, 정리하면 최적의 결정 경계는 마진을 최대로 해야 함

## 3.1 지도 학습

▼ 그림 3-7 서포트 벡터 머신의 서포트 벡터



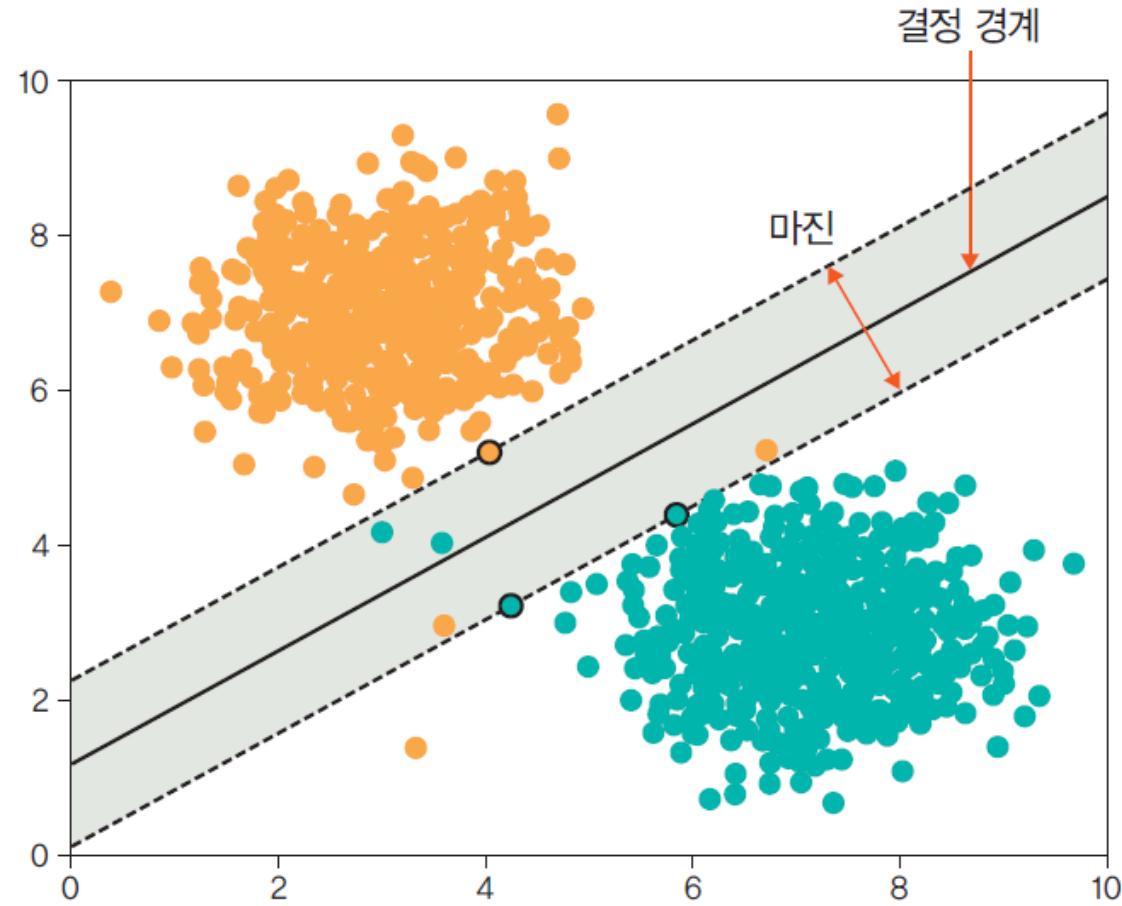
## 3.1 지도 학습

### ● 서포트 벡터 머신

- 서포트 벡터 머신은 데이터들을 올바르게 분리하면서 마진 크기를 최대화해야 하는데, 결국 이상치(outlier)를 잘 다루는 것이 중요함
- 이때 이상치를 허용하지 않는 것을 하드 마진(hard margin)이라고 하며, 어느 정도의 이상치들이 마진 안에 포함되는 것을 허용한다면 소프트 마진(soft margin)이라고 함
- 그림 3-7이 이상치를 허용하지 않는 하드 마진이라면, 그림 3-8은 이상치를 허용하는 소프트 마진

## 3.1 지도 학습

▼ 그림 3-8 서포트 벡터 머신의 마진



# 3.1 지도 학습

## ● 서포트 벡터 머신

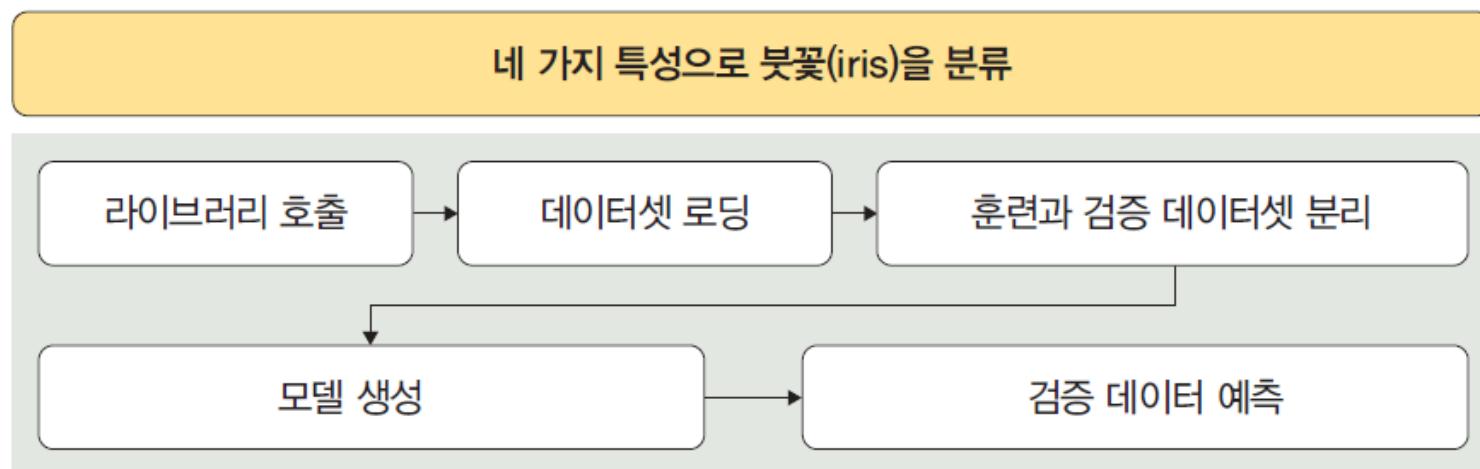
- 이제 코드로 서포트 벡터 머신을 자세히 살펴보자
- 서포트 벡터 머신의 예제도 붓꽃 분류로 진행해 보자
- 코드로 풀어 나가는 방법은 다르므로 잘 살펴보기 바람

▼ 그림 3-9 서포트 벡터 머신 예제

목표

네 가지 특성으로 붓꽃(iris)을 분류

분석  
절차



## 3.1 지도 학습

### ● 서포트 벡터 머신

- 먼저 훈련에 필요한 데이터를 로드하고 필요한 라이브러리를 호출

코드 3-6 라이브러리 호출

```
from sklearn import svm
from sklearn import metrics
from sklearn import datasets
from sklearn import model_selection
import tensorflow as tf
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' -----①
```

## 3.1 지도 학습

### ● 서포트 벡터 머신

- ① TF\_CPP\_MIN\_LOG\_LEVEL이라는 환경 변수를 사용하여 로깅을 제어(기본값은 0으로

모든 로그가 표시되며, INFO 로그를 필터링하려면 1, WARNING 로그를 필터링하려면 2,

ERROR 로그를 추가로 필터링하려면 3으로 설정)

- 환경 변수 값을 바꾸어 가면서 실행해 보는 것도 학습에 좋은 방법

## 3.1 지도 학습

## ● 서포트 벡터 머신

- 데이터셋을 불러와 훈련과 검증 데이터셋으로 분리

코드 3-7 iris 데이터를 준비하고 훈련과 검증 데이터셋으로 분리

```
iris = datasets.load_iris() ----- 사이킷런에서 제공하는 iris 데이터 호출  
X_train, X_test, y_train, y_test =  
model_selection.train_test_split(iris.data,  
                                iris.target,  
                                test_size=0.6,  
                                random_state=42) ----- 사이킷런의 model_selection 패키지에서  
----- 제공하는 train_test_split 메서드를 활용하여  
----- 훈련셋(train set)과 검증셋(test set)으로 분리
```

# 3.1 지도 학습

## ● 서포트 벡터 머신

- 먼저 사이킷런으로 SVM 모델을 생성 및 훈련시킨 후 검증 데이터를 이용한 예측을 수행

### 코드 3-8 SVM 모델에 대한 정확도

```
svm = svm.SVC(kernel='linear', C=1.0, gamma=0.5) ----- ①
svm.fit(X_train, y_train) ----- 훈련 데이터를 사용하여 SVM 분류기를 훈련
predictions = svm.predict(x_test) ----- 훈련된 모델을 사용하여 검증 데이터에서 예측
score = metrics.accuracy_score(y_test, predictions)
print('정확도: {:.2f}'.format(score)) ----- 검증 데이터 (예측) 정확도 측정
```

- 다음은 SVM 모델에 대한 정확도 출력 결과

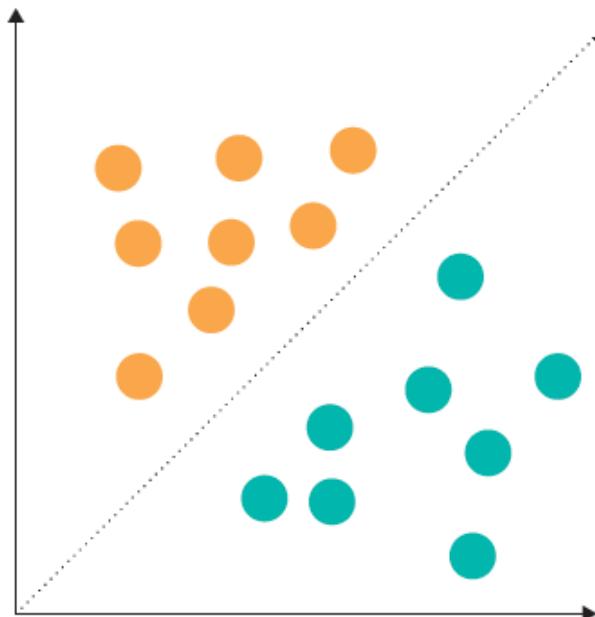
정확도: 0.988889

## 3.1 지도 학습

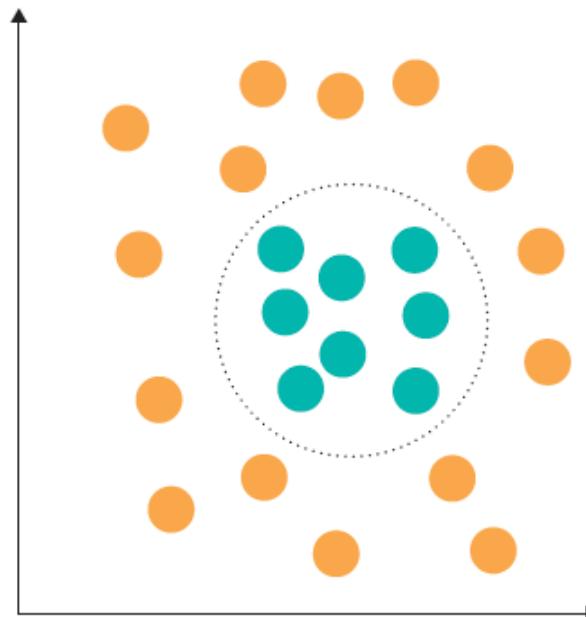
### ● 서포트 벡터 머신

- ① SVM은 선형 분류와 비선형 분류를 지원
- 비선형에 대한 커널은 선형으로 분류될 수 없는 데이터들 때문에 발생

▼ 그림 3-10 선형과 비선형 분류



선형



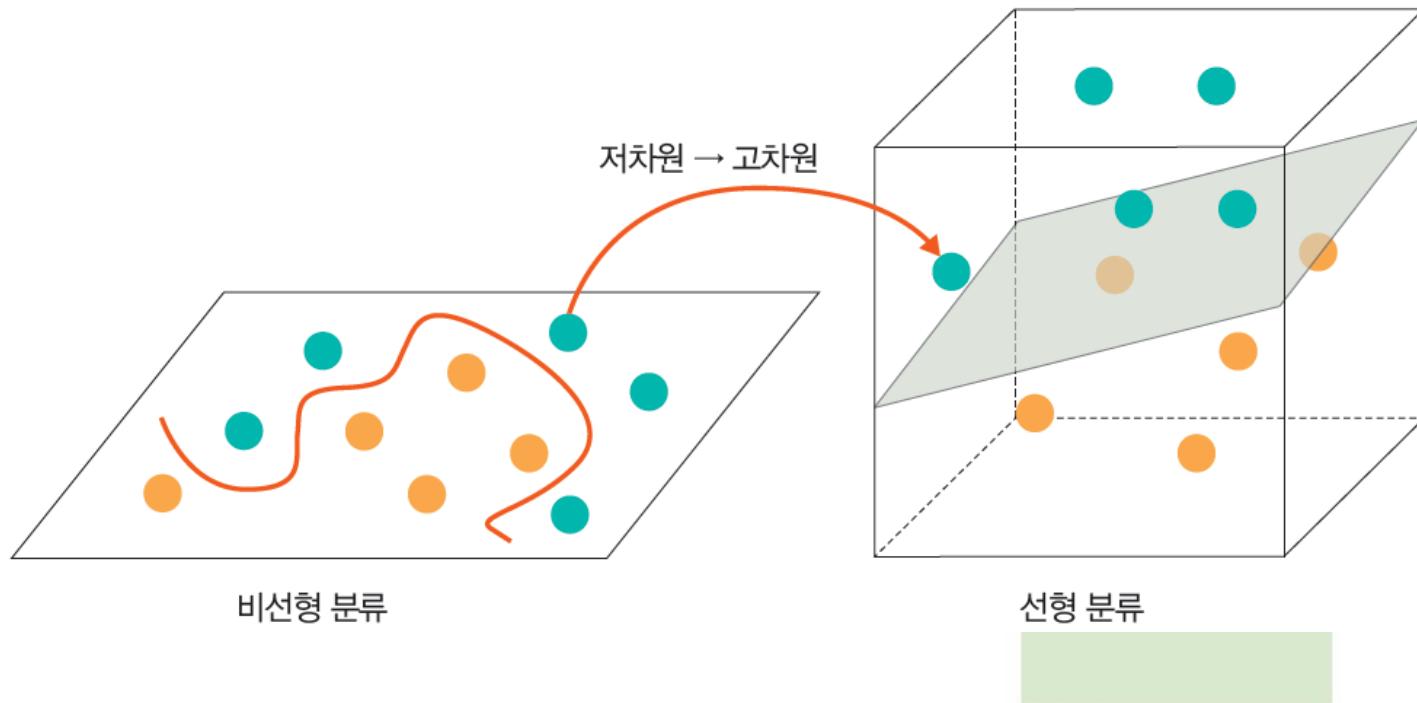
비선형

## 3.1 지도 학습

### ● 서포트 벡터 머신

- 비선형 문제를 해결하는 가장 기본적인 방법은 저차원 데이터를 고차원으로 보내는 것인데, 이것은 많은 수학적 계산이 필요하기 때문에 성능에 문제를 줄 수 있음

▼ 그림 3-11 비선형과 선형 분류



# 3.1 지도 학습

## ● 서포트 벡터 머신

- 이러한 문제를 해결하고자 도입한 것이 바로 '커널 트릭(kernel trick)'
- 선형 모델을 위한 커널(kernel)에는 선형(linear) 커널이 있고, 비선형을 위한 커널에는 가우시안 RBF 커널과 다항식 커널(poly)이 있음
- 가우시안 RBF 커널과 다항식 커널은 수학적 기교를 이용하는 것으로, 벡터 내적을 계산한 후 고차원으로 보내는 방법으로 연산량을 줄였음(벡터 내적은 별도의 인공지능 수학 관련 도서를 참고하기 바람)

- **선형 커널**(linear kernel): 선형으로 분류 가능한 데이터에 적용하며, 다음 수식을 사용

$$K(a, b) = a^T \cdot b$$

( $a, b$ : 입력 벡터)

또한, 선형 커널은 기본 커널 트릭으로 커널 트릭을 사용하지 않겠다는 의미와 일맥상통함

## 3.1 지도 학습

### ● 서포트 벡터 머신

- **다항식 커널**(polynomial kernel): 실제로는 특성을 추가하지 않지만, 다항식 특성을 많이 추가한 것과 같은 결과를 얻을 수 있는 방법  
즉, 실제로는 특성을 추가하지 않지만, 엄청난 수의 특성 조합이 생기는 것과 같은 효과를 얻기 때문에 고차원으로 데이터 매핑이 가능하게 함

$$K(a, b) = (\gamma a^T \cdot b)^d$$

$a, b$ : 입력 벡터  
 $\gamma$ : 감마(gamma)  
 $d$ : 차원, 이때  $\gamma, d$ 는 하이퍼파라미터

## 3.1 지도 학습

### ● 서포트 벡터 머신

- **가우시안 RBF 커널**(Gaussian RBF kernel): 다항식 커널의 확장이라고 생각해도 좋음  
입력 벡터를 차원이 무한한 고차원으로 매핑하는 것으로, 모든 차수의 모든 다항식을 고려함  
즉, 다항식 커널은 차수에 한계가 있는데, 가우시안 RBF는 차수에 제한 없이 무한한 확장이 가능

$$K(a, b) = \exp(-\gamma \|a - b\|^2)$$

(이때  $\gamma$ 는 하이퍼파라미터)

## 3.1 지도 학습

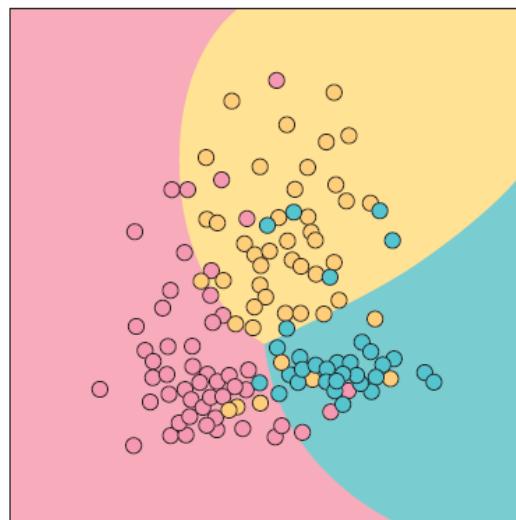
### ● 서포트 벡터 머신

- 코드 3-8과 같이 세 가지 커널에서 사용되는 수치 값 중 C 값은 오류를 어느 정도 허용할지 지정하는 파라미터이며, C 값이 클수록 하드 마진이고 작을수록 소프트 마진
- 감마(gamma)는 결정 경계를 얼마나 유연하게 가져갈지 지정함
- 즉, 훈련 데이터에 얼마나 민감하게 반응할지 지정하기 때문에 C와 개념이 비슷함
- 감마 값이 높으면 훈련 데이터에 많이 의존하기 때문에 결정 경계가 곡선 형태를 띠며 과적합을 초래할 수 있으니 주의해야 함

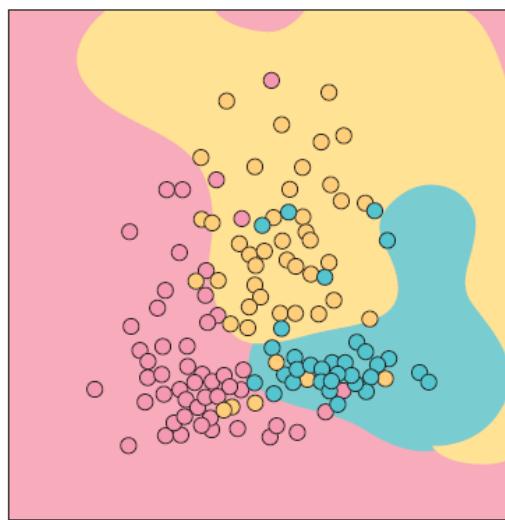
## 3.1 지도 학습

▼ 그림 3-12 서포트 벡터 머신 커널의 감마 값에 따른 변화

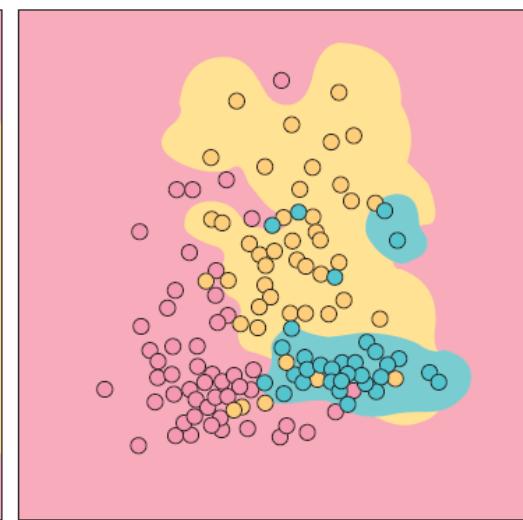
RBF 커널, 감마 = 0.1



RBF 커널, 감마 = 1



RBF 커널, 감마 = 10



## 3.1 지도 학습

### ● 결정 트리

#### ▼ 표 3-4 결정 트리를 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터에 대한 분류
언제 사용하면 좋을까?	결정 트리는 이상치가 많은 값으로 구성된 데이터셋을 다룰 때 사용하면 좋습니다. 또한, 결정 과정이 시각적으로 표현되기 때문에 머신 러닝이 어떤 방식으로 의사 결정을 하는지 알고 싶을 때 유용합니다.

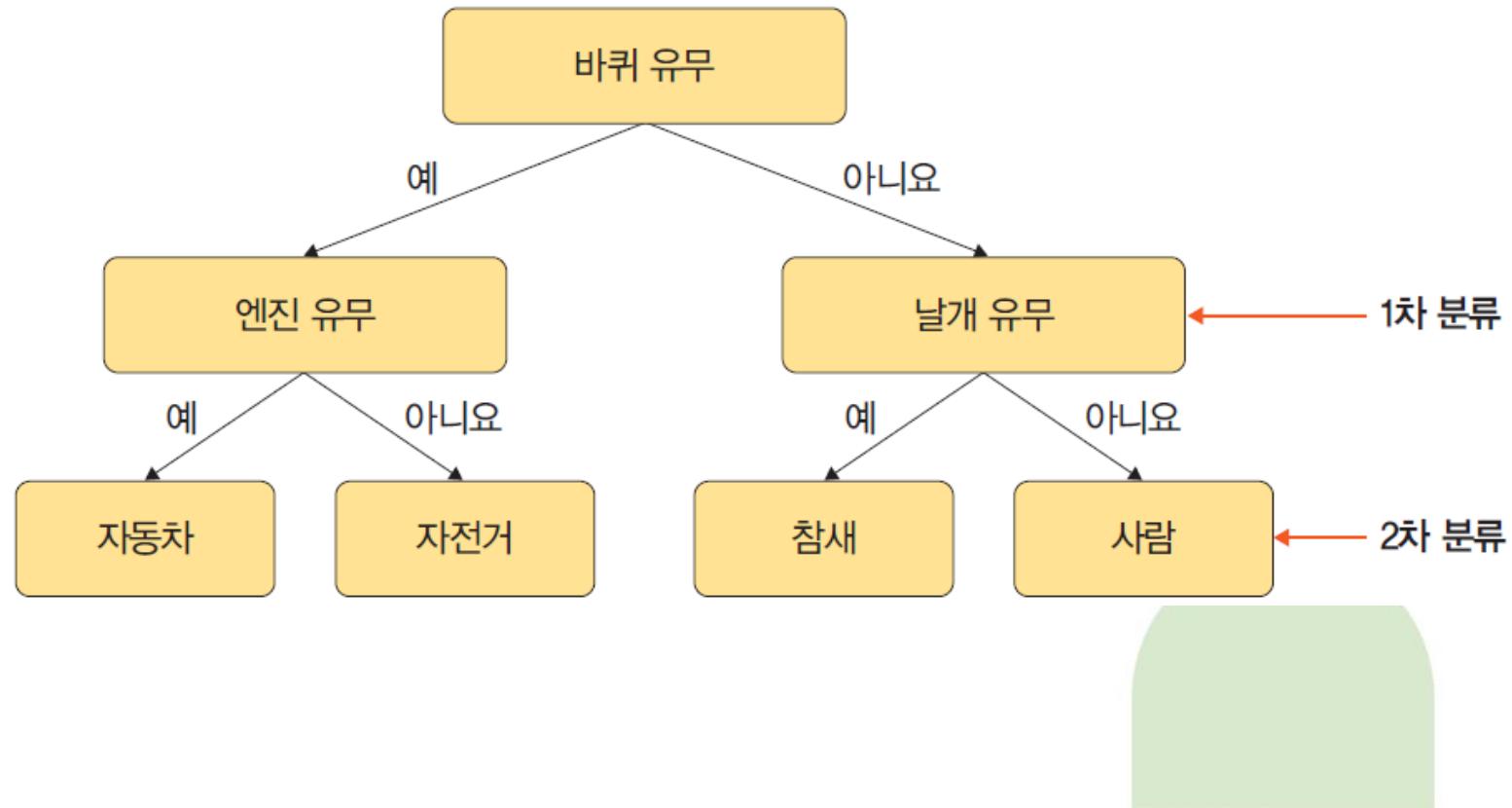
- 결정 트리(decision tree)는 데이터를 분류하거나 결괏값을 예측하는 분석 방법
- 결과 모델이 트리 구조이기 때문에 결정 트리라고 함

## 3.1 지도 학습

### ● 결정 트리

- 다음 그림은 결정 과정을 보여 줌

▼ 그림 3-13 결정 트리 사례



## 3.1 지도 학습

### ● 결정 트리

- 결정 트리는 데이터를 1차로 분류한 후 각 영역의 순도(homogeneity)는 증가하고, 불순도(impurity)와 불확실성(uncertainty)은 감소하는 방향으로 학습을 진행
- 순도가 증가하고 불확실성이 감소하는 것을 정보 이론에서는 정보 획득(information gain)이라고 하며, 순도를 계산하는 방법에는 다음 두 가지를 많이 사용

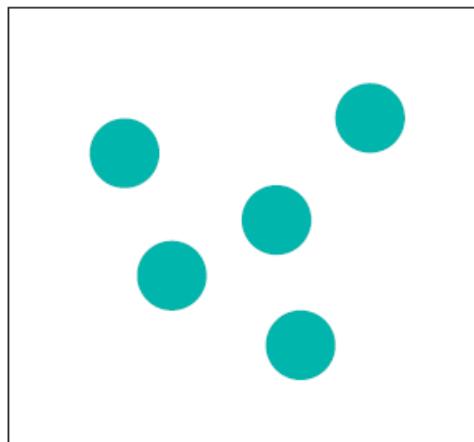
## 3.1 지도 학습

### ● 결정 트리

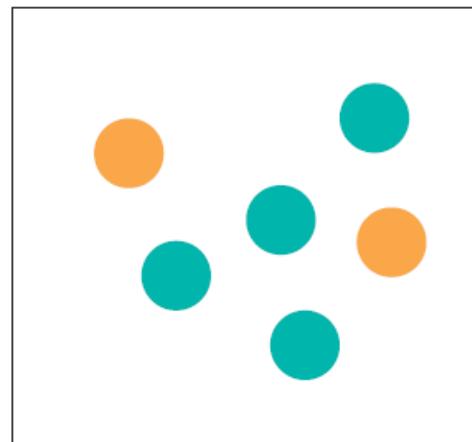
#### 순도와 불순도

- 순도는 범주 안에서 같은 종류의 데이터만 모여 있는 상태이며, 불순도는 서로 다른 데이터가 섞여 있는 상태

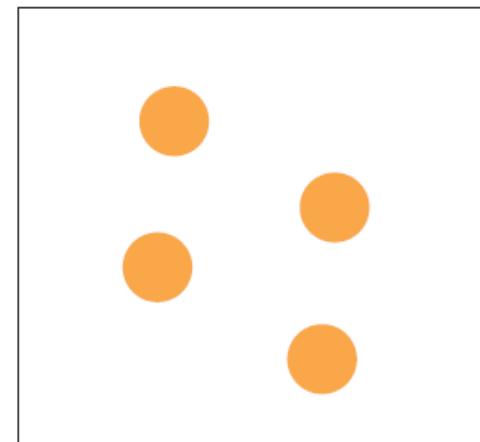
#### ▼ 그림 3-14 순도와 불순도



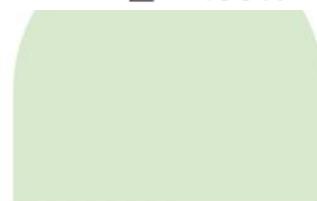
순도 100%



불순도가 높은 상태



순도 100%



## 3.1 지도 학습

### ● 결정 트리

- 결정 트리에서 불확실성을 계산하는 방법은 두 가지

#### 엔트로피(entropy)

- 확률 변수의 불확실성을 수치로 나타낸 것으로, 엔트로피가 높을수록 불확실성이 높다는 의미
- 즉, 엔트로피 값이 0과 0.5라고 가정할 때 다음 도출이 가능

엔트로피 = 0 = 불확실성 최소 = 순도 최대

엔트로피 = 0.5 = 불확실성 최대 = 순도 최소

- 레코드 m개가 A 영역에 포함되어 있다면 엔트로피는 다음 식으로 정의

$$Entropy(A) = - \sum_{k=1}^m p_k \log_2(p_k)$$

( $P_k = A$  영역에 속하는 데이터 가운데 k 범주에 속하는 데이터 비율)

## 3.1 지도 학습

### ● 결정 트리

- 예를 들어 동전을 두 번 던져 앞면이 나올 확률이 1/4이고 뒷면이 나올 확률이 3/4일 때, 엔트로피는 다음과 같음

$$\begin{aligned}Entropy(A) &= -\left(\frac{1}{4}\right)\log\left(\frac{1}{4}\right)-\left(\frac{3}{4}\right)\log\left(\frac{3}{4}\right) \\&= \frac{2}{4} + \frac{3}{4} \times 0.42 \\&= 0.31\end{aligned}$$

# 3.1 지도 학습

## ● 결정 트리

### 지니 계수(Gini index)

- 불순도를 측정하는 지표로, 데이터의 통계적 분산 정도를 정량화해서 표현한 값
- 즉, 지니 계수는 원소  $n$ 개 중에서 임의로 두 개를 추출했을 때, 추출된 두 개가 서로 다른 그룹에 속해 있을 확률을 의미
- 지니 계수는 다음 공식으로 구할 수 있으며, 지니 계수가 높을수록 데이터가 분산되어 있음을 의미

$$G(S) = 1 - \sum_{i=1}^c p_i^2$$

( $S$ : 이미 발생한 사건의 모음,  $c$ : 사건 개수)

- 지니 계수는 로그를 계산할 필요가 없어 엔트로피보다 계산이 빠르기 때문에 결정 트리에서 많이 사용

## 3.1 지도 학습

### ● 결정 트리

- 그림 코드로 자세히 살펴보자
- 이 예제의 목표는 타이타닉 승객의 생존 여부를 예측하는 것

▼ 그림 3-15 결정 트리 예제

목표

타이타닉 승객 생존 여부 예측

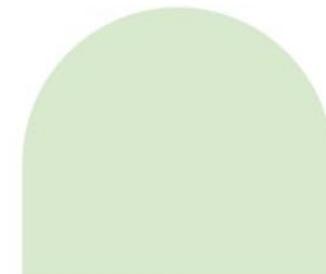
분석  
절차

데이터셋 로딩

훈련과 검증 데이터셋 분리

모델 생성

검증 데이터 예측



# 3.1 지도 학습

## ● 결정 트리

- 먼저 필요한 데이터를 불러오자
- 데이터는 내려받은 예제 파일의 data 폴더에 있는 train.csv 파일을 사용

코드 3-9 라이브러리 호출 및 데이터 준비

```
import pandas as pd  
df = pd.read_csv('../chap3/data/titanic/train.csv', index_col='PassengerId')  
print(df.head())
```

판다스를 이용하여 train.csv 파일을 로드해서 df에 저장  
-----  
train.csv 데이터의 상위 행 다섯 개를 출력  
-----

# 3.1 지도 학습

## ● 결정 트리

- 라이브러리 호출 및 데이터 준비 코드를 실행하면 다음과 같이 출력

```
Survived Pclass \
PassengerId
1          0      3
2          1      1
3          1      3
4          1      1
5          0      3
```

```
Name      Sex   Age \
PassengerId
1           Braund, Mr. Owen Harris    male  22.0
2           Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0
3           Heikkinen, Miss. Laina    female 26.0
4           Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0
5           Allen, Mr. William Henry   male  35.0
```

## 3.1 지도 학습

### ● 결정 트리

	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId						
1	1	0	A/5 21171	7.2500	NaN	S
2	1	0	PC 17599	71.2833	C85	C
3	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	0	113803	53.1000	C123	S
5	0	0	373450	8.0500	NaN	S

# 3.1 지도 학습

## ● 결정 트리

- 타이타닉 전체 데이터 중 분석에 필요한 데이터(칼럼)만 추출하여 전처리함

코드 3-10 데이터 전처리

승객의 생존 여부를 예측하려고 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare' 사용

```
df = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Survived']] -----  
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1}) ----- 성별을 나타내는 'sex'를 0 또는 1의  
df = df.dropna() ----- 값이 없는 데이터 삭제  
X = df.drop('Survived', axis=1)  
y = df['Survived'] ----- 'Survived' 값을 예측 레이블로 사용
```



# 3.1 지도 학습

## ● 결정 트리

- 훈련과 검증 데이터셋으로 분리

코드 3-11 훈련과 검증 데이터셋으로 분리

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

- 사이킷런에서 제공하는 결정 트리 라이브러리를 이용하여 모델을 생성

코드 3-12 결정 트리 모델 생성

```
from sklearn import tree  
model = tree.DecisionTreeClassifier()
```

## 3.1 지도 학습

### ● 결정 트리

- 준비된 훈련 데이터셋을 이용하여 모델을 훈련시킴

코드 3-13 모델 훈련

```
model.fit(X_train, y_train) ----- 모델을 훈련시킵니다.
```

- 다음은 모델 훈련에 대한 실행 결과

`DecisionTreeClassifier()`

# 3.1 지도 학습

## ● 결정 트리

- 모델을 이용하여 검증 데이터에 대한 예측을 진행

코드 3-14 모델 예측

```
y_predict = model.predict(X_test)  
from sklearn.metrics import accuracy_score  
accuracy_score(y_test, y_predict) ----- 검증 데이터에 대한 예측 결과를 보여 줍니다.
```

- 다음은 모델 예측에 대한 출력 결과

0.8379888268156425

- 결과가 83%로 높은 수치를 보이고 있음
- 즉, 학습이 잘되었음

# 3.1 지도 학습

## ● 결정 트리

- 이번에는 혼동 행렬을 이용한 결과를 살펴보겠음

코드 3-15 혼동 행렬을 이용한 성능 측정

```
from sklearn.metrics import confusion_matrix
pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted Not Survival', 'Predicted Survival'],
    index=[ 'True Not Survival', 'True Survival']
)
```

## 3.1 지도 학습

### ● 결정 트리

- 다음 그림은 혼동 행렬에 대한 출력 결과
- ▼ 그림 3-16 결정 트리 코드 실행 결과

	Predicted Not Survival	Predicted Survival
True Not Survival	99	13
True Survival	16	51

- 결과가 제대로 나왔는지 확인하려면 혼동 행렬의 개념을 이해해야 함

## 3.1 지도 학습

### ● 결정 트리

- 혼동 행렬은 알고리즘 성능 평가에 사용
- 혼동 행렬에서 사용되는 다음 표를 먼저 살펴보자

▼ 표 3-5 혼동 행렬

		예측 값	
		Positive	Negative
실제 값	Positive	TP	FN
	Negative	FP	TN

## 3.1 지도 학습

### ● 결정 트리

● 혼동 행렬에서 사용하는 용어는 2장에서 다루었지만 리마인드 차원에서 다시 정리하면 다음과 같음

- **True Positive:** 모델(분류기)이 '1'이라고 예측했는데 실제 값도 '1'인 경우
- **True Negative:** 모델(분류기)이 '0'이라고 예측했는데 실제 값도 '0'인 경우
- **False Positive:** 모델(분류기)이 '1'이라고 예측했는데 실제 값은 '0'인 경우
- **False Negative:** 모델(분류기)이 '0'이라고 예측했는데 실제 값은 '1'인 경우

## 3.1 지도 학습

### ● 결정 트리

- 혼동 행렬을 이용하면 2장에서 배운 정밀도, 재현율, 정확도 같은 지표를 얻을 수 있음
- 혼동 행렬을 바탕으로 모델의 훈련 결과를 확인해 보자
- 잘못된 예측(다음 그림의 파란색)보다는 정확한 예측(다음 그림의 빨간색)의 수치가 더 높으므로 잘 훈련되었다고 할 수 있음

#### ▼ 그림 3-17 혼동 행렬 훈련 결과

		Predicted Not Survival	Predicted Survival
		True Not Survival	True Survival
True Not Survival	Predicted Not Survival	99	13
	Predicted Survival	16	51

## 3.1 지도 학습

### ● 결정 트리

- 이와 같이 주어진 데이터를 사용하여 트리 형식으로 데이터를 이진 분류(0 혹은 1)해 나가는 방법이 결정 트리
- 결정 트리를 좀 더 확대한 것(결정 트리를 여러 개 묶어 놓은 것)이 랜덤 포레스트(random forest)

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 회귀란 변수가 두 개 주어졌을 때 한 변수에서 다른 변수를 예측하거나 두 변수의 관계를 규명하는 데 사용하는 방법
- 이때 사용되는 변수 유형은 다음과 같음
  - 독립 변수(예측 변수): 영향을 미칠 것으로 예상되는 변수
  - 종속 변수(기준 변수): 영향을 받을 것으로 예상되는 변수
- 이때 두 변수 간 관계에서 독립 변수와 종속 변수의 설정은 논리적인 타당성이 있어야 함
- 예를 들어 몸무게(종속 변수)와 키(독립 변수)는 둘 간의 관계를 규명하는 용도로 사용

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

#### 로지스틱 회귀

- 먼저 로지스틱 회귀 분석에 대해 살펴보자

#### ▼ 표 3-6 로지스틱 회귀를 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터에 대한 분류
언제 사용하면 좋을까?	로지스틱 회귀 분석은 주어진 데이터에 대한 확신이 없거나(예를 들어 분류 결과에 대해 확신이 없을 때) 향후 추가적으로 훈련 데이터셋을 수집하여 모델을 훈련시킬 수 있는 환경에서 사용하면 유용합니다.

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 로지스틱 회귀(logistic regression)는 분석하고자 하는 대상들이 두 집단 혹은 그 이상의 집단으로 나누어진 경우, 개별 관측치들이 어느 집단에 분류될 수 있는지 분석하고 이를 예측하는 모형을 개발하는 데 사용되는 통계 기법
- 일반적인 회귀 분석과는 차이가 있음

▼ 표 3-7 일반 회귀 분석과 로지스틱 회귀 분석 차이

구분	일반적인 회귀 분석	로지스틱 회귀 분석
종속 변수	연속형 변수	이산형 변수
모형 탐색 방법	최소제곱법	최대우도법
모형 검정	F-테스트, t-테스트	$\chi^2$ 테스트

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

#### 최소제곱법과 최대우도법

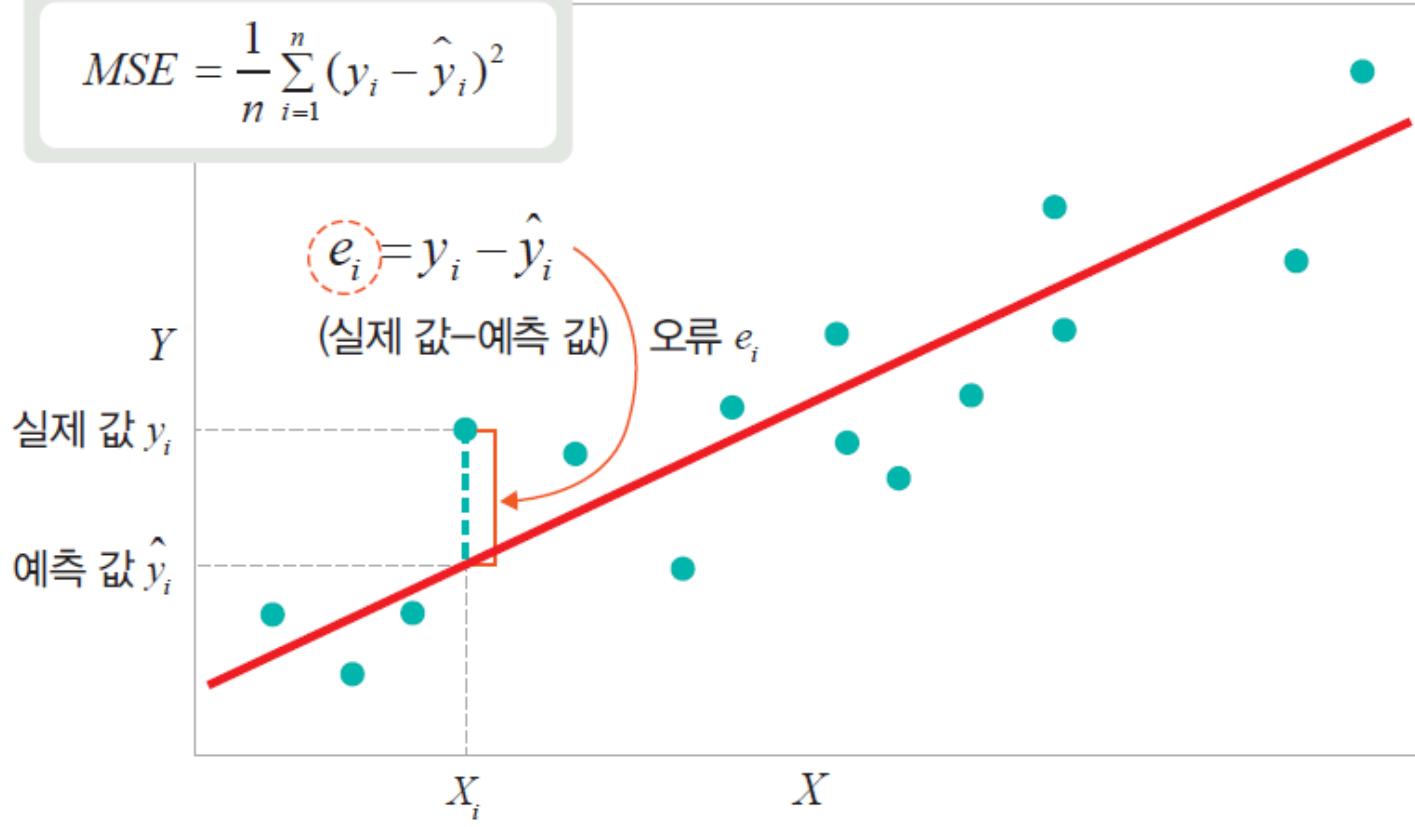
- 최소제곱법(mean squared)과 최대우도법(maximum likelihood)은 랜덤 표본에서 모집단 모수를 추정하는 데 사용
- 최소제곱법은 일반적인 회귀 분석에서 사용하지만, 최대우도법은 로지스틱 회귀 분석에서 사용
- 이 둘 간에 어떤 차이가 있는지 알아보자
- 최소제곱법은 실제 값에서 예측 값을 뺀 후 제곱해서 구할 수 있음(최소제곱법은 1장에서 언급한 평균 제곱 오차와 동일함)

## 3.1 지도 학습

▼ 그림 3-18 최소제곱법

### 최소제곱법

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$



## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 최대우도법을 이해하려면 먼저 우도 개념을 알아야 함
- 우도(likelihood, 가능성)는 나타난 결과에 따라 여러 가능한 가설을 평가할 수 있는 척도(measure)를 의미
- 최대우도는 나타난 결과에 해당하는 가설마다 계산된 우도 값 중 가장 큰 값
- 즉, 일어날 가능성(우도)이 가장 큰 것을 의미
- 이 모든 것을 종합하여 최대우도법을 정의하면 최대우도 추정치 또는 최대 가능성 추정량이라고 할 수 있음

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 최대우도법은 다음 수식으로 구할 수 있음

$$\theta_{ml} = \arg \max_{\theta} P_{\text{model}}(Y | X; \theta) \quad \dots \dots \dots \quad ①$$

$$\theta_{ml} = \arg \max_{\theta} \sum_{i=1}^m \log P_{\text{model}}(y_i | x_i; \theta) \quad \dots \dots \dots \quad ②$$

- 수식 ①과 같이 입력 값  $X$ 와 모델의 파라미터  $\theta$  가 주어졌을 때,  $Y$ 가 나타날 확률을 최대화하는  $\theta$  를 찾는 것이 최대 우도법
- $X$ 와  $Y$ 가 고정된 상태에서 모델에  $X$ 를 넣었을 때 실제 값  $Y$ 에 가장 가까운  $\theta$  를 찾는 것이 수식
- 이때 관측치  $m$ 개가 모두 서로 독립이라고 가정할 때, 언더플로를 방지하고자 우도에 로그를 취한다면 최대우도 추정치는 수식 ②와 같음

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 로지스틱 회귀 분석은 다음 절차에 따라 분석을 진행

- 1단계: 각 집단에 속하는 확률의 추정치를 예측  
이때 추정치는 이진 분류의 경우 집단 1에 속하는 확률  $P(Y=1)$ 로 구함
- 2단계: 분류 기준 값(cut-off)을 설정한 후 특정 범주로 분류

예  $P(Y=1) \geq 0.5 \rightarrow$  집단 1로 분류

$P(Y=1) < 0.5 \rightarrow$  집단 0으로 분류

- 로지스틱 회귀 분석이 어렵게 느껴진다면 확률과 통계에 익숙하지 않기 때문임
- 이쯤에서 인공지능 관련 수학을 다시 찾아보길 권함

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 로지스틱 회귀 분석 코드를 살펴보자
- 목표는 신규 데이터(숫자(digit))에 대한 정확한 예측

#### ▼ 그림 3-19 로지스틱 회귀 분석 예제

목표

신규 입력(숫자, digit) 데이터의 정확한 예측

분석  
절차

라이브러리 호출 → 데이터셋 로딩 → 훈련과 검증 데이터셋 분리

모델 생성 → 검증 데이터 예측 → 모델 성능 검증

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

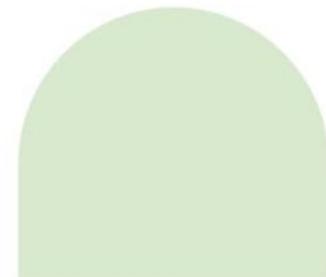
- 라이브러리를 호출하고 데이터를 준비
- 사용할 digits 숫자 데이터셋은 사이킷런에서 제공하는 데이터셋

코드 3-16 라이브러리 호출 및 데이터 준비

```
%matplotlib inline  
from sklearn.datasets import load_digits  
digits = load_digits() ----- 숫자 데이터셋(digits)은 사이킷런에서 제공  
  
print("Image Data Shape", digits.data.shape) ----- digits 데이터셋의 형태(이미지가 1797개 있으며,  
8×8 이미지의 64차원을 가짐)  
print("Label Data Shape", digits.target.shape) ----- 레이블 이미지 1797개가 있음
```

- 코드를 실행하면 다음과 같이 digits 데이터셋 형태를 출력해서 보여 줌

Image Data Shape (1797, 64)  
Label Data Shape (1797, )



# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- digits 데이터셋의 이미지와 레이블이 어떻게 생겼는지 시각화해서 확인해 보자

코드 3-17 digits 데이터셋의 시각화

```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):
    plt.subplot(1, 5, index+1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize=20)
```

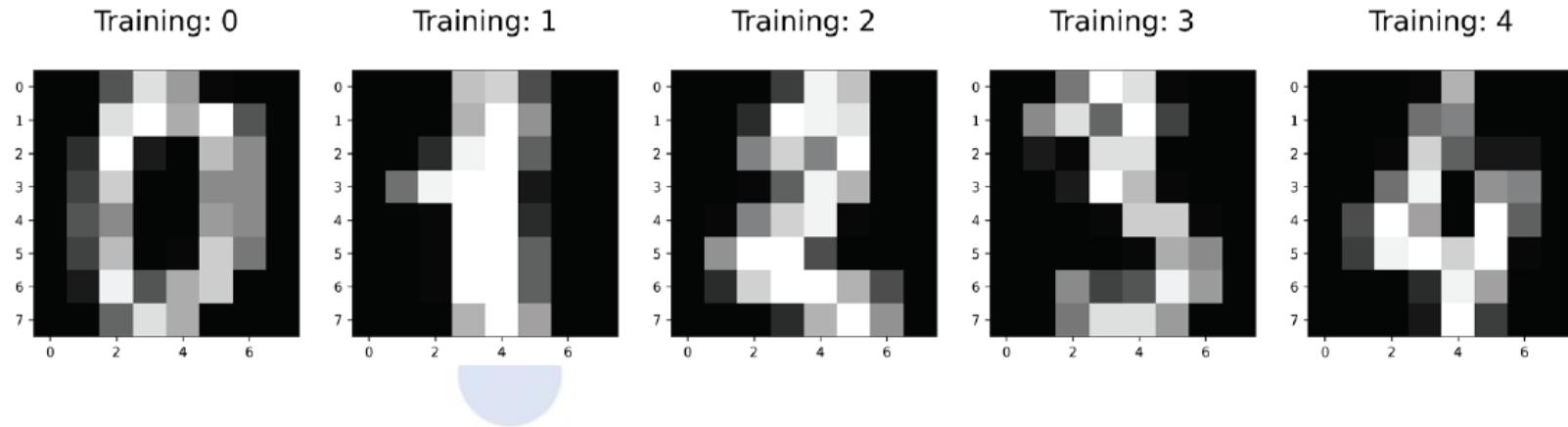
예시로 이미지 다섯 개만 확인

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 다음 그림은 digits 데이터셋을 시각화한 출력 결과

▼ 그림 3-20 로지스틱 회귀 예제 데이터



# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 훈련과 검증 데이터셋으로 분리한 후 분리된 데이터를 사용하여 모델을 훈련시킴

코드 3-18 훈련과 검증 데이터셋 분리 및 로지스틱 회귀 모델 생성

```
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target,  
                                                    test_size=0.25, random_state=0)  
  
from sklearn.linear_model import LogisticRegression  
logisticRegr = LogisticRegression() ----- 로지스틱 회귀 모델의 인스턴스 생성  
logisticRegr.fit(x_train, y_train) ----- 모델 훈련
```

- 코드를 실행하면 다음과 같이 출력

LogisticRegression()

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 모델에 대한 예측을 검증 데이터를 사용해서 진행해 보자

코드 3-19 일부 데이터를 사용한 모델 예측

```
logisticRegr.predict(x_test[0].reshape(1,-1)) ----- 새로운 이미지(검증 데이터)에 대한  
logisticRegr.predict(x_test[0:10]) ----- 예측 결과를 넘파이 배열로 출력  
----- 이미지 열 개에 대한 예측을 한 번에 배열로 출력
```

- 열 개의 이미지 데이터를 사용한 로지스틱 회귀 모델에 대한 예측 결과는 다음과 같이 출력

```
array([2, 8, 2, 6, 6, 7, 1, 9, 8, 5])
```

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 모델 성능을 측정하겠음
- 모델 성능을 측정하는 방법으로는 혼동 행렬(정확도, 정밀도, 재현율), F1-스코어, ROC 커브 등이 있음
- 먼저 정확도에 대한 성능을 확인해 보자
- 정확도는 전체 데이터를 이용하여 진행

코드 3-20 전체 데이터를 사용한 모델 예측

```
predictions = logisticRegr.predict(x_test) ----- 전체 데이터셋에 대한 예측  
score = logisticRegr.score(x_test, y_test) ----- 스코어(score) 메서드를 사용한 성능 측정  
print(score)
```

- 코드를 실행하면 전체 데이터를 사용한 로지스틱 회귀 모델에 대한 예측 결과가 **0.9511111111111111**
- 성능 측정 결과는 95%로 나쁘지 않음

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

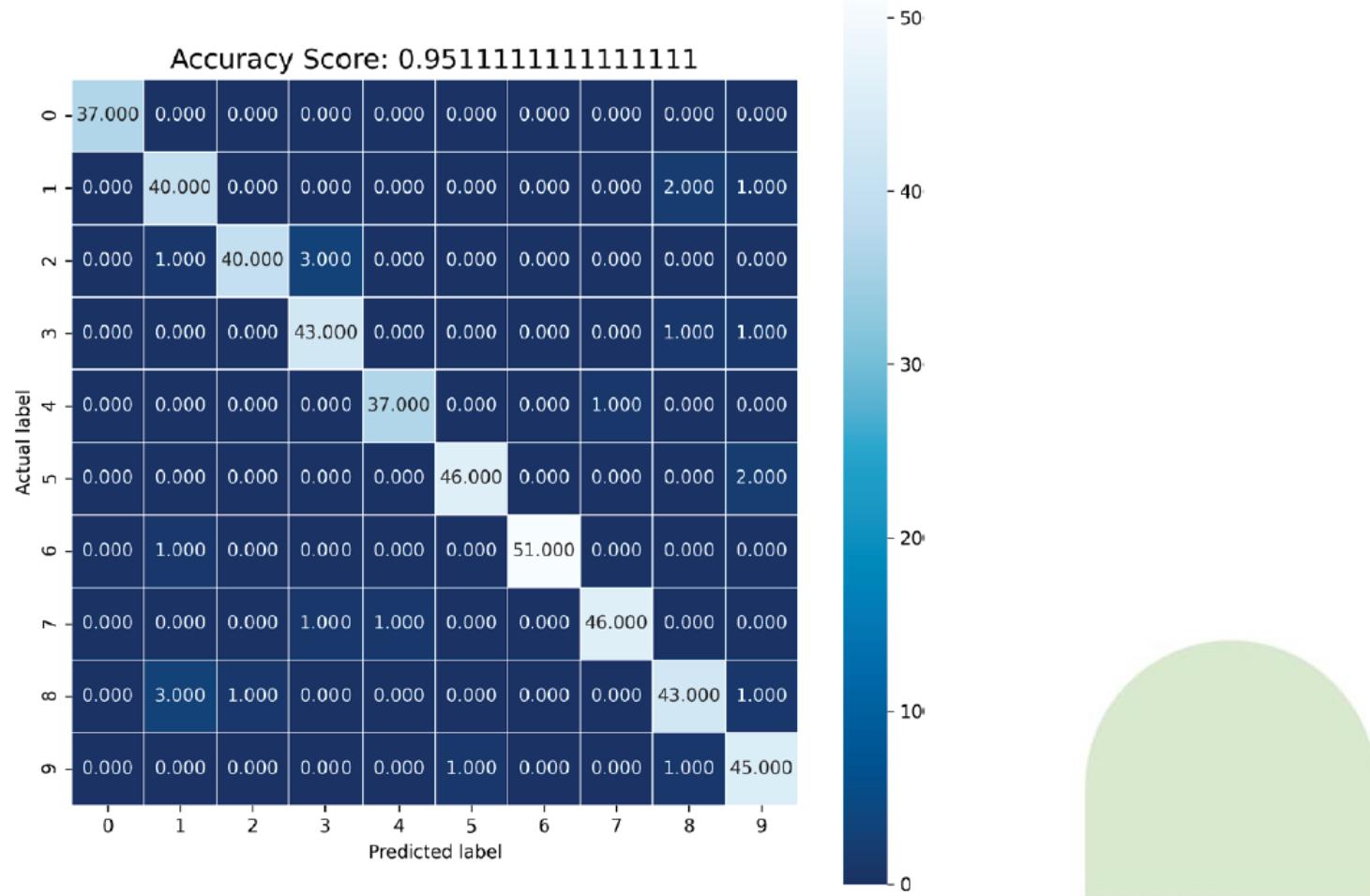
- 결과를 좀 더 명확하게 확인하고자 혼동 행렬로 표현해 보자
- 혼동 행렬은 지도 학습에서 검증 데이터셋에 대한 분류 모델 성능을 설명하는 데 자주 사용

코드 3-21 혼동 행렬 시각화

```
import numpy as np
import seaborn as sns
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predictions) ----- 혼동 행렬(confusion_matrix)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square=True, cmap='Blues_r'); -----
plt.ylabel('Actual label'); ----- y축                                         heatmap으로 표현
plt.xlabel('Predicted label'); ----- x축
all_sample_title = 'Accuracy Score: {}'.format(score)
plt.title(all_sample_title, size=15);
plt.show();
```

## 3.1 지도 학습

▼ 그림 3-21 로지스틱 회귀 예제 실행 결과



## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 결과를 확인하기는 좋지만, 직관적으로 이해하기에는 난해함
- 혼동 행렬은 단지 결과 확인용으로만 사용하길 권장

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

#### 선형 회귀

- 이번에는 선형 회귀 분석을 살펴보자

▼ 표 3-8 선형 회귀를 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터에 대한 분류
언제 사용하면 좋을까?	로지스틱 회귀는 주어진 데이터에서 독립 변수( $x$ )와 종속 변수( $y$ )가 선형 관계를 가질 때 사용하면 유용합니다. 또한, 복잡한 연산 과정이 없기 때문에 컴퓨팅 성능이 낮은 환경(CPU/GPU 혹은 메모리 성능이 좋지 않을 때)에서 사용하면 좋습니다.

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 선형 회귀(linear regression)는 독립 변수  $x$ 를 사용하여 종속 변수  $y$ 의 움직임을 예측하고 설명하는데 사용
- 독립 변수  $x$ 는 하나일 수도 있고,  $x_1, x_2, x_3$ 처럼 여러 개일 수도 있음
- 하나의  $x$  값으로  $y$  값을 설명할 수 있다면 단순 선형 회귀(simple linear regression)라고 하며,  $x$  값이 여러개라면 다중 선형 회귀(multiple linear regression)라고 함

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 선형 회귀는 종속 변수와 독립 변수 사이의 관계를 설정하는 데 사용
- 즉, 독립 변수가 변경되었을 때 종속 변수를 추정하는 데 유용함
- 예를 들어 더운 여름철 아이스크림이 시간당 100개가 팔린다고 할 때  $y=100x$ 라는 함수를 가정할 수 있음(실제로는 더 복잡한 수식이겠지만, 설명을 위해 간단히  $y=100x$ 라고 하겠음)
- 이 함수에 따라 아이스크림 가격이 1000원이라고 한다면 시간당 10만 원의 매출이 될 것
- 이와 같이 단순 회귀를 사용하면 변수 값을 추정할 수 있음

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

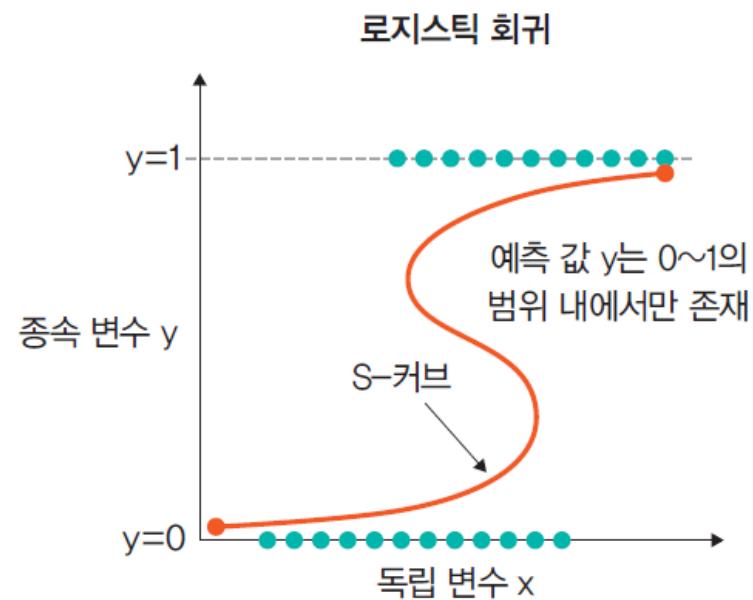
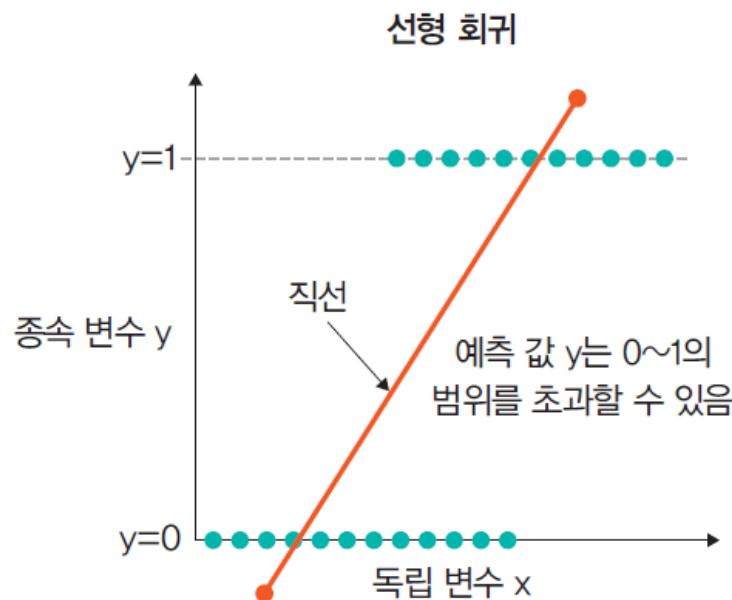
- 반면 로지스틱 회귀는 사건의 확률(0 또는 1)을 확인하는 데 사용
- 예를 들어 고객이 A 제품을 구매할지 여부를 확인하고 싶을 때 로지스틱 회귀 분석을 이용함(종속 변수는 이진 변수(1=예, 0=아니요)로 표현되기 때문)

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 다음 그림과 같이 그래픽으로 살펴보면 선형 회귀는 직선을 출력하고, 로지스틱 회귀는 S-커브를 출력

#### ▼ 그림 3-22 선형 회귀와 로지스틱 회귀



## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 이번에는 선형 회귀에 대해 코드로 확인해 보자
- 캐글에서 제공하는 날씨 데이터셋을 이용할 것

▼ 그림 3-23 선형 회귀 예제

목표

훈련 데이터(특성)에서 최대 기온 예측

분석  
절차

라이브러리 호출

데이터셋 로딩

훈련과 검증 데이터셋 분리

모델 생성

검증 데이터 예측

예측 시각화



## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 날씨 데이터셋에는 전 세계 여러 기상 관측소에서 매일 기록된 기상 조건 정보가 포함되어 있음
- 강수량, 강설량, 기온, 풍속 및 그 날의 놀우 등 정보들이 포함되어 있으나, 예제에서는 최대 온도를 예측하기 때문에 최소/최대 기온(MinTemp, MaxTemp) 정보만 사용

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 필요한 라이브러리를 호출

코드 3-22 라이브러리 호출

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as seabornInstance  
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LinearRegression  
from sklearn import metrics  
%matplotlib inline
```

# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 내려받은 예제 파일의 data 폴더에서 날씨 데이터셋 weather.csv 파일을 불러옴

코드 3-23 weather.csv 파일 불러오기

```
dataset = pd.read_csv('..../chap3/data/weather.csv')
```

- MinTemp와 MaxTemp 데이터 간 분포를 확인하고자 2D 그래프로 시각화함

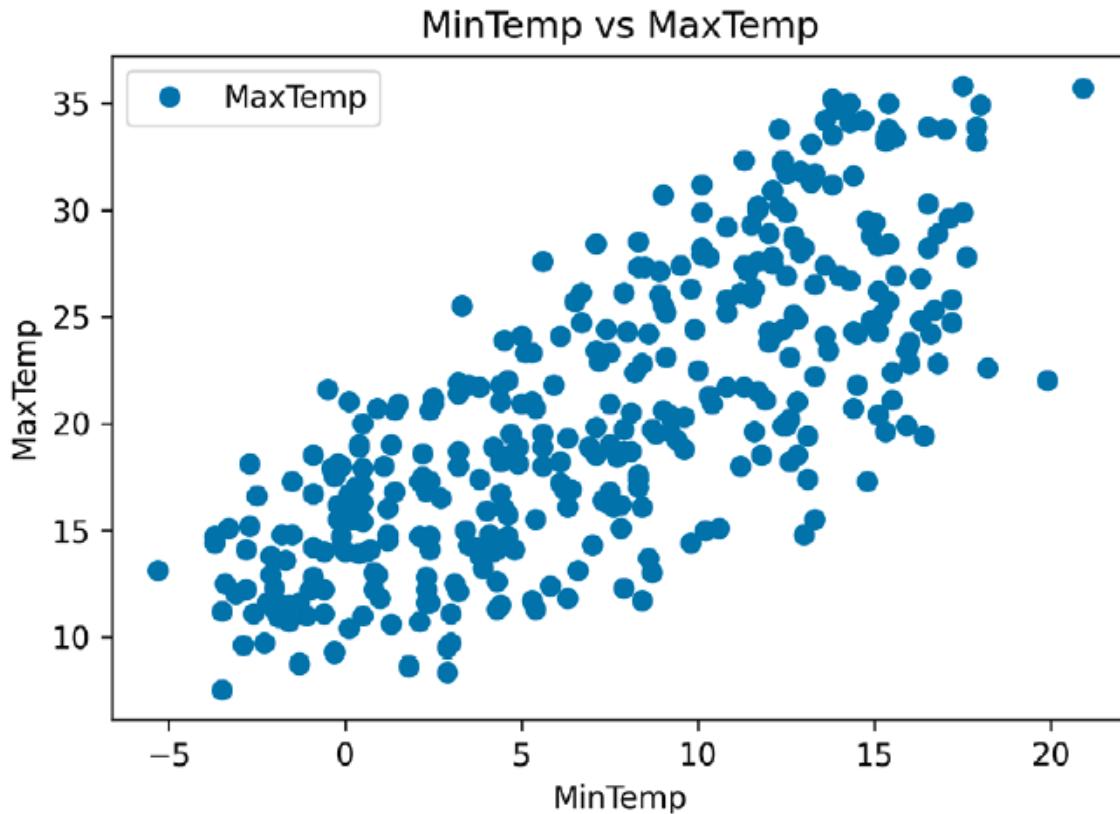
코드 3-24 데이터 간 관계를 시각화로 표현

```
dataset.plot(x='MinTemp', y='MaxTemp', style='o')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()
```

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 다음 그림은 데이터 간 관계를 시각화로 표현한 결과
- ▼ 그림 3-24 선형 회귀 예제 실행 결과



# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 데이터를 '속성(attribute)'과 '레이블(label)'로 나눔
- 속성은 독립 변수이고 레이블은 종속 변수
- MinTemp에 따라 MaxTemp를 예측하기 위해 x 변수는 'MinTemp'로 구성하고, y 변수는 'MaxTemp'로 구성

코드 3-25 데이터를 독립 변수와 종속 변수로 분리하고 선형 회귀 모델 생성

```
X = dataset['MinTemp'].values.reshape(-1,1)          데이터의 80%를 훈련 데이터셋으로 하고  
y = dataset['MaxTemp'].values.reshape(-1,1)          데이터의 20%를 검증 데이터셋으로 분할  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) -----  
  
regressor = LinearRegression() ----- 선형 회귀 클래스를 가져옴  
regressor.fit(X_train, y_train) ----- fit() 메서드를 사용하여 모델 훈련
```

- 다음은 선형 회귀 모델에 대한 실행 결과

`LinearRegression()`



# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 검증 데이터셋을 사용하여 몇 가지 예측을 해 보자
- 먼저 X\_test의 실제 출력 값을 예측 값과 비교해 보자

코드 3-26 회귀 모델에 대한 예측

```
y_pred = regressor.predict(X_test)  
df = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})  
df
```

## 3.1 지도 학습

### ▼ 그림 3-25 선형 회귀 예제 예측 결과

	Actual	Predicted
0	25.2	23.413030
1	11.5	13.086857
2	21.1	27.264856
3	22.2	25.461874
4	20.4	26.937041
...	...	...
69	18.9	20.216833
70	22.8	27.674625
71	16.1	21.446140
72	25.1	24.970151
73	12.2	14.070302

74 rows × 2 columns

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 이번에는 검증 데이터셋을 회귀선(직선)으로 표현해 보자

코드 3-27 검증 데이터셋을 사용한 회귀선 표현

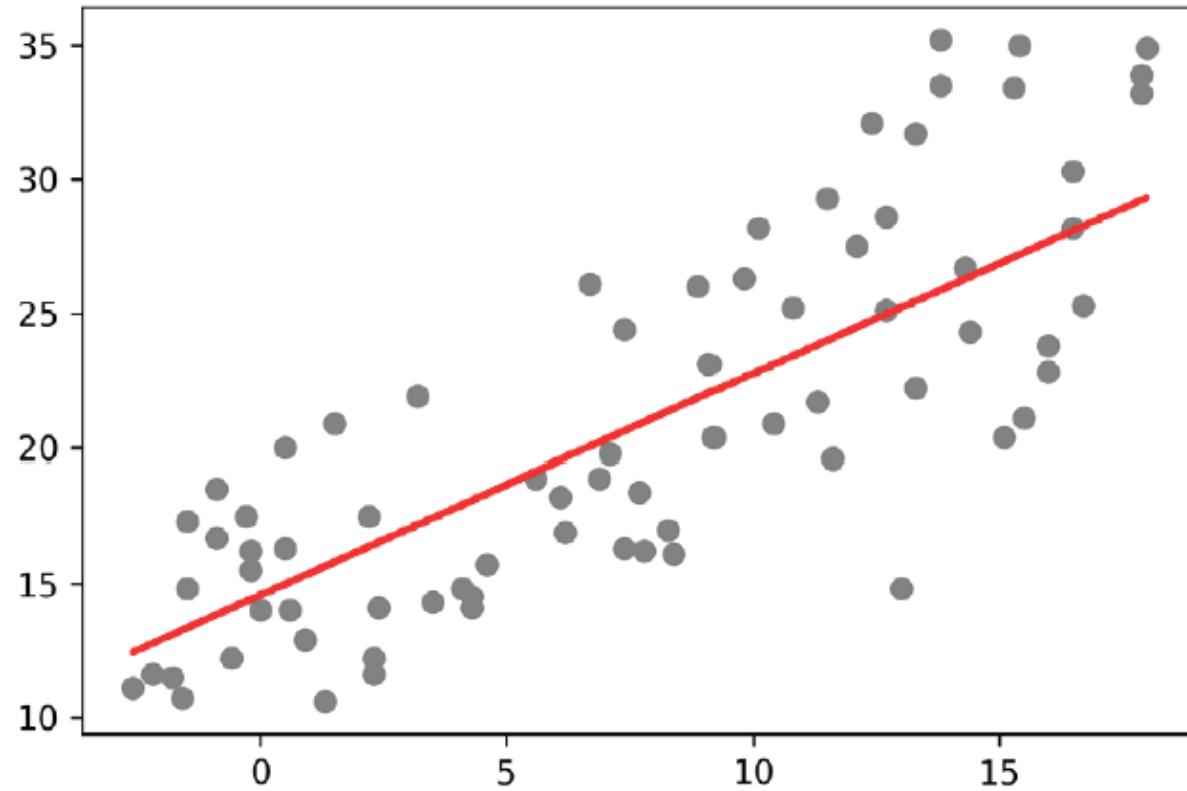
```
plt.scatter(X_test, y_test, color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 다음 그림은 검증 데이터셋을 사용하여 회귀선으로 표현한 출력 결과
- ▼ 그림 3-26 선형 회귀 예측 결과를 회귀선으로 표현



# 3.1 지도 학습

## ● 로지스틱 회귀와 선형 회귀

- 출력 결과 그림을 보면 회귀선이 실제 데이터와 비슷하다는 것을 확인할 수 있음
- 마지막으로 모델을 평가해 보자
- 선형 회귀는 평균 제곱 오차(평균 제곱법)와 루트 평균 제곱 오차(루트 평균제곱법)를 사용하여 모델을 평가

코드 3-28 선형 회귀 모델 평가

```
print('평균제곱법:', metrics.mean_squared_error(y_test, y_pred))
print('루트 평균제곱법:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

- 다음은 선형 회귀 모델 평가에 대한 실행 결과

평균제곱법: 17.011877668640622

루트 평균제곱법: 4.124545753006096

## 3.1 지도 학습

### ● 로지스틱 회귀와 선형 회귀

- 루트 평균제곱법(root mean squared)은 이름에서도 알 수 있듯이 평균제곱법(mean squared)에 루트를 써운 것
- 루트 평균제곱법의 공식은 다음과 같음

평균제곱법이  $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$  이라면,

루트 평균제곱법은 루트만 써운  $RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$  이 됩니다.

- 루트 평균제곱법 값(4.12)은 모든 기온 백분율에 대한 평균값(22.41)과 비교하여 10% 이상임을 알 수 있음
- 모델 정확도는 높지 않지만 여전히 합리적으로 좋은 예측을 할 수 있음을 의미

### 3.2 비지도 학습

---

## 3.2 비지도 학습

### ● 비지도 학습

- 비지도 학습은 지도 학습처럼 레이블이 필요하지 않으며 정답이 없는 상태에서 훈련시키는 방식
- 비지도 학습에는 군집(clustering)과 차원 축소(dimensionality reduction)가 있음
- 군집은 각 데이터의 유사성(거리)을 측정한 후 유사성이 높은(거리가 짧은) 데이터끼리 집단으로 분류하는 것
- 차원 축소는 차원을 나타내는 특성을 줄여서 데이터를 줄이는 방식

## 3.2 비지도 학습

▼ 표 3-9 비지도 학습 군집과 차원 축소 비교

구분	군집	차원 축소
목표	데이터 그룹화	데이터 간소화
주요 알고리즘	K-평균 군집화(K-Means)	주성분 분석(PCA)
예시	사용자의 관심사에 따라 그룹화하여 마케팅에 활용	<ul style="list-style-type: none"><li>데이터 압축</li><li>중요한 속성 도출</li></ul>

## 3.2 비지도 학습

### ● 비지도 학습

#### 군집, 군집화, 클러스터

- 통계학에서는 군집이라고 하며, 머신 러닝에서는 클러스터라고 함
- 또한, 클러스터를 한국어로 바꾸면 군집화가 됨
- 즉, 군집, 군집화, 클러스터는 같은 의미의 다른 표현
- 이 책에서는 군집, 군집화, 클러스터 용어를 혼용하여 사용하지만, 모두 동일한 의미로 이해하면 됨

## 3.2 비지도 학습

### ● 비지도 학습

#### 데이터 간 유사도(거리) 측정 방법

- 데이터 간 유사도(거리)를 측정하는 방법으로 유클리드 거리, 맨해튼 거리, 민코프스키 거리, 코사인 유사도 등이 있음
- 각각에 대한 설명은 인공지능 수학 관련 도서를 참고하기 바람

## 3.2 비지도 학습

### ● K-평균 군집화

▼ 표 3-10 K-평균 군집화를 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터에 대한 군집화
언제 사용하면 좋을까?	주어진 데이터셋을 이용하여 몇 개의 클러스터를 구성할지 사전에 알 수 있을 때 사용하면 유용합니다.

## 3.2 비지도 학습

### ● K-평균 군집화

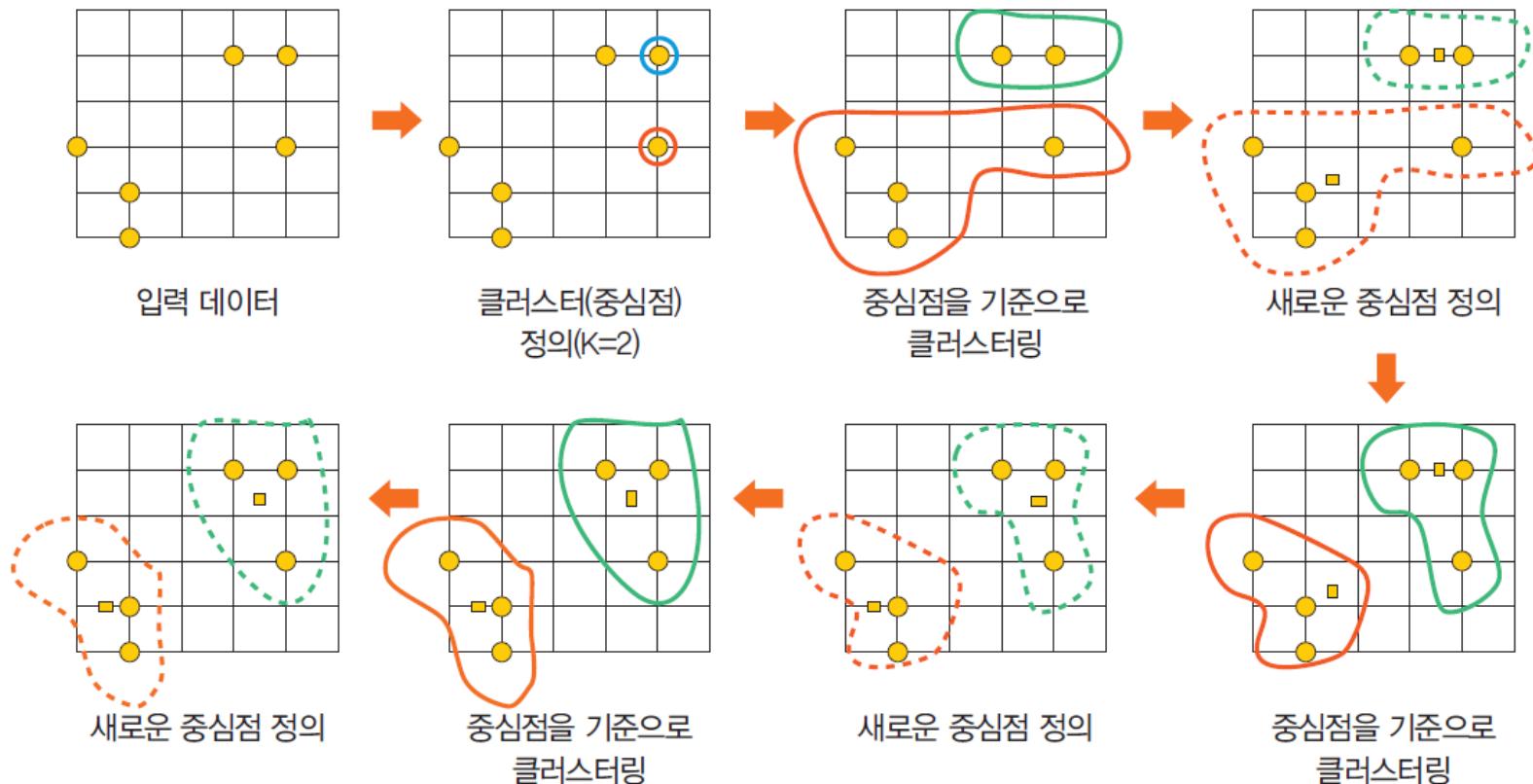
- K-평균 군집화(K-means clustering)는 데이터를 입력받아 소수의 그룹으로 묶는 알고리즘
- 레이블이 없는 데이터를 입력받아 각 데이터에 레이블을 할당해서 군집화를 수행하는데, 학습 과정은 다음과 같음

1. 중심점 선택: 랜덤하게 초기 중심점(centroid)을 선택(그림에서는 K=2로 초기화)
2. 클러스터 할당: K개의 중심점과 각각의 개별 데이터 간의 거리(distance)를 측정한 후, 가장 가까운 중심점을 기준으로 데이터를 할당(assign)  
이 과정을 통해 클러스터가 구성(이때 클러스터링은 데이터를 하나 혹은 둘 이상의 덩어리로 묶는 과정이며, 클러스터는 덩어리 자체를 의미)
3. 새로운 중심점 선택: 클러스터마다 새로운 중심점을 계산
4. 범위 확인(convergence): 선택된 중심점에 더 이상의 변화가 없다면 진행을 멈춤  
만약 계속 변화가 있다면 1~3 과정을 반복

## 3.2 비지도 학습

### ● K-평균 군집화

- 다음 그림은 반복 횟수에 따른 데이터 분류 과정을 보여 줌
- ▼ 그림 3-27 K-평균 군집화



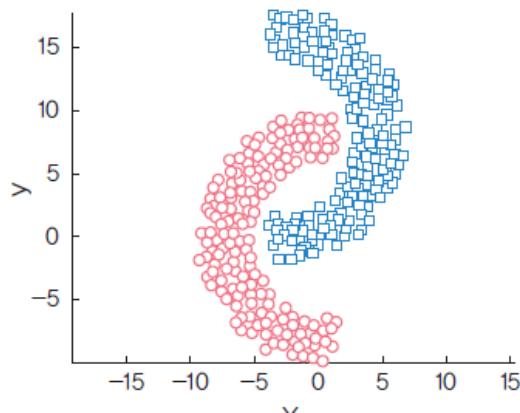
## 3.2 비지도 학습

### ● K-평균 군집화

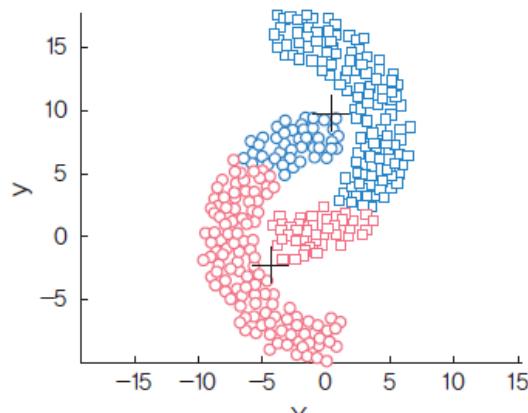
- 참고로 K-평균 군집화 알고리즘은 다음 상황에서는 데이터 분류가 원하는 결과와 다르게 발생할 수 있으므로 사용하지 않는 것이 좋음

데이터가 비선형일 때

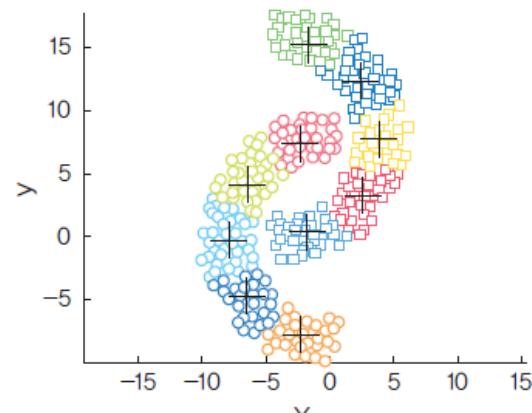
#### ▼ 그림 3-28 비선형 데이터



주어진 데이터



K-means(K=2)



K-means(K=10)

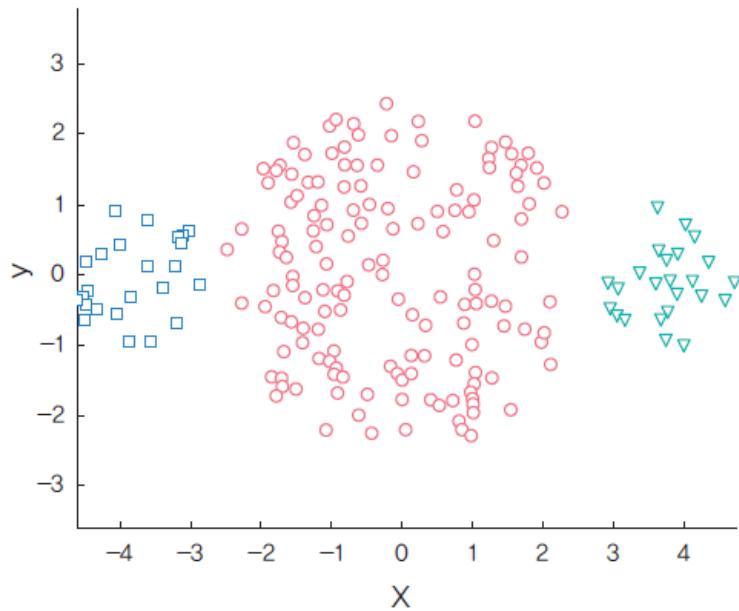


## 3.2 비지도 학습

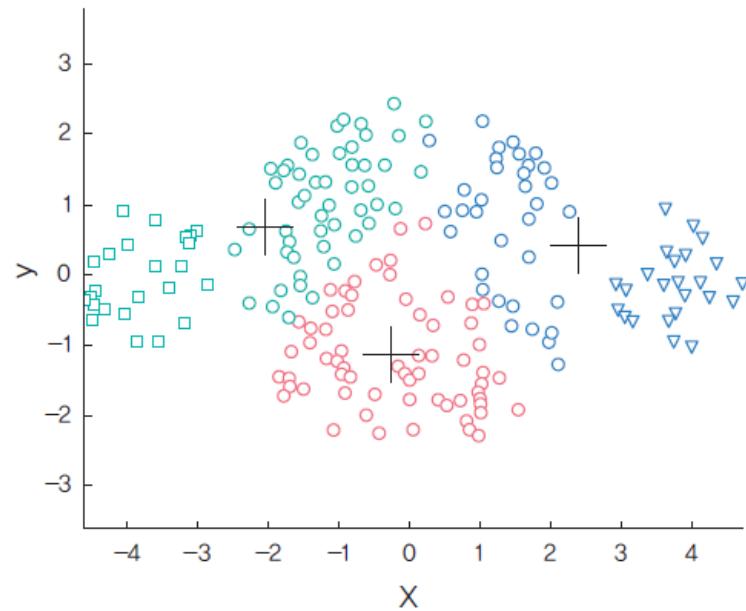
### ● K-평균 군집화

군집 크기가 다를 때

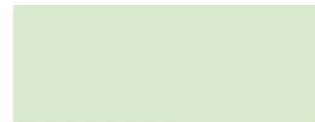
▼ 그림 3-29 서로 다른 군집 크기



주어진 데이터



K-means( $K=3$ )

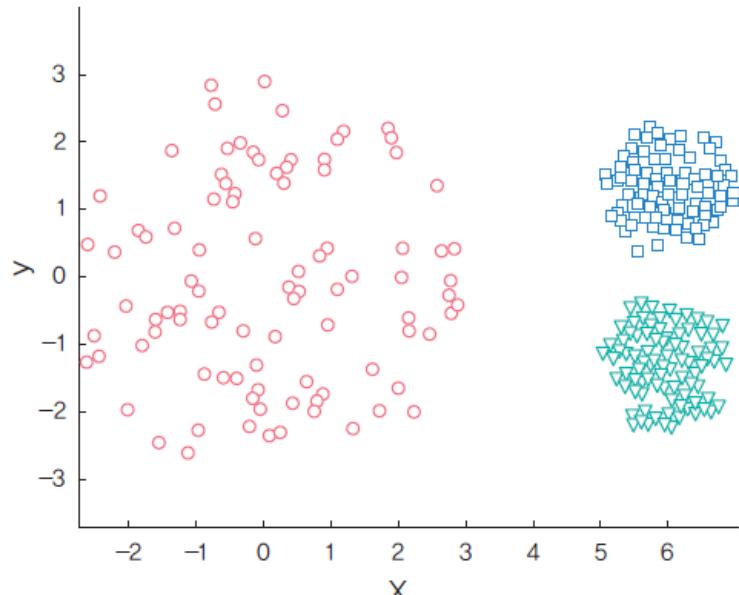


## 3.2 비지도 학습

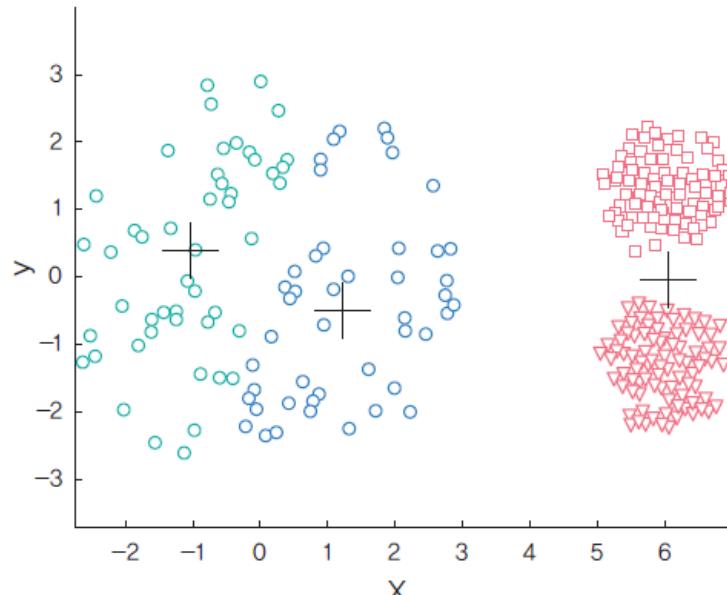
### ● K-평균 군집화

군집마다 밀집도(density)와 거리가 다를 때

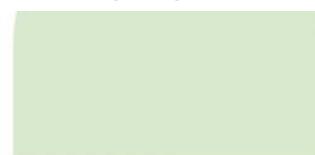
▼ 그림 3-30 밀집도와 거리가 다른 군집



주어진 데이터



K-means( $K=3$ )

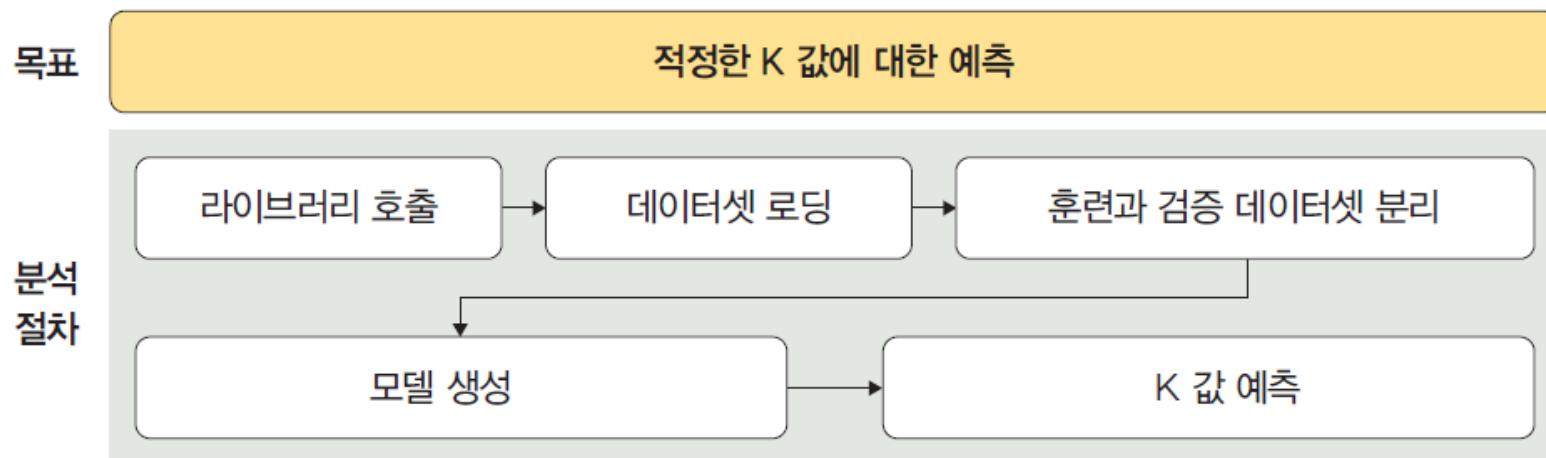


## 3.2 비지도 학습

## ● K-평균 군집화

- K-평균 군집화 예제로 자세히 알아보자
  - 앞서 살펴보았듯이 K-평균 군집화 알고리즘의 성능은 K 값에 따라 달라짐
  - 이번 예제는 적절한 K 값을 찾는 것을 목표로 진행해 보겠음

#### ▼ 그림 3-31 K-평균 군집화 예제



## 3.2 비지도 학습

### ● K-평균 군집화

- 먼저 필요한 라이브러리를 호출

코드 3-29 라이브러리 호출

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

- 내려받은 예제 파일의 data 폴더에서 상품에 대한 연 지출 데이터(sales data.csv) 파일을 불러옴

코드 3-30 상품에 대한 연 지출 데이터(sales data.csv) 호출

```
data = pd.read_csv('../chap3/data/sales data.csv')
data.head()
```

## 3.2 비지도 학습

### ● K-평균 군집화

- 코드를 실행하면 다음 그림과 같이 다양한 제품에 대한 연 지출을 확인할 수 있음

▼ 그림 3-32 K-평균 군집화 예제 데이터

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

## 3.2 비지도 학습

### ● K-평균 군집화

- 불러온 데이터셋은 도매 유통업체의 고객 데이터로 신선한 제품, 유제품, 식료품 등에 대한 고객의 연간 지출 정보가 포함되어 있음

- **Channel:** 고객 채널(호텔/레스토랑/카페) 또는 소매 채널(명목형 데이터)
- **Region:** 고객 지역(명목형 데이터)
- **Fresh:** 신선한 제품에 대한 연간 지출(연속형 데이터)
- **Milk:** 유제품에 대한 연간 지출(연속형 데이터)
- **Grocery:** 식료품에 대한 연간 지출(연속형 데이터)
- **Frozen:** 냉동 제품에 대한 연간 지출(연속형 데이터)
- **Detergents\_Paper:** 세제 및 종이 제품에 대한 연간 지출(연속형 데이터)
- **Delicassen:** 조제 식품에 대한 연간 지출(연속형 데이터)

## 3.2 비지도 학습

### ● K-평균 군집화

#### 자료 유형

- 데이터 형태에 따라 다음과 같은 유형으로 구분할 수 있음

▼ 표 3-11 자료 유형

데이터 형태	설명	예시
수치형 자료	관측된 값이 수치로 측정되는 자료	키, 몸무게, 시험 성적
연속형 자료	값이 연속적인 자료	키, 몸무게
이산형 자료	셀 수 있는 자료	자동차 사고
범주형 자료	관측 결과가 몇 개의 범주 또는 항목의 형태로 나타나는 자료	성별(남, 여), 선호도(좋다, 싫다)
순위형 자료	범주 간에 순서 의미가 있는 자료	'매우 좋다', '좋다', '그저 그렇다', '싫다', '매우 싫다' 다섯 가지 범주가 주어졌을 때, 이 범주에는 순서가 있음
명목형 자료	범주 간에 순서 의미가 없는 자료	혈액형

## 3.2 비지도 학습

### ● K-평균 군집화

- 데이터 형태에 따라 연속형 데이터와 명목형 데이터로 분류

코드 3-31 연속형 데이터와 명목형 데이터로 분류

```
categorical_features = ['Channel', 'Region'] ----- 명목형 데이터  
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper',  
'Delicassen'] ----- 연속형 데이터  
  
for col in categorical_features:  
    dummies = pd.get_dummies(data[col], prefix=col) ----- 명목형 데이터는 판다스의 get_dummies() 메서드를  
    data = pd.concat([data, dummies], axis=1)           사용하여 바이너리로 변환  
    data.drop(col, axis=1, inplace=True)  
data.head()
```

## 3.2 비지도 학습

### ● K-평균 군집화

- 코드를 실행하면 다음과 같이 연속형 데이터와 명목형 데이터로 분류

▼ 그림 3-33 예제 데이터를 연속형 데이터와 명목형 데이터로 분류

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	Channel_1	Channel_2	Region_1	Region_2	Region_3
0	12669	9656	7561	214		2674	1338	0	1	0	0
1	7057	9810	9568	1762		3293	1776	0	1	0	0
2	6353	8808	7684	2405		3516	7844	0	1	0	1
3	13265	1196	4221	6404		507	1788	1	0	0	0
4	22615	5410	7198	3915		1777	5185	0	1	0	1

## 3.2 비지도 학습

### ● K-평균 군집화

- 연속형 데이터의 모든 특성에 동일하게 중요성을 부여하기 위해 스케일링(scaling)을 적용
- 이는 데이터 범위가 다르기 때문에 범위에 따라 중요도가 달라질 수 있는 것(예를 들어 1000원과 1억 원이 있을 때 1000원의 데이터는 무시)을 방지하기 위함

코드 3-32 데이터 전처리(스케일링 적용)

```
mms = MinMaxScaler()  
mms.fit(data)  
data_transformed = mms.transform(data)
```

## 3.2 비지도 학습

### ● K-평균 군집화

- 데이터에 대한 전처리가 완료되었기 때문에 우리가 원하는 적당한 K 값을 알아보자

코드 3-33 적당한 K 값 추출

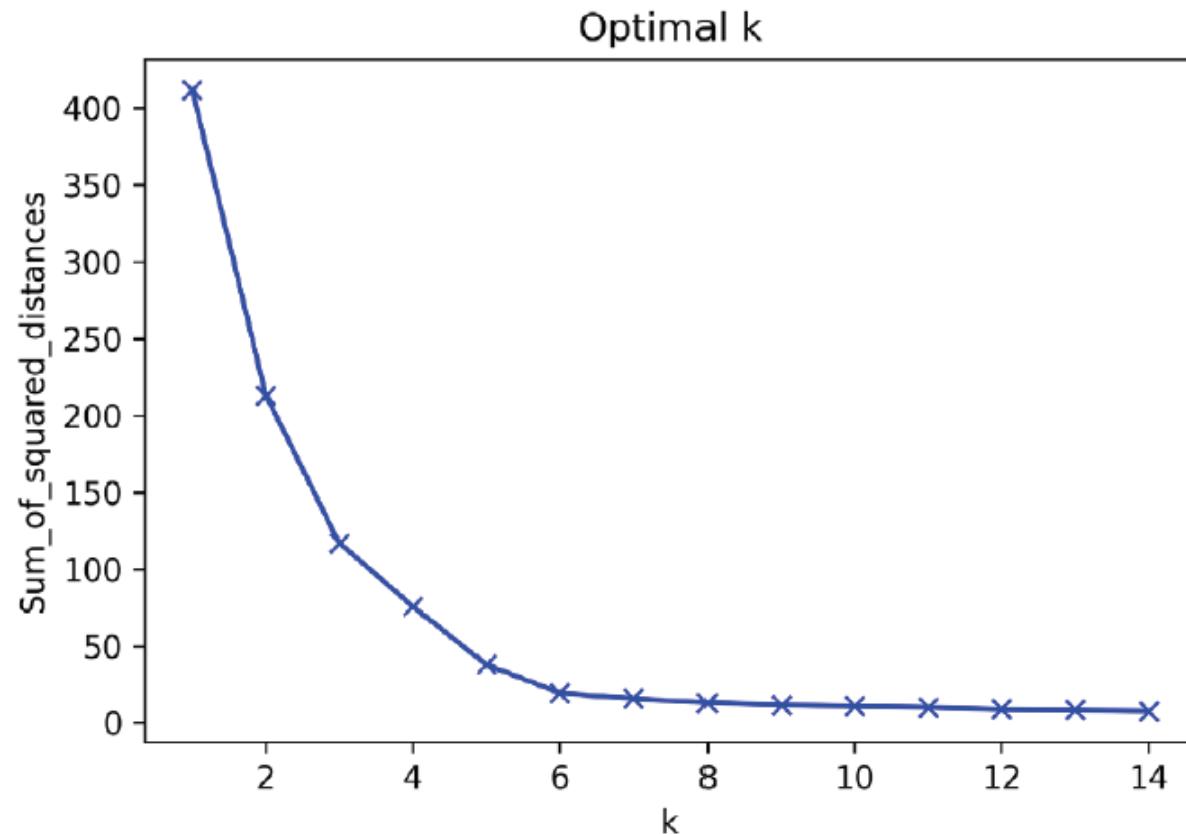
```
Sum_of_squared_distances = [] ----- ①
K = range(1, 15) ----- K에 1부터 15까지 적용해 봅니다.
for k in K:
    km = KMeans(n_clusters=k) ----- 1~15의 K 값 적용
    km = km.fit(data_transformed) ----- KMeans 모델 훈련
    Sum_of_squared_distances.append(km.inertia_)

plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Optimal k')
plt.show()
```

## 3.2 비지도 학습

### ● K-평균 군집화

- 코드를 실행하면 다음 그림과 같이 적당한 K 값이 출력
- ▼ 그림 3-34 K-평균 군집화 예제 실행 결과



## 3.2 비지도 학습

### ● K-평균 군집화

- ① 거리 제곱의 합(Sum of Squared Distances, SSD)은 x, y 두 데이터의 차를 구해서

제곱한 값을 모두 더한 후 유사성을 측정하는 데 사용

- 즉, 가장 가까운 클러스터 중심까지 거리를 제곱한 값의 합을 구할 때 사용하며, 다음 수식을 씀

$$SSD = \sum_{x,y} (I_1(x, y) - I_2(x, y))^2$$

- K가 증가하면 거리 제곱의 합은 0이 되는 경향이 있음
- K를 최댓값 n(여기에서 n은 샘플 개수)으로 설정하면 각 샘플이 자체 클러스터를 형성하여 거리 제곱 합이 0과 같아지기 때문임
- 출력 그래프는 클러스터 개수(x축)에 따른 거리 제곱의 합(y축)을 보여 줌
- K가 6부터 0에 가까워지고 있으므로 K=5가 적정하다고 판단할 수 있음

## 3.2 비지도 학습

### ● 밀도 기반 군집 분석

▼ 표 3-12 밀도 기반 군집 분석을 사용하는 이유와 적용 환경

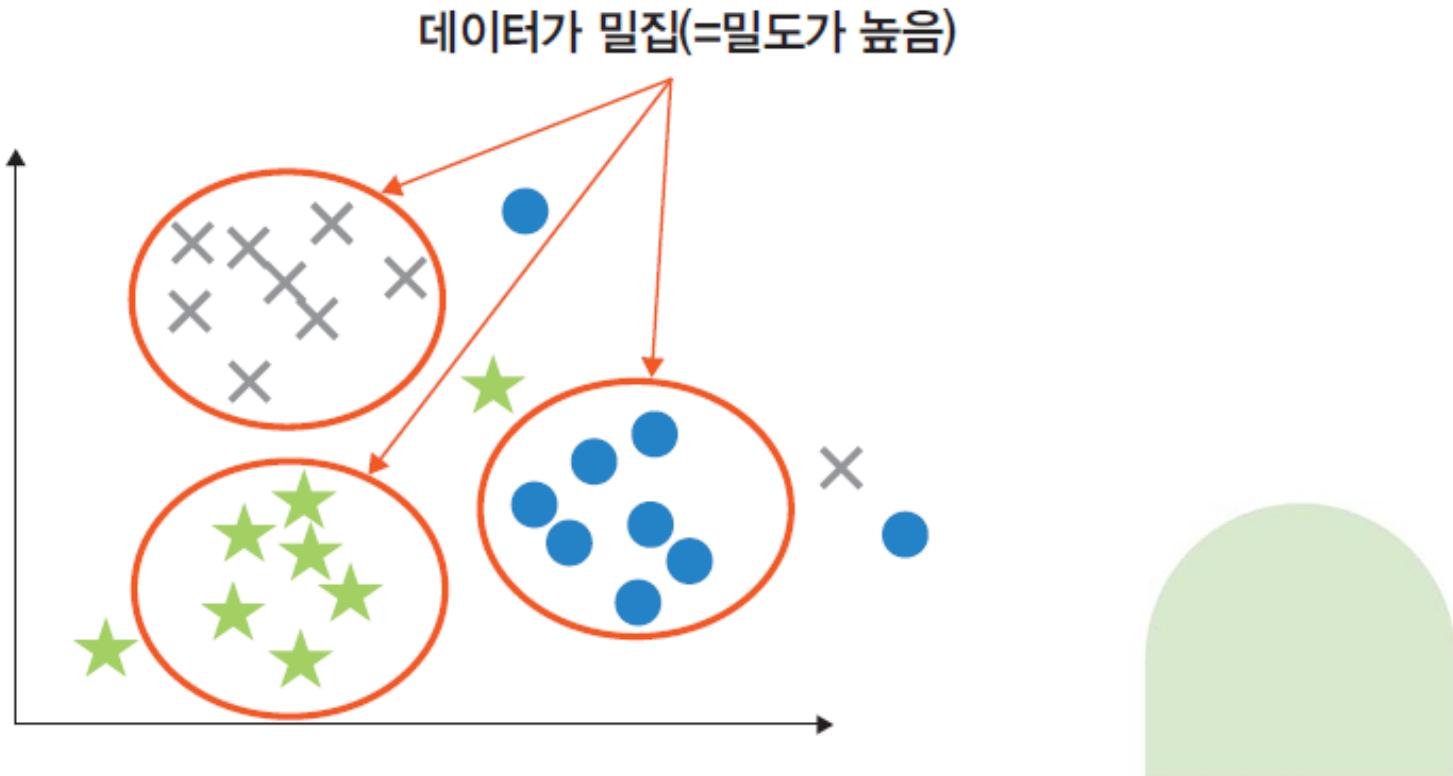
왜 사용할까?	주어진 데이터에 대한 군집화
언제 사용하면 좋을까?	K-평균 군집화와는 다르게 사전에 클러스터의 숫자를 알지 못할 때 사용하면 유용합니다. 또한, 주어진 데이터에 이상치가 많이 포함되었을 때 사용하면 좋습니다.

## 3.2 비지도 학습

### ● 밀도 기반 군집 분석

- 밀도 기반 군집 분석(Density-Based Spatial Clustering of Applications with Noise, DBSCAN)은 일정 밀도 이상을 가진 데이터를 기준으로 군집을 형성하는 방법

#### ▼ 그림 3-35 밀도 기반 군집 분석의 밀집도

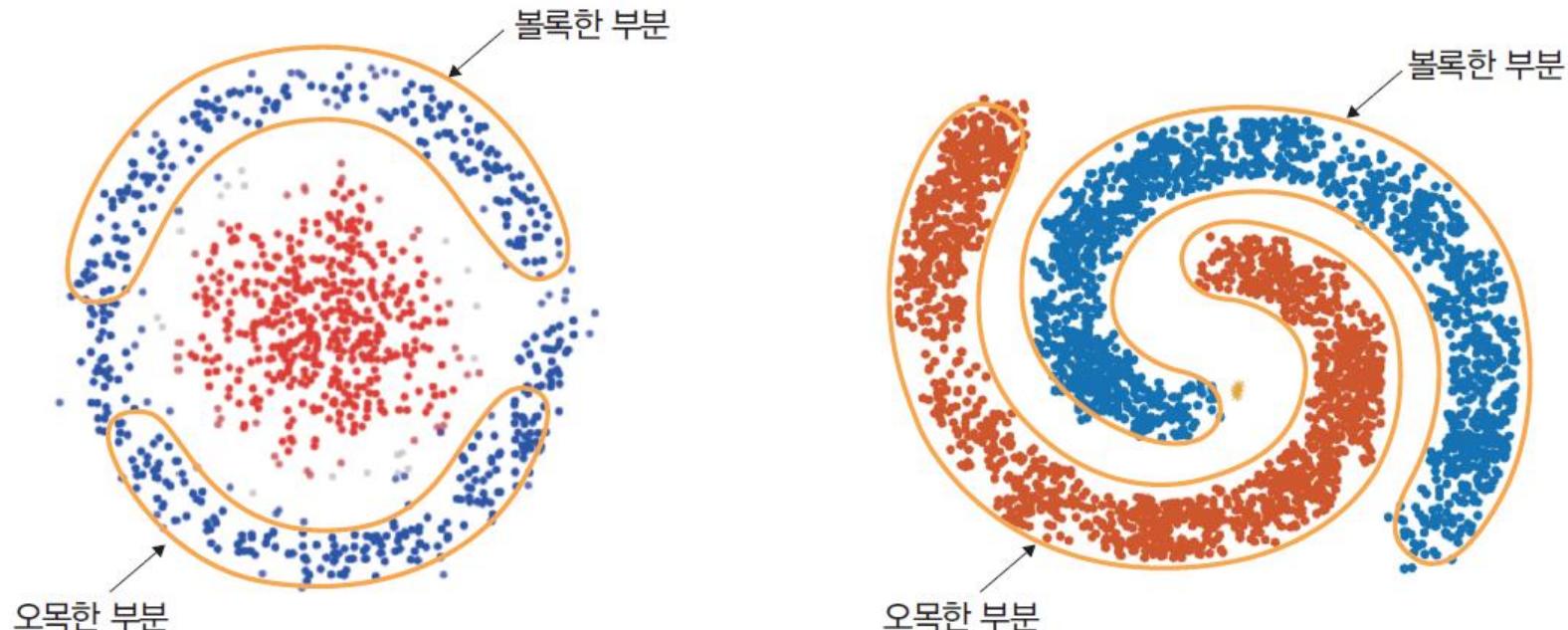


## 3.2 비지도 학습

### ● 밀도 기반 군집 분석

- 노이즈(noise)에 영향을 받지 않으며, K-평균 군집화에 비해 연산량은 많지만 K-평균 군집화가 잘 처리하지 못하는 오목하거나 볼록한 부분을 처리하는 데 유용함

#### ▼ 그림 3-36 밀도 기반 군집 분석의 데이터 표현



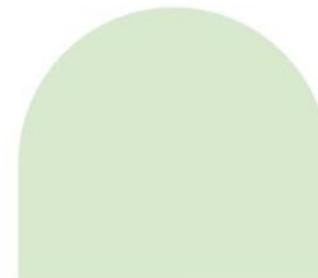
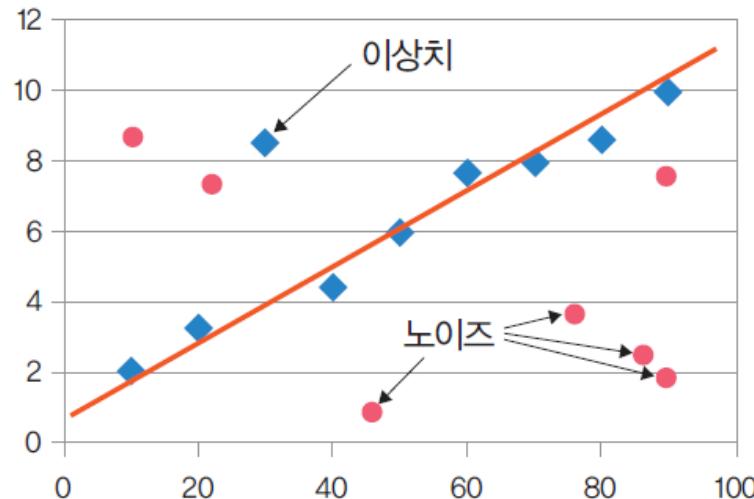
## 3.2 비지도 학습

### ● 밀도 기반 군집 분석

#### 노이즈와 이상치 차이

- 노이즈는 주어진 데이터셋과 무관하거나 무작위성 데이터로 전처리 과정에서 제거해야 할 부분
- 이상치는 관측된 데이터 범위에서 많이 벗어난 아주 작은 값이나 아주 큰 값을 의미

▼ 그림 3-37 노이즈와 이상치



## 3.2 비지도 학습

### ● 밀도 기반 군집 분석

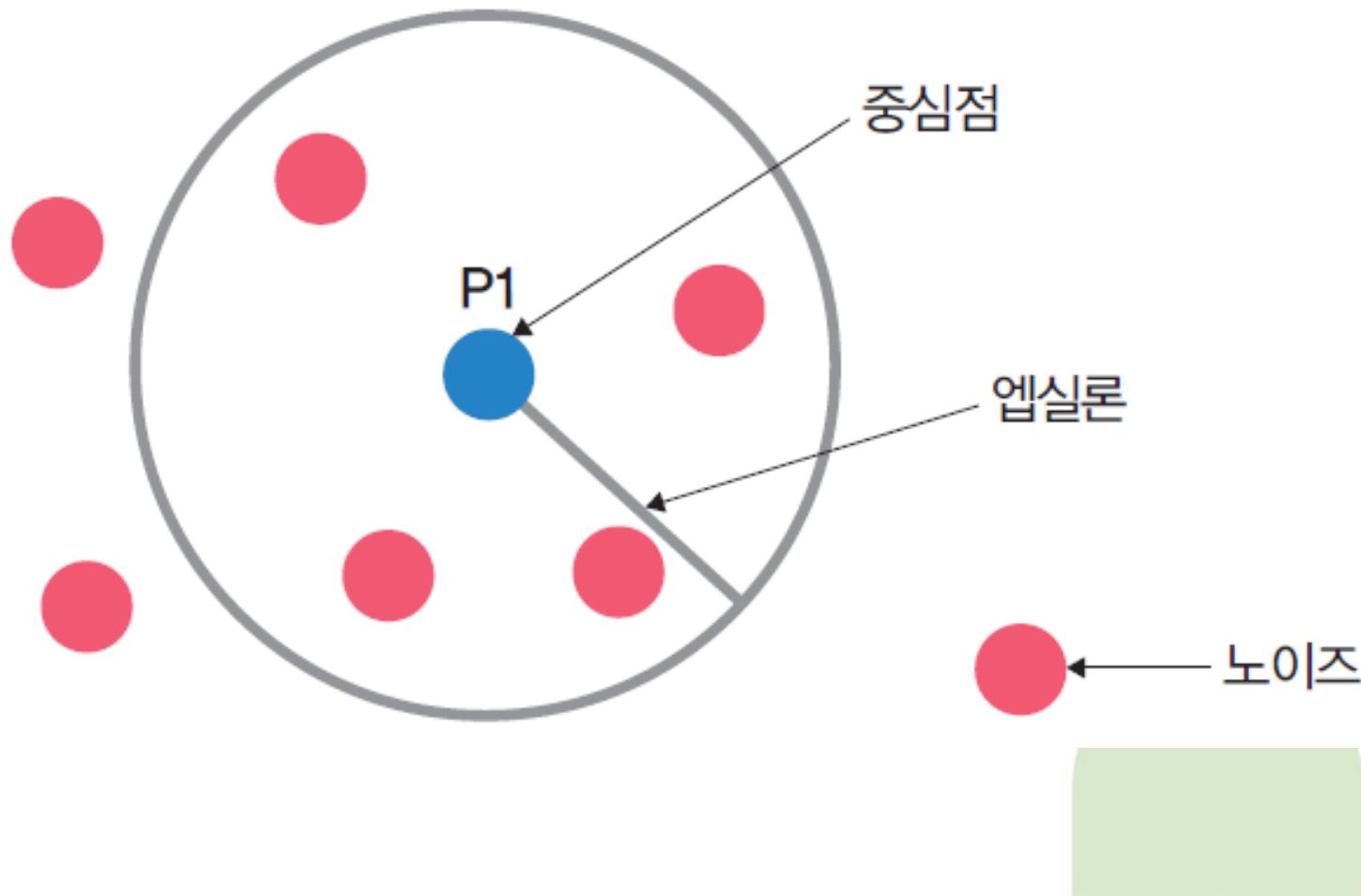
- 밀도 기반 군집 분석을 이용한 군집 방법은 다음 절차에 따라 진행

#### 1단계. 엡실론 내 점 개수 확인 및 중심점 결정

- 다음 그림과 같이 원 안에 점 P1이 있다고 할 때, 점 P1에서 거리 엡실론(epsilon) 내에 점이 m(minPts) 개 있으면 하나의 군집으로 인식한다고 하자
- 이때 엡실론 내에 점(데이터) m개를 가지고 있는 점 P1을 중심점(core point)이라고 함
- 예를 들어 minPts=3이라면 파란색 점 P1을 중심으로 반경 엡실론 내에 점이 세 개 이상 있으면 하나의 군집으로 판단할 수 있음
- 다음 그림은 점이 네 개가 있기 때문에 하나의 군집이 되고, P1은 중심점이 됨

## 3.2 비지도 학습

▼ 그림 3-38 중심점과 엡실론



## 3.2 비지도 학습

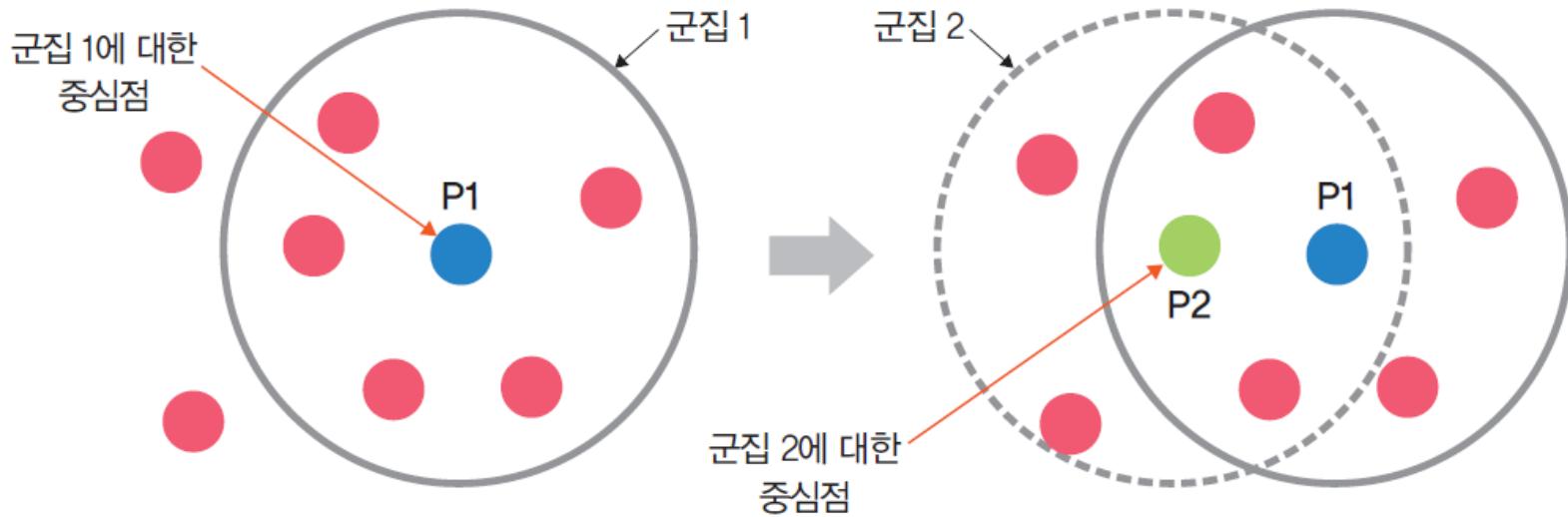
### ● 밀도 기반 군집 분석

#### 2단계. 군집 확장

- 1단계에서 새로운 군집을 생성했는데, 주어진 데이터를 사용하여 두 번째 군집을 생성해 보자
- 데이터의 밀도 기반으로 군집을 생성하기 때문에 밀도가 높은 지역에서 중심점을 만족하는 데이터가 있다면 그 지역을 포함하여 새로운 군집을 생성
- 예를 들어 P1 옆에 있던 빨간색 점(그림 3-39의 오른쪽 초록색 점)을 중심점 P2로 설정하면  $\text{minPts}=3$ 을 만족하기 때문에 새로운 군집을 생성할 수 있음

## 3.2 비지도 학습

▼ 그림 3-39 경계점



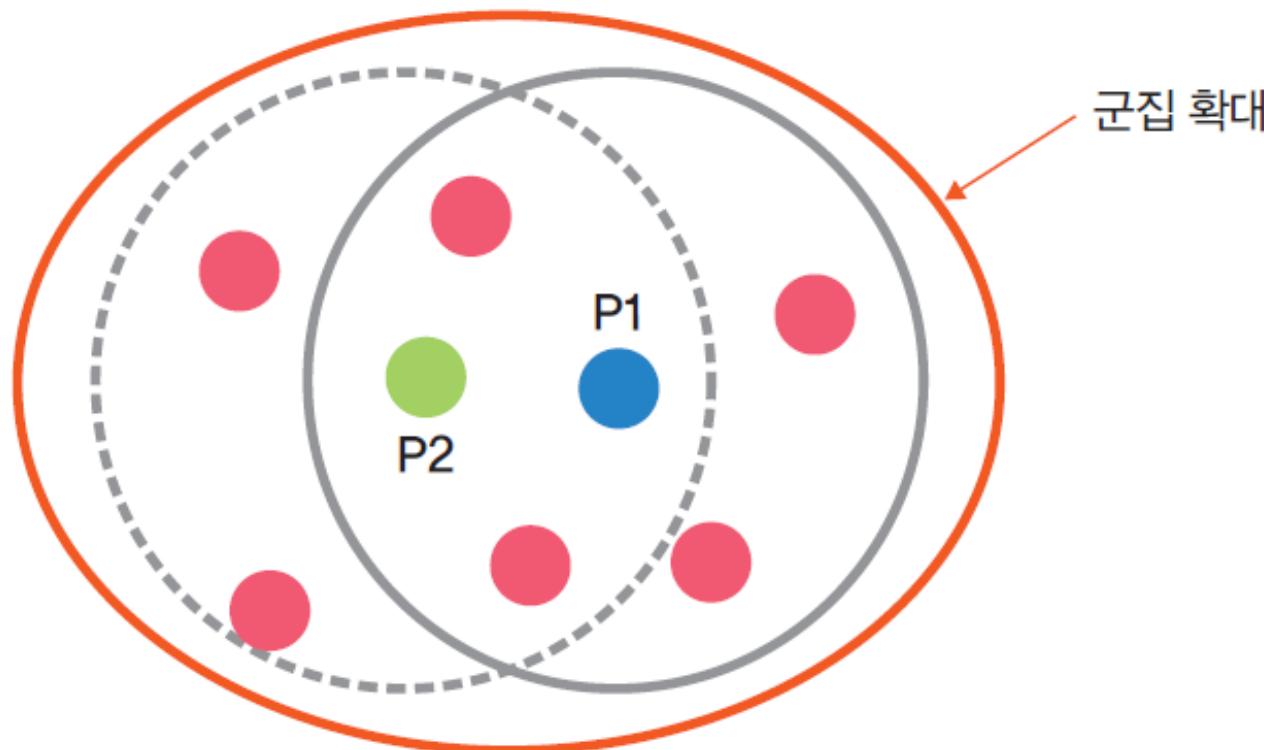
## 3.2 비지도 학습

### ● 밀도 기반 군집 분석

- 밀도 기반 군집 분석은 밀도 기반이기 때문에 주위의 점들을 대상으로 중심점을 설정하고 새로운 군집을 생성하는 것이 가능함
- 이제 군집 두 개를 하나의 군집으로 확대

## 3.2 비지도 학습

▼ 그림 3-40 군집 확대



## 3.2 비지도 학습

- 밀도 기반 군집 분석

- 3단계. 1~2단계 반복

- 데이터가 밀집된 밀도가 높은 지역에서 더 이상 중심점을 정의할 수 없을 때까지 1~2단계를 반복

- 4단계. 노이즈 정의

- 어떤 군집에도 포함되지 않은 데이터를 노이즈로 정의

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

▼ 표 3-13 PCA를 사용하는 이유와 적용 환경

왜 사용할까?	주어진 데이터의 간소화
언제 사용하면 좋을까?	현재 데이터의 특성(변수)이 너무 많을 경우에는 데이터를 하나의 플롯(plot)에 시각화해서 살펴보는 것이 어렵습니다. 이때 특성 p개를 두세 개 정도로 압축해서 데이터를 시각화하여 살펴보고 싶을 때 유용한 알고리즘입니다.

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 변수가 많은 고차원 데이터의 경우 중요하지 않은 변수로 처리해야 할 데이터양이 많아지고 성능 또한 나빠지는 경향이 있음
- 이러한 문제를 해결하고자 고차원 데이터를 저차원으로 축소시켜 데이터가 가진 대표 특성만 추출한다면 성능은 좋아지고 작업도 좀 더 간편해짐
- 이때 사용하는 대표적인 알고리즘이 PCA(Principal Component Analysis)
- 즉, PCA는 고차원 데이터를 저차원(차원 축소) 데이터로 축소시키는 알고리즘

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

차원 축소 방법은 다음과 같음

데이터들의 분포 특성을 잘 설명하는 벡터를 두 개 선택

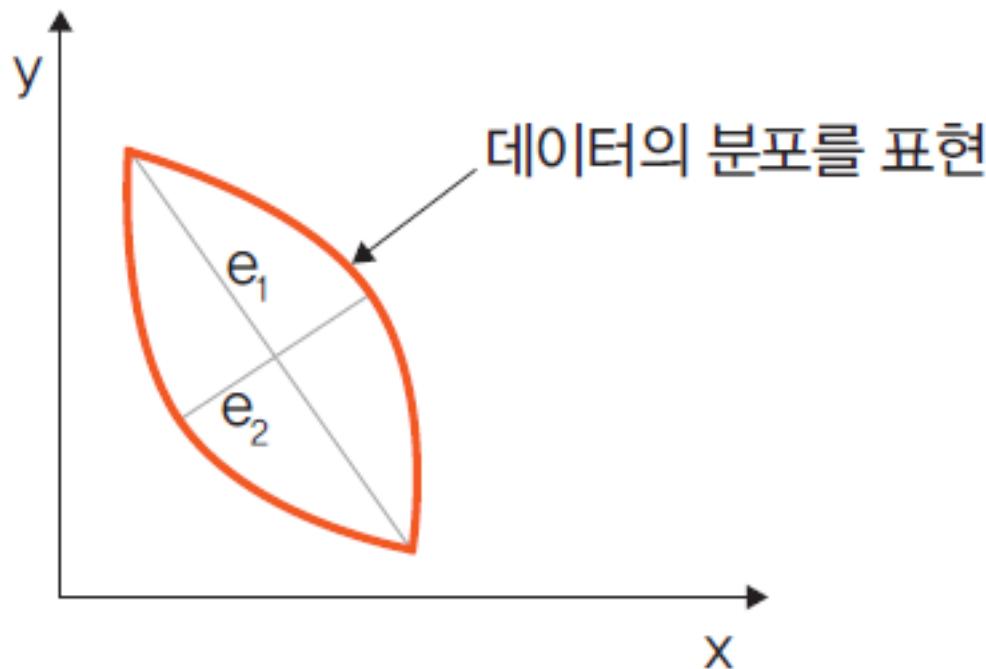
- 다음 그림에서  $e_1$ 과  $e_2$  두 벡터는 데이터 분포를 잘 설명함
- $e_1$ 의 방향과 크기,  $e_2$ 의 방향과 크기를 알면 데이터 분포가 어떤 형태인지 알 수 있기 때문임

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

벡터 두 개를 위한 적정한 가중치를 찾을 때까지 학습을 진행

▼ 그림 3-41 2D에서 PCA 예시



## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 즉, PCA는 데이터 하나하나에 대한 성분을 분석하는 것이 아니라, 여러 데이터가 모여 하나의 분포를 이룰 때 이 분포의 주성분을 분석하는 방법
- 예를 들어 코드는 간단하게 다음과 같이 구현할 수 있음

```
pca = decomposition.PCA(n_components=1)
pca_x = pca.fit_transform(x_std)

result = pd.DataFrame(pca_x, columns=[ 'dog' ])
result[ 'y-axis' ] = 0.0
result[ 'label' ] = Y

sns.lmplot('dog', 'y-axis', data=result, fit_reg=False,
           scatter_kws={"s":50}, hue='label');
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 밀도 기반 군집 분석과 PCA 예제를 끌어서 진행해 보자
- 밀도 기반 군집 분석을 이용하여 클러스터링을 진행하겠지만, 시각화를 위해 PCA를 사용해 보자
- 이번 예제의 목표는 훈련 데이터를 정확하게 클러스터링하는 것

## 3.2 비지도 학습

- ▼ 그림 3-42 밀도 기반 군집 분석과 PCA 예제

목표

훈련 데이터의 정확한 클러스터링

분석  
절차

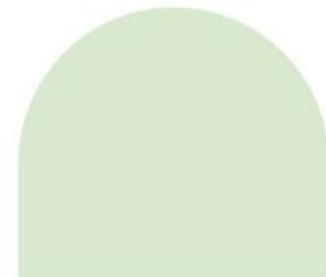
라이브러리 호출

데이터셋 로딩

데이터 전처리

모델 생성 및 훈련

클러스터링에 대한 시각화



## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 먼저 필요한 라이브러리를 호출

코드 3-34 라이브러리 호출

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN ----- 밀도 기반 군집 분석
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 훈련을 위해 내려받은 예제 파일에서 data 폴더에 있는 credit card.csv 파일을

코드 3-35 데이터 불러오기

```
X = pd.read_csv('..../chap3/data/credit card.csv')
X = X.drop('CUST_ID', axis=1) ----- 불러온 데이터에서 'CUST_ID' 열(칼럼)을 삭제
X.fillna(method='ffill', inplace=True) ----- ①
print(X.head()) ----- 데이터셋 형태 확인
```

---

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 코드를 실행하면 credit card.csv 파일의 데이터셋 정보를 보여 줌

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
0	95.4	0.000000	0.166667	
1	0.0	6442.945483	0.000000	
2	0.0	0.000000	1.000000	
3	0.0	205.788017	0.083333	
4	0.0	0.000000	0.083333	

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
0	0.000000	0.083333	
1	0.000000	0.000000	
2	1.000000	0.000000	
3	0.083333	0.000000	
4	0.083333	0.000000	

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	\
0	0.000000	0	2	1000.0	
1	0.250000	4	0	7000.0	
2	0.000000	0	12	7500.0	
3	0.083333	1	1	7500.0	
4	0.000000	0	1	1200.0	

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0	201.802084	139.509787	0.000000	12
1	4103.032597	1072.340217	0.222222	12
2	622.066742	627.284787	0.000000	12
3	0.000000	627.284787	0.000000	12
4	678.334763	244.791237	0.000000	12

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- ① 결측 값을 앞의 값으로 채움
- 예를 들어 df.fillna(method='ffill')을 실행할 경우 다음과 같이 앞의 값으로 결측치가 채워짐

▼ 그림 3-43 df.fillna( ) 메서드

	Data1	Data2	Data13
0		0.2	0.8
1		0.5	
2	0.2		0.6
3	0.3		



df.fillna(method='ffill')

	Data1	Data2	Data13
0	NaN	0.2	0.8
1	NaN	0.5	0.8
2	0.2	0.5	0.6
3	0.3	0.5	0.6

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 데이터 전처리 및 차원 축소를 진행

코드 3-36 데이터 전처리 및 데이터를 2차원으로 차원 축소

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) ----- 평균이 0, 표준편차가 1이 되도록 데이터 크기를 조정

X_normalized = normalize(X_scaled) ----- 데이터가 가우스 분포를 따르도록 정규화
X_normalized = pd.DataFrame(X_normalized) ----- 넘파일 배열을 데이터프레임(dataframe)으로 변환

pca = PCA(n_components=2) ----- 2차원으로 차원 축소 선언
X_principal = pca.fit_transform(X_normalized) ----- 차원 축소 적용
X_principal = pd.DataFrame(X_principal)
X_principal.columns = [ 'P1' , 'P2' ]
print(X_principal.head())
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 다음은 데이터를 2차원으로 차원 축소한 결과

	P1	P2
0	-0.489949	-0.679976
1	-0.519099	0.544827
2	0.330633	0.268880
3	-0.481656	-0.097611
4	-0.563512	-0.482506

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 훈련된 모델에 대해 시각적으로 표현해 보자

코드 3-37 DBSCAN 모델 생성 및 결과의 시각화

```
db_default = DBSCAN(eps=0.0375, min_samples=3).fit(X_principal) ----- 모델 생성 및 훈련  
labels = db_default.labels_ ----- 각 데이터 포인트에 할당된 모든 클러스터 레이블의 넘파일 배열을 labels에 저장  
  
colours = {} ----- 출력 그래프의 색상을 위한 레이블 생성  
colours[0] = 'y'  
colours[1] = 'g'  
colours[2] = 'b'  
colours[-1] = 'k'  
  
cvec = [colours[label] for label in labels] ----- 각 데이터 포인트에 대한 색상 벡터 생성  
  
r = plt.scatter(X_principal['P1'], X_principal['P2'], color='y');  
g = plt.scatter(X_principal['P1'], X_principal['P2'], color='g');
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

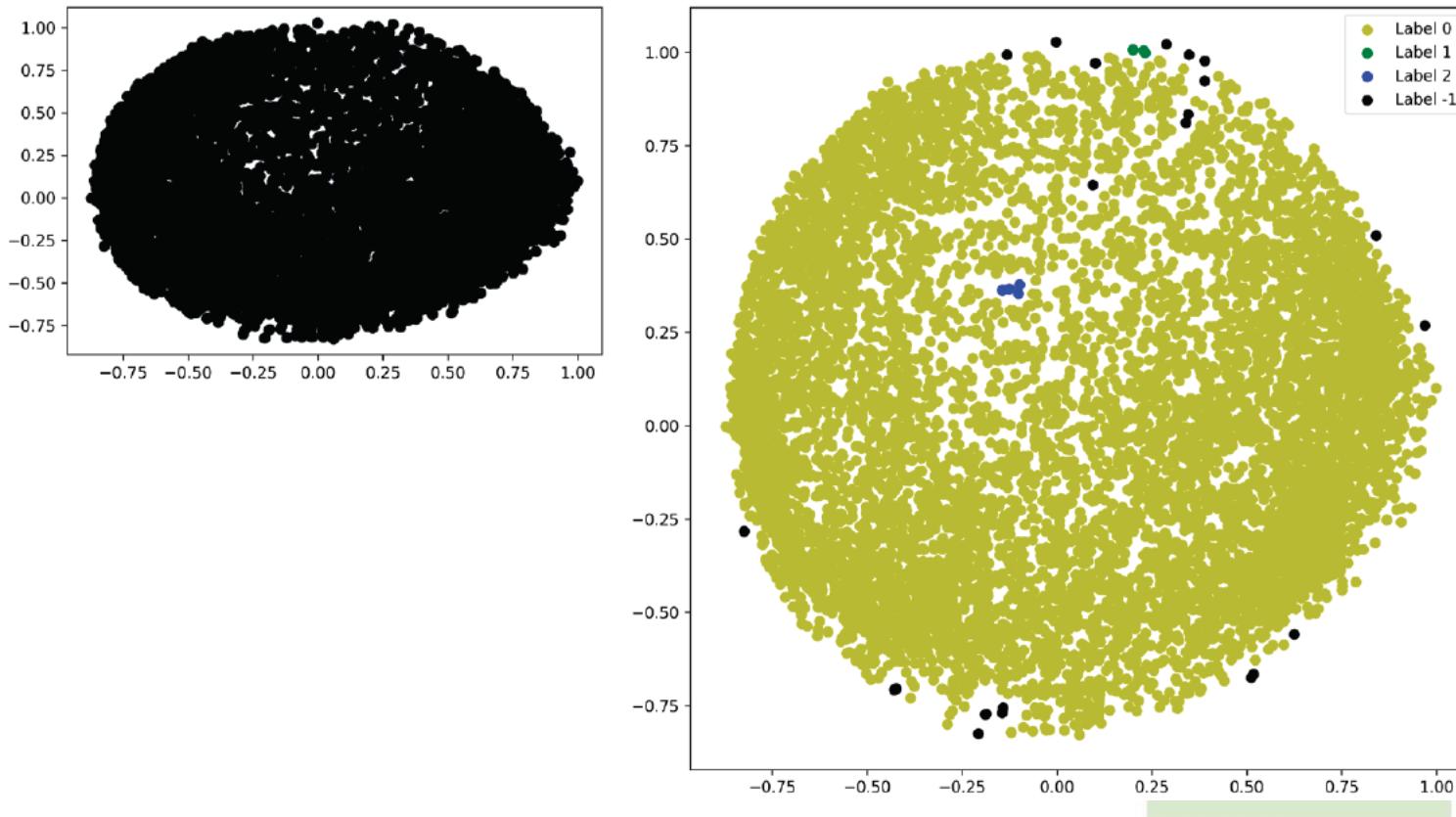
```
b = plt.scatter(X_principal['P1'], X_principal['P2'], color='b');  
k = plt.scatter(X_principal['P1'], X_principal['P2'], color='k'); ----- 플롯(plot)의  
                                            범례(legend) 구성  
  
plt.figure(figsize=(9,9))  
plt.scatter(X_principal['P1'], X_principal['P2'], c=cvec) ----- 정의된 색상 벡터에 따라 X축에  
                                            P1, Y축에 P2 플로팅(plotting)  
  
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -1')) ----- 범례 구축  
plt.show()
```

---

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 다음 그림은 DBSCAN 모델을 실행하여 시각화한 결과
- ▼ 그림 3-44 밀도 기반 군집 분석과 PCA 예제 실행 결과



## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 출력 결과를 보면 알겠지만, 클러스터링에 대한 튜닝이 필요함
- 밀도 기반 군집 분석에서 사용하는 min-samples(minPts)의 하이퍼파라미터를 3에서 50으로 변경한 후 시각화 부분을 수정해보겠음

코드 3-38 모델 튜닝

```
db = DBSCAN(eps=0.0375, min_samples=50).fit(X_principal)
labels1 = db.labels_

colours1 = {}
colours1[0] = 'r'
colours1[1] = 'g'
colours1[2] = 'b'
colours1[3] = 'c'
colours1[4] = 'y'
colours1[5] = 'm'
colours1[-1] = 'k'
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

```
cvec = [colours1[label] for label in labels1]
colors1 = ['r', 'g', 'b', 'c', 'y', 'm', 'k']

r = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[0])
g = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[1])
b = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[2])
c = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[3])
y = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[4])
m = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[5])
k = plt.scatter(
    X_principal['P1'], X_principal['P2'], marker='o', color=colors1[6])
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

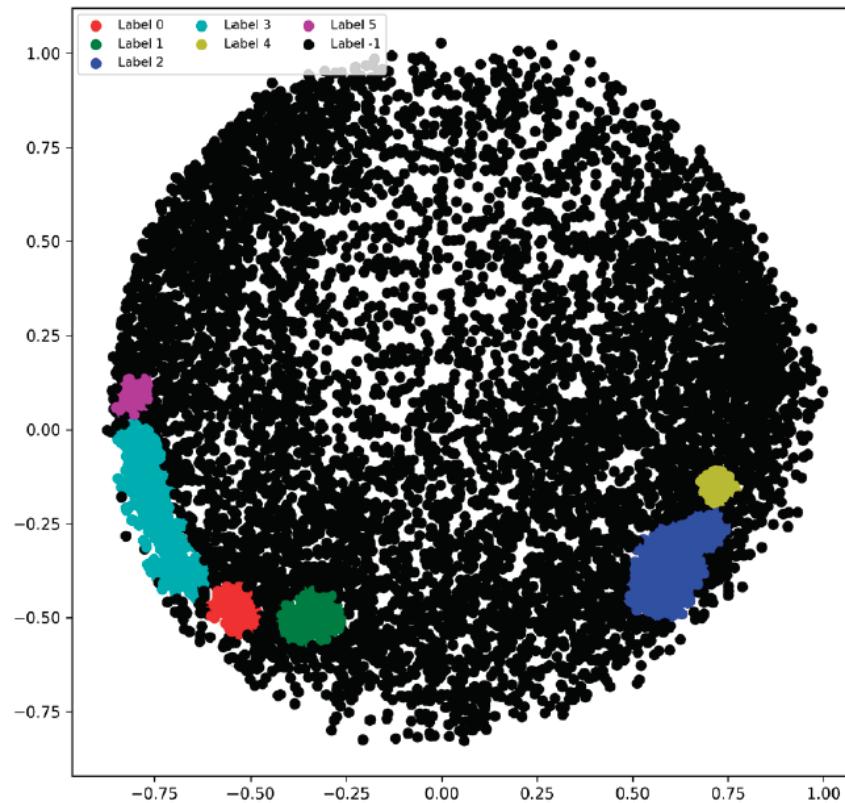
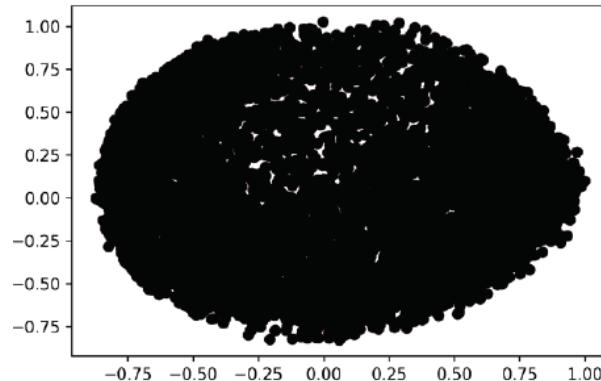
```
plt.figure(figsize=(9,9))
plt.scatter(X_principal['P1'], X_principal['P2'], c=cvec)
plt.legend((r, g, b, c, y, m, k),
           ('Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label -1'),
           scatterpoints=1,
           loc='upper left',
           ncol=3,
           fontsize=8)
plt.show()
```

---

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 다음 그림은 모델 튜닝을 실행한 결과
- ▼ 그림 3-45 밀도 기반 군집 분석과 PCA 예제 튜닝 결과



## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 앞에서 진행했던 코드(코드 3-37)보다 군집이 잘 표현되었음
- 추가적으로 밀도 기반 군집 분석 모델의 하이퍼파라미터 인자 min\_samples를 50에서 100으로 변경해 보면 그림 3-46과 같은 그래프를 출력

코드 3-39 min\_samples를 50에서 100으로 변경

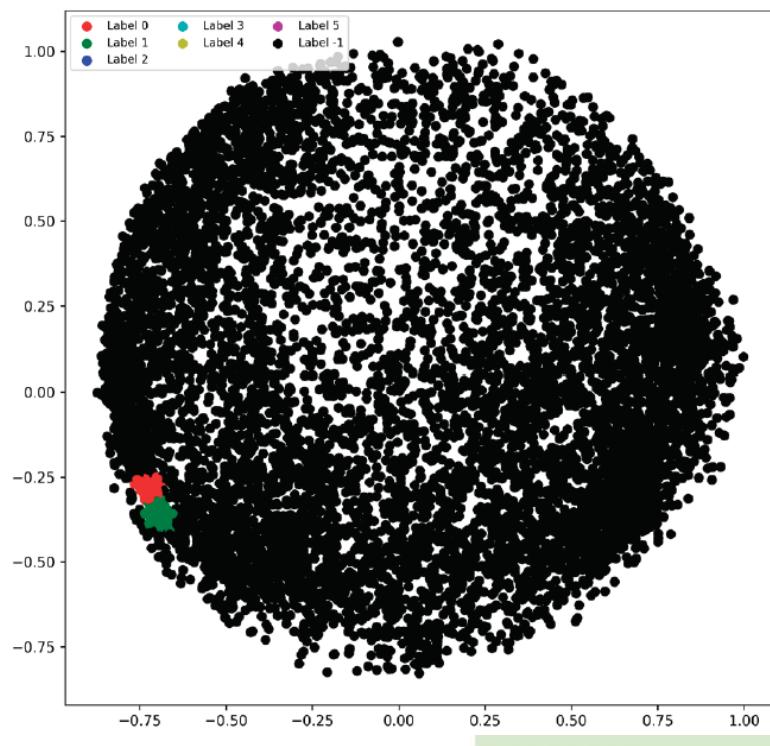
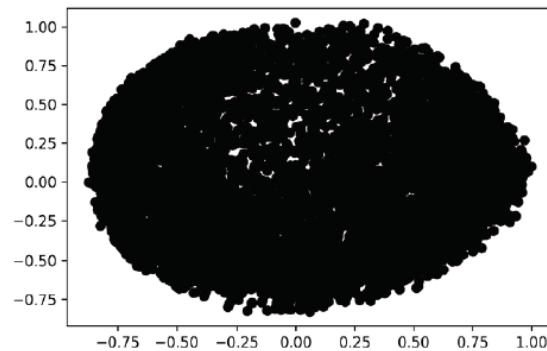
```
db = DBSCAN(eps=0.0375, min_samples=100).fit(X_principal)
```

## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 코드를 실행하면 그림 3-46과 같이 출력

▼ 그림 3-46 밀도 기반 군집 분석과 PCA 예제에서 잘못된 하이퍼파라미터를 적용



## 3.2 비지도 학습

### ● 주성분 분석(PCA)

- 많은 클러스터 부분이 무시된 것을 확인할 수 있음
- 이와 같이 모델에서 하이퍼파라미터 영향에 따라 클러스터 결과(성능)가 달라지므로, 최적의 성능을 내려면 하이퍼파라미터를 이용한 튜닝이 중요함