6장. 기타 주요 머신러닝 알고리즘

1. **지도 학습**
    1. K-최근접 이웃(K-nearest neighbor)
    2. 서포트 벡터 머신(Support Vector Machine, SVM)
    3. 결정 트리(decision tree)
2. **비지도 학습**
    1. 군집(clustering)
    2. 차원 축소(dimensionality reduction)

1. 지도학습

❖ 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
- **표 3-1 분류와 회귀 차이**

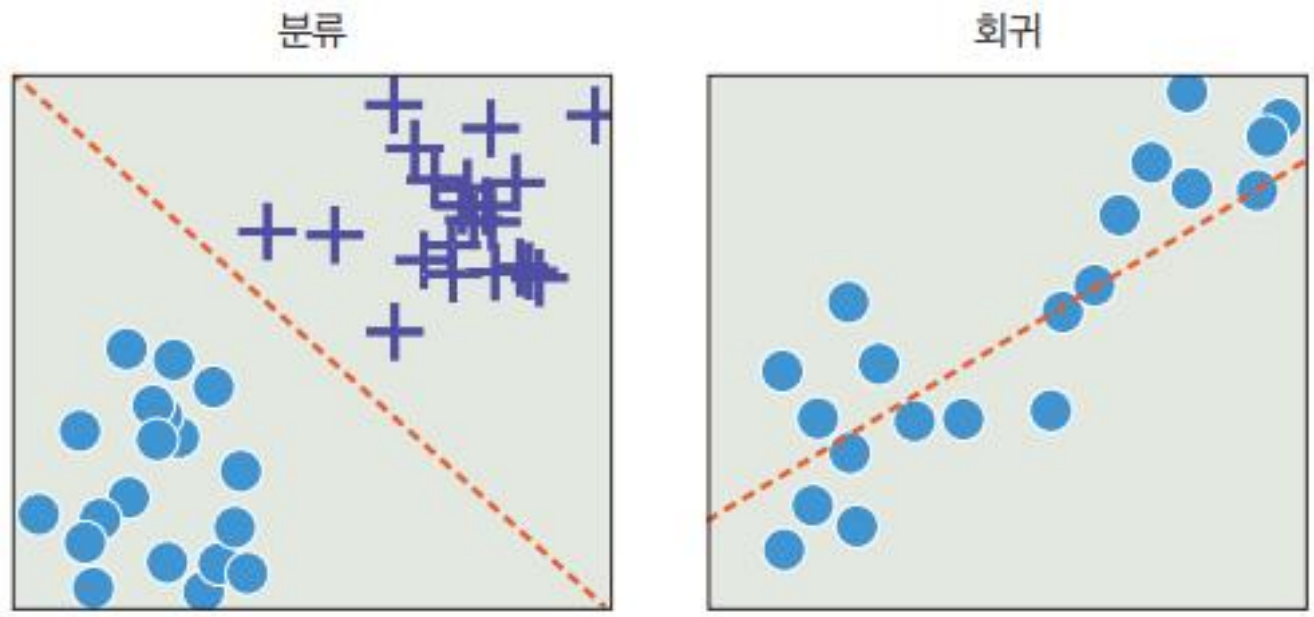
구분	분류	회귀
데이터 유형	이산형 데이터	연속형 데이터
결과	훈련 데이터의 레이블 중 하나를 예측	연속된 값을 예측
예시	학습 데이터를 A · B · C 그룹 중 하나로 매핑 예 스팸 메일 필터링	결괏값이 어떤 값이든 나올 수 있음 예 주가 분석 예측

1. 지도학습

❖ 지도 학습

- 다음 그림을 보면 분류와 회귀 차이를 좀 더 명확히 알 수 있음

▼ 그림 1 분류와 회귀



1. 지도학습

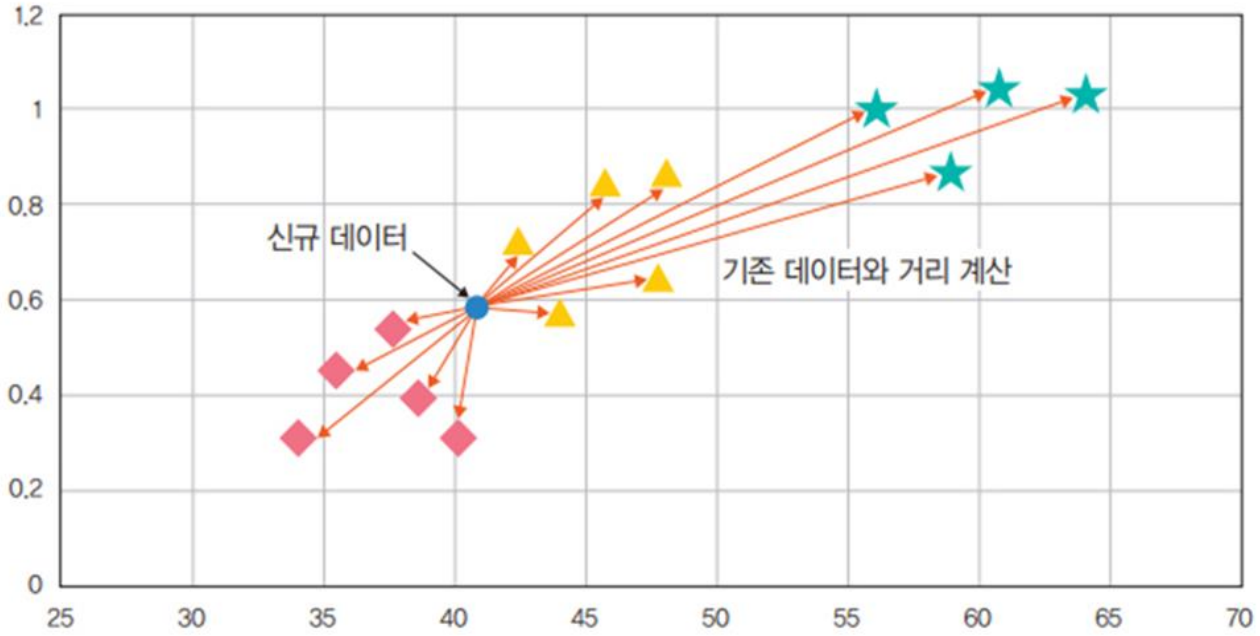
❖ K-최근접 이웃(K-nearest neighbor KNN)

- K-최근접 이웃(K-nearest neighbor)은 새로운 입력(학습에 사용하지 않은 새로운 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

1. 지도학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 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, 임의로 지정)를 선택하여 해당 클러스터에 할당



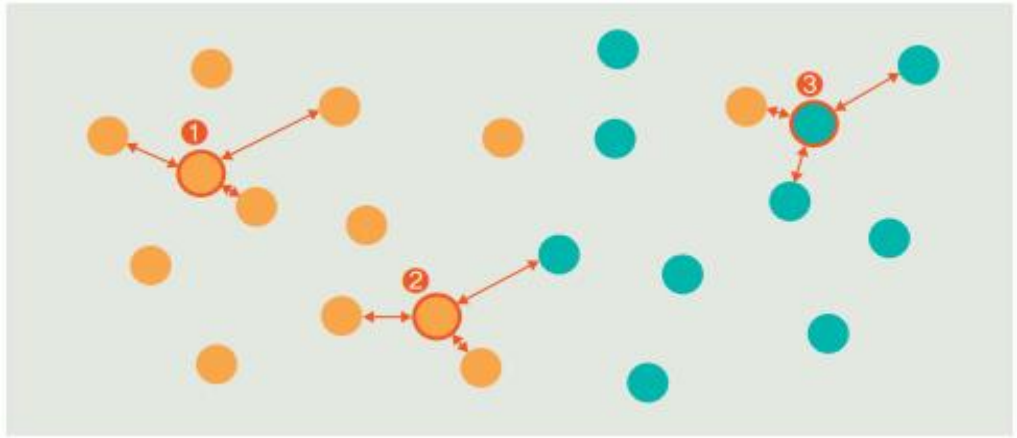
▼ 그림 2 K-최근접 이웃

1. 지도 학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 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

▼ 그림 3 K-최근접 이웃 학습 절차

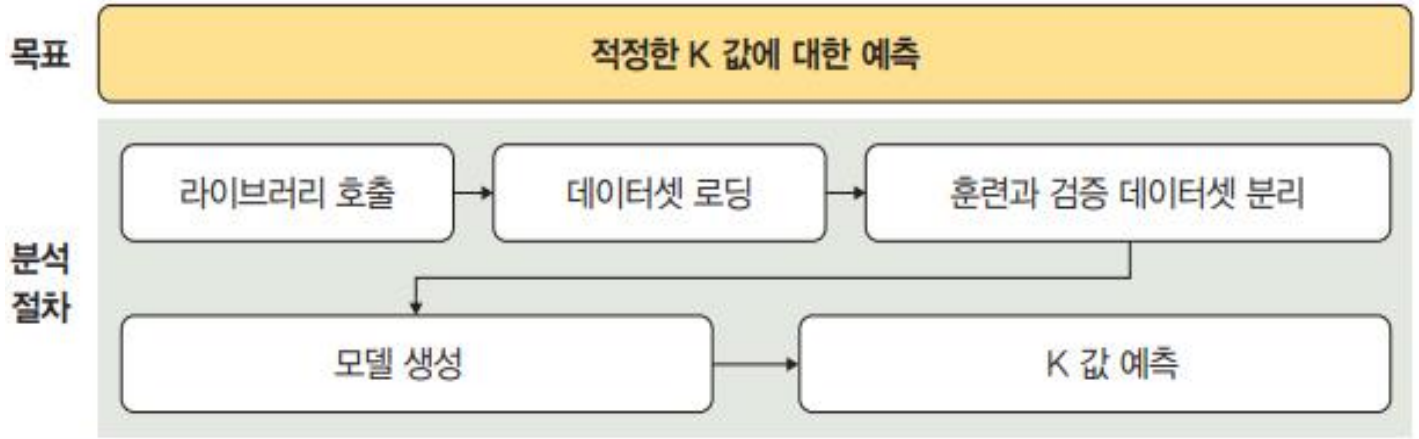


1. 지도 학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 예제 목표는 붓꽃에 대한 분류(iris.data 파일 사용)
- 참고로 머신 러닝 코드는 심층 신경망이 필요하지 않기 때문에 사이킷런(scikit-learn)을 이용
- 다음 과정으로 K 값을 예측할 것

▼ 그림 4 K-최근접 이웃 예제



1. 지도 학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 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

```
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'] #데이터 셋의 열이름
dataset = pd.read_csv('data/iris.data', names=names) #판다스 데이터프레임으로 데이터 읽어오기
dataset
```

1. 지도학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 데이터를 전처리하고 훈련과 테스트 데이터셋으로 분리

```
X = dataset.iloc[:, :-1].values # 모든 행의 마지막 열의 데이터를 제외한 모든 데이터를 X에 저장
y = dataset.iloc[:, 4].values  # 모든 행의 마지막 데이터만 선택하여 y에 저장
print(X)
print(y)

from sklearn.model_selection import train_test_split # train-data와 test-data 분리 라이브러리
# 80:20으로 X,y를 train data와 test data로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

1. 지도학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 데이터 스케일링

```
from sklearn.preprocessing import StandardScaler
s = StandardScaler() # 값을 스케일링(scaling), 평균:0, 표준편차:1이 되도록 변환

X_train = s.fit_transform(X_train) # X_train 스케일링 처리
X_test = s.fit_transform(X_test)  # X_test 스케일링 처리
print(X_train)
print(X_test)
```

1. 지도학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 모델 생성과 학습

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

모델 생성 및 훈련에 대한 결과 ➡

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=50)
```

- 모델에 대한 정확도 측정

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

출력 결과 : `정확도: 0.9`

- train_test_split() 메서드는 데이터를 무작위로 분할하므로 코드를 실행할 때마다 정확도에 차이가 있음
- 여러 차례 실행한 후 평균을 찾는 것이 좋음
- K=50일 때 예측 값이 약 90%로, 수치가 높음

1. 지도학습

❖ K-최근접 이웃(K-nearest neighbor KNN)

- 최적의 K 값을 구하고 정확도 찾기 : For 문을 이용하여 K 값을 1부터 10까지 순환하면서 최적의 K 값과 정확도를 찾음

```
k=10
acc_array=np.zeros(k)
for k in np.arange(1,k+1,1):
    classifier = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
    y_pred = classifier.predict(X_test)
    acc = metrics.accuracy_score(y_test, y_pred)
    acc_array[k-1]=acc

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

```
결과 => 정확도 0.9666666666666667 으로 최적의 k는 3 입니다.
```

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

표 3-3 서포트 벡터 머신을 사용하는 이유와 적용 환경

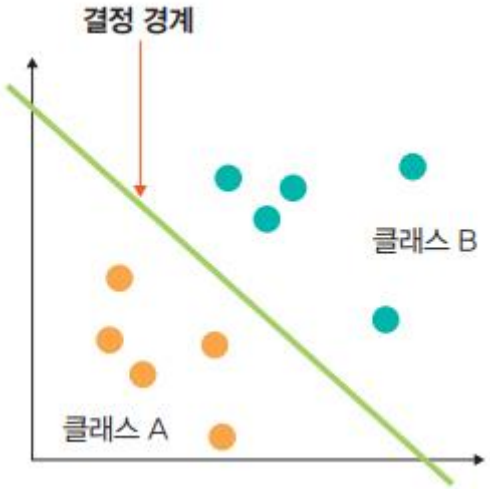
왜 사용할까?	주어진 데이터에 대한 분류
언제 사용하면 좋을까?	서포트 벡터 머신은 커널만 적절히 선택한다면 정확도가 상당히 좋기 때문에 정확도를 요구하는 분류 문제를 다룰 때 사용하면 좋습니다. 또한, 텍스트를 분류할 때도 많이 사용합니다.

- 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 중요

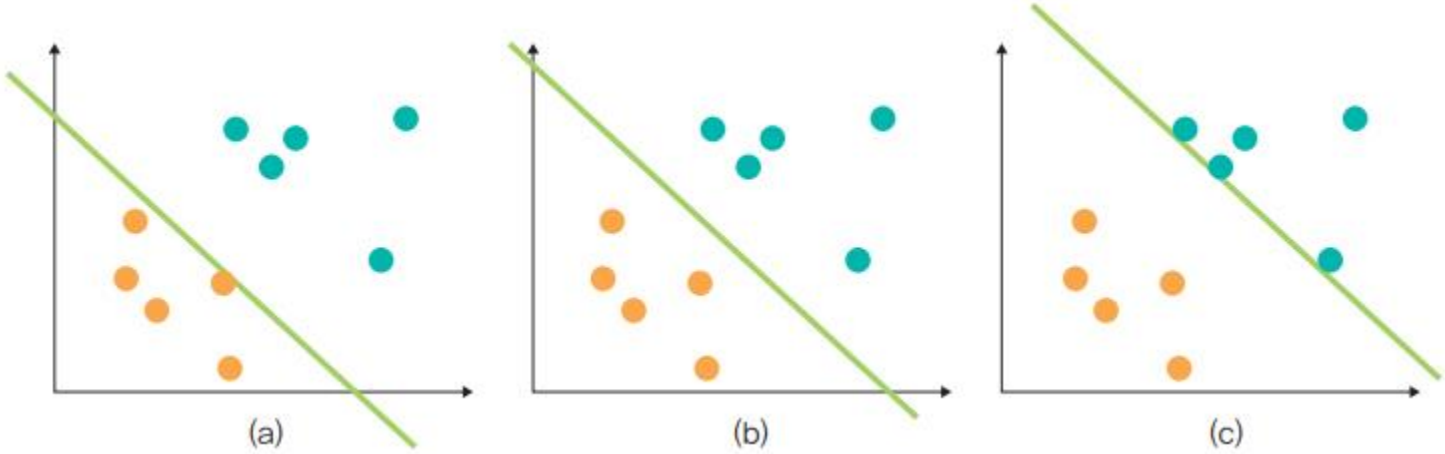
1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 결정 경계는 데이터를 분류하기 위한 기준선
- 다음 그림과 같이 주황색 공과 녹색 공이 있을 때 이 공들을 색상별로 분류하기 위한 기준선이 결정 경계
- 결정 경계는 어디에 위치하면 가장 좋을까? 그림 6의 (a)~(c) 중에서 어떤 그림이 가장 안정적으로 보이나?, (b)가 가장 안정적으로 보이지 않나?



▼ 그림 5 서포트 벡터 머신 결정 경계

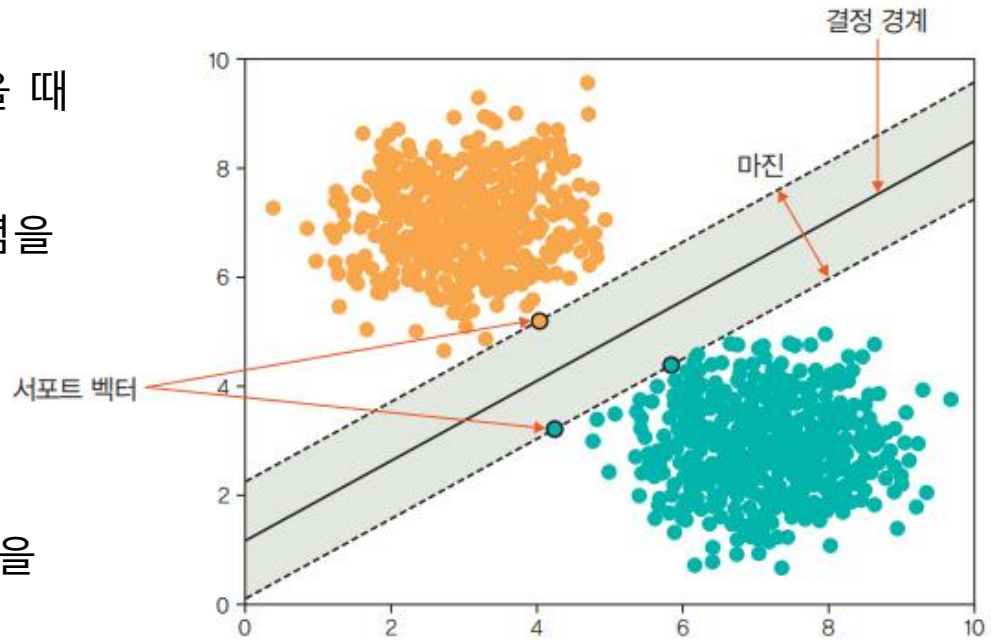


▼ 그림 6 서포트 벡터 머신 결정 경계의 위치 결정

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 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
- 그럼 서포트 벡터는 무엇일까?
- 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

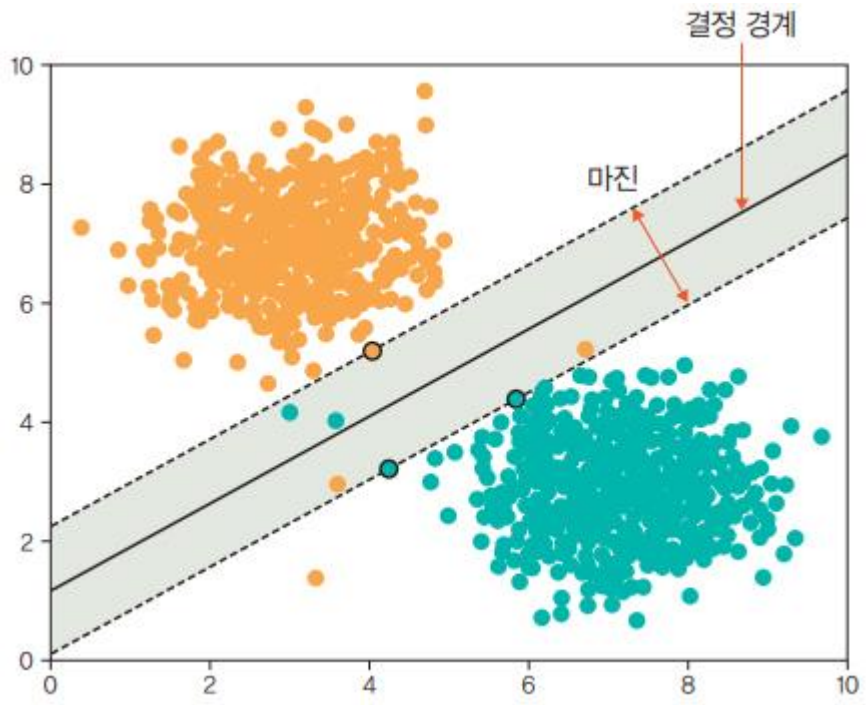


▼ 그림 7 서포트 벡터 머신의 서포트 벡터

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 서포트 벡터 머신은 데이터들을 올바르게 분리하면서 마진 크기를 최대화해야 하는데, 결국 이상치(outlier)를 잘 다루는 것이 중요
- 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 7이 이상치를 허용하지 않는 하드 마진이라면, 그림 8은 이상치를 허용하는 소프트 마진



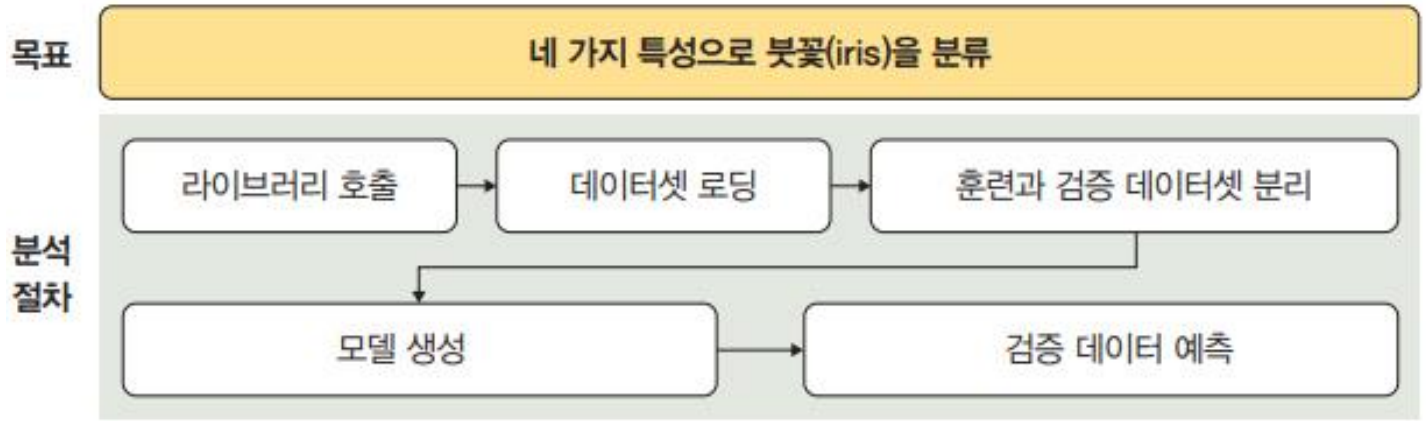
▼ 그림 8 서포트 벡터 머신의 마진

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

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

▼ 그림 3-9 서포트 벡터 머신 예제



1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 훈련에 필요한 데이터를 로드하고 필요한 라이브러리를 호출

```
! pip install tensorflow  # tensorflow 라이브러리 설치
```

```
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 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
```

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 데이터셋을 로드하여 훈련과 테스트셋으로 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_splict 메서드를 사용하여
# 훈련 데이터셋과 테스트 데이터 셋을 40:60으로 분리

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 사이킷런으로 SVM 모델을 생성 및 훈련시킨 후 테스트 데이터셋을 이용한 예측을 수행
- 모델에 대한 정확도 확인

```
svm = svm.SVC(kernel='linear', C=1.0, gamma=0.5) # (1)
svm.fit(x_train, y_train) # 훈련데이터를 사용하여 SVM 분류기 훈련
predictions = svm.predict(x_test)
score = metrics.accuracy_score(y_test, predictions)
print('정확도: {0:f}'.format(score)) # 테스트 데이터(예측) 정확도 측정
```

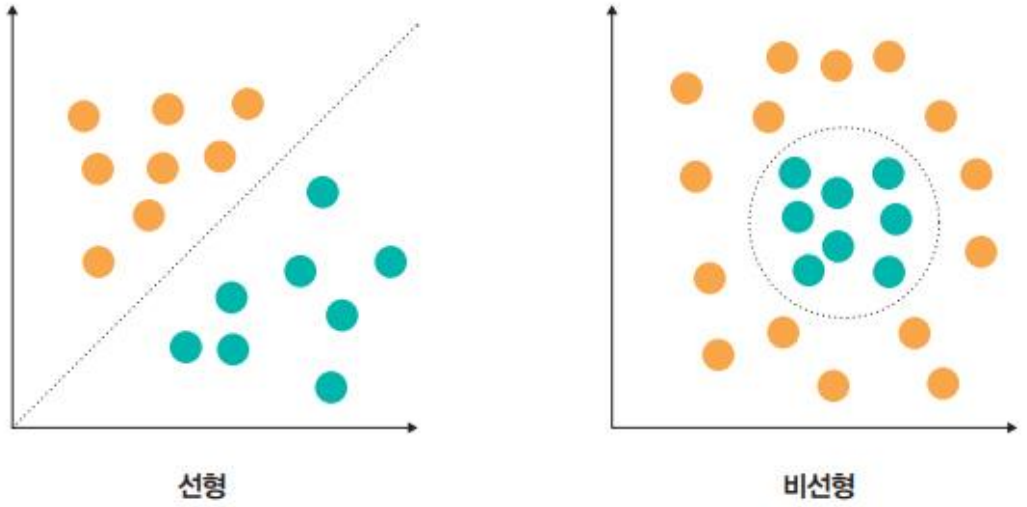
실행결과 => 정확도: 0.988889

1. 지도 학습

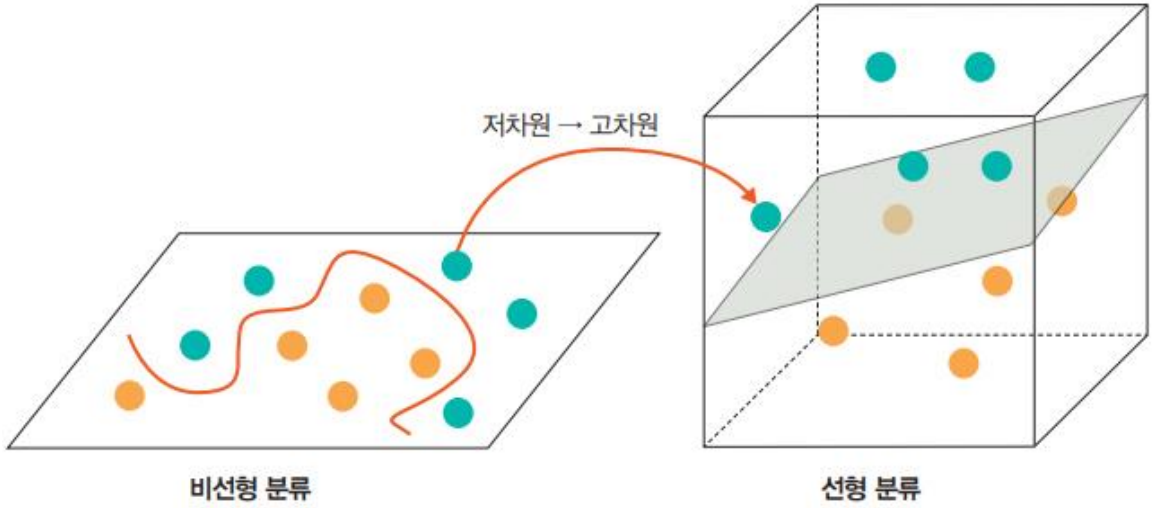
❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- ① SVM은 선형 분류와 비선형 분류를 지원
- 비선형에 대한 커널은 선형으로 분류될 수 없는 데이터들 때문에 발생
- 비선형 문제를 해결하는 가장 기본적인 방법은 저차원 데이터를 고차원으로 보내는 것인데, 이것은 많은 수학적 계산이 필요하기 때문에 성능에 문제를 줄 수 있음

▼ 그림 10 선형과 비선형 분류



▼ 그림 11 비선형과 선형 분류



1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

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

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

(a, b: 입력 벡터)

  - 선형 커널은 기본 커널 트릭으로 커널 트릭을 사용하지 않겠다는 의미와 일맥상통

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 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 가능

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

(a, b: 입력 벡터
γ: 감마
d: 차원, 이때 γ, d는 하이퍼파라미터)

- 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

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

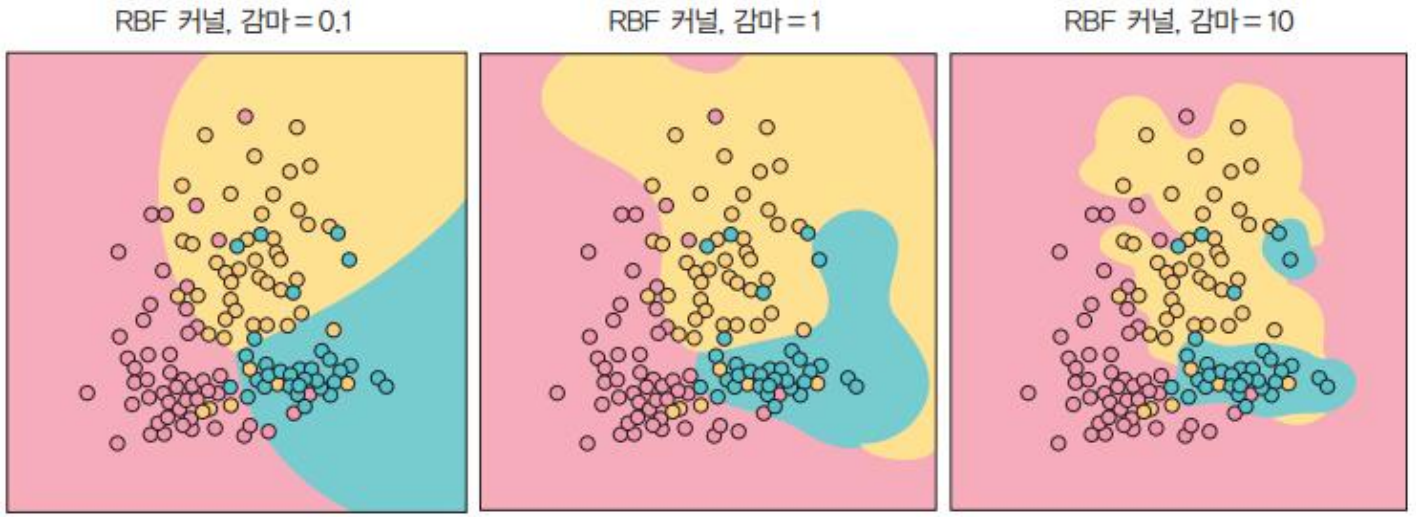
(이때 γ는 하이퍼파라미터)

1. 지도 학습

❖ 서포트 벡터 머신 (Support Vector Machine, SVM)

- 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 지정
- 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 초래

▼ 그림 12 서포트 벡터 머신 커널의 감마 값에 따른 변화



1. 지도 학습

❖ 결정 트리(decision tree)

▼ 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.

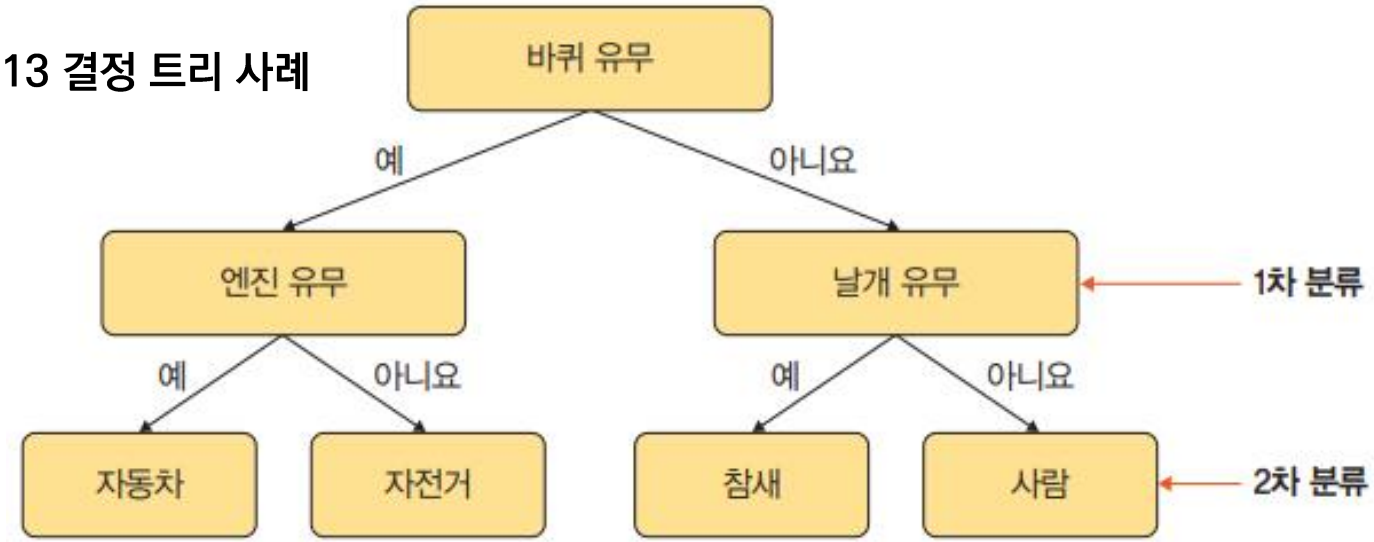
- 결정 트리(decision tree)는 데이터를 분류하거나 결괏값을 예측하는 분석 방법
- 결과 모델이 트리 구조이기 때문에 결정 트리라고 함
- 다음 그림은 결정 과정을 보여 줌

1. 지도 학습

❖ 결정 트리(decision tree)

- 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

▼ 그림 13 결정 트리 사례



1. 지도 학습

❖ 결정 트리(decision tree)

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

▼ 그림 3-14 순도와 불순도

1. 지도 학습

❖ 결정 트리(decision tree)

- 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

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)

1. 지도 학습

❖ 결정 트리(decision tree)

엔트로피(entropy)

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

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

1. 지도 학습

❖ 결정 트리(decision tree)

지니 계수(Gini index)

- 불순도를 측정하는 지표로, 데이터의 통계적 분산 정도를 정량화해서 표현한 값
- 즉, 지니 계수는 원소 n개 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)

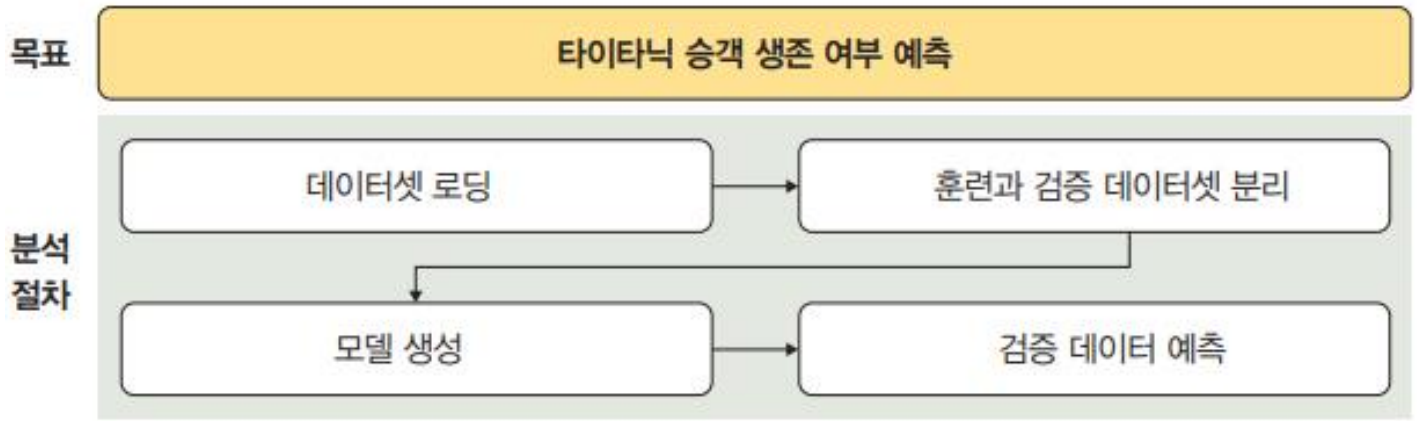
- 지니 계수는 로그를 계산할 필요가 없어 엔트로피보다 계산이 빠르기 때문에 결정 트리에서 많이 사용

1. 지도 학습

❖ 결정 트리(decision tree)

- 예제의 목표는 타이타닉 승객의 생존 여부를 예측하는 것

▼ 그림 3-15 결정 트리 예제



1. 지도 학습

❖ 결정 트리(decision tree)

- 예제 파일의 data 폴더에 있는 train.csv 파일을 사용

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

- 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

             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
```

1. 지도 학습

❖ 결정 트리(decision tree)

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

```
df = df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Survived']]
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df = df.dropna()
X = df.drop('Survived', axis=1)
y = df['Survived']
```

- 훈련과 테스트 데이터셋으로 분리

```
# 승객의 생존 여부를 예측하기 위해 '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 0또는 1로 변환
df = df.dropna() # 값이 없는 데이터 삭제
X = df.drop('Survived', axis=1) # 'Survivied'를 정답으로 사용하기 위해 X에서 제거
y = df['Survived'] # 'Survivied'를 정답(target) 레이블로 사용
```

1. 지도 학습

❖ 결정 트리(decision tree)

- 훈련과 테스트셋으로 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 결정트리 라이브러리 사용 모델 생성

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

- 모델 훈련

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

모델훈련에 대한 실행 결과 =>

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

1. 지도 학습

❖ 결정 트리(decision tree)

- 테스트 데이터셋을 이용하여 모델에 대한 예측을 진행

```
y_predict = model.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_predict)
```

- 예측결과 `0.8268156424581006`
- 결과가 83%로 높은 수치를 보이고 있음
- 즉, 학습이 잘되었음

1. 지도 학습

❖ 결정 트리(decision tree)

- 혼동 행렬

```
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']
)
```

	Predicted Not Survival	Predicted Survival
True Not Survival	98	14
True Survival	17	50

1. 지도 학습

❖ 결정 트리

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

▼ 표 5 혼동 행렬

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

- 혼동 행렬에서 사용하는 용어를 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

1. 지도 학습

❖ 결정 트리

- 혼동 행렬을 이용하면 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 17 혼동 행렬 훈련 결과

	Predicted Not Survival	Predicted Survival
True Not Survival	98	14
True Survival	17	50

- 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)

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

▼ 표 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	• 데이터 압축 • 중요한 속성 도출

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
- 군집, 군집화, 클러스터 용어를 혼용하여 사용하지만, 모두 동일한 의미로 이해하면 됨

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 참고

2. 비지도 학습

❖ K-평균 군집화

▼ 표 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.

2. 비지도 학습

❖ K-평균 군집화

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

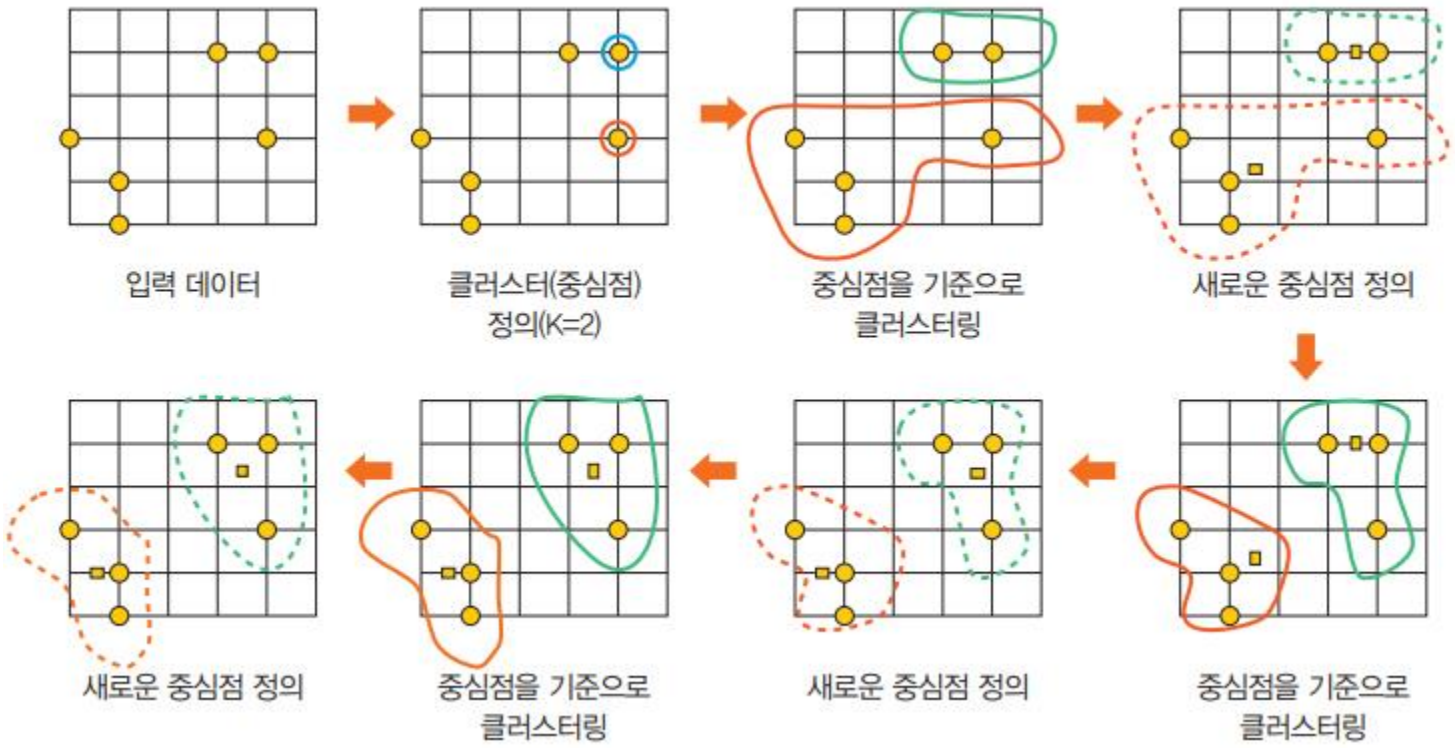
1. 중심점 선택: 랜덤하게 초기 중심점(centroid)을 선택(그림에서는 K=2로 초기화)
2. 클러스터 할당: K개의 중심점과 각각의 개별 데이터 간의 거리(distance)를 측정한 후,
   가장 가까운 중심점을 기준으로 데이터를 할당(assign)
   이 과정을 통해 클러스터가 구성(이때 클러스터링은 데이터를 하나 혹은 둘 이상의
   덩어리로 묶는 과정이며, 클러스터는 덩어리 자체를 의미)
3. 새로운 중심점 선택: 클러스터마다 새로운 중심점을 계산
4. 범위 확인(convergence): 선택된 중심점에 더 이상의 변화가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

2. 비지도 학습

❖ K-평균 군집화

- 다음 그림은 반복 횟수에 따른 데이터 분류 과정을 보여 줌

▼ 그림 27 K-평균 군집화



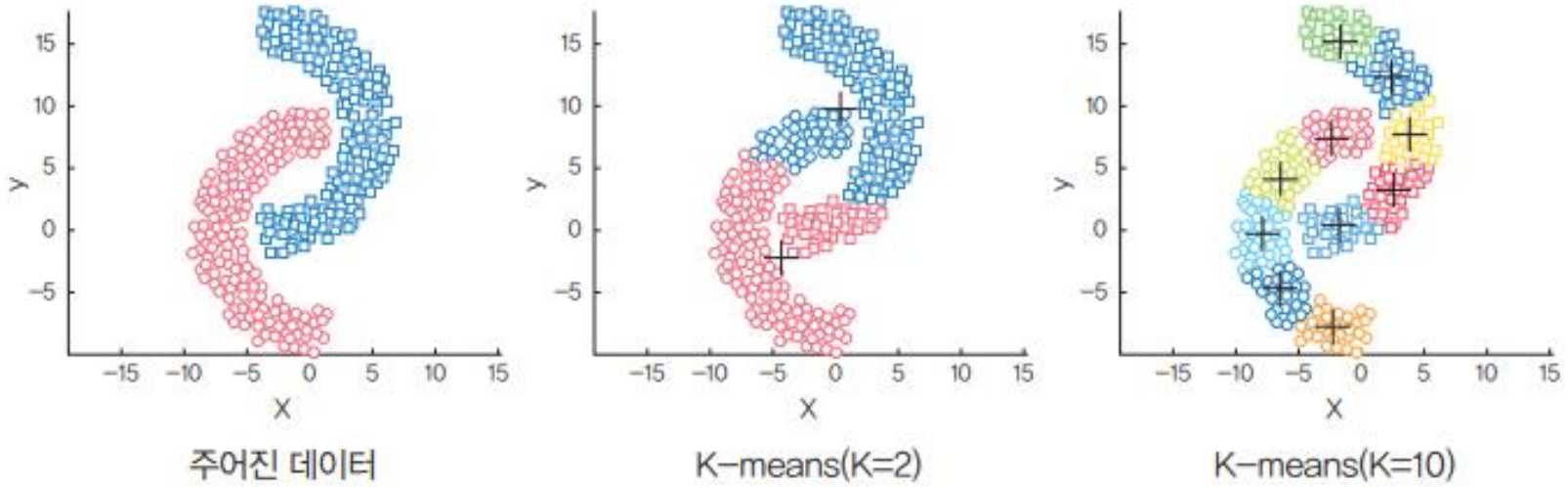
2. 비지도 학습

❖ 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 28 비선형 데이터



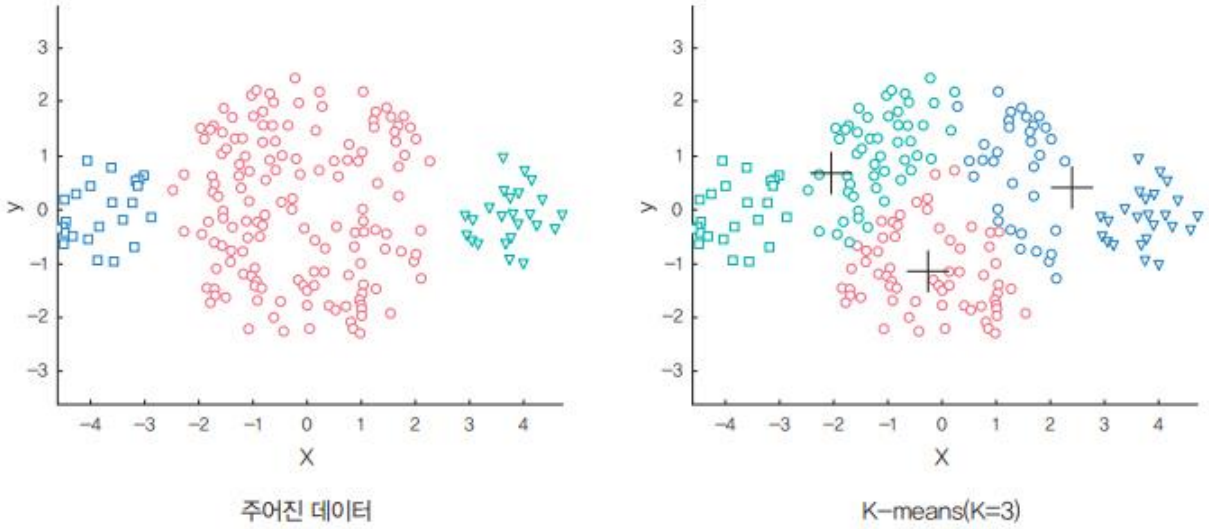
2. 비지도 학습

❖ K-평균 군집화

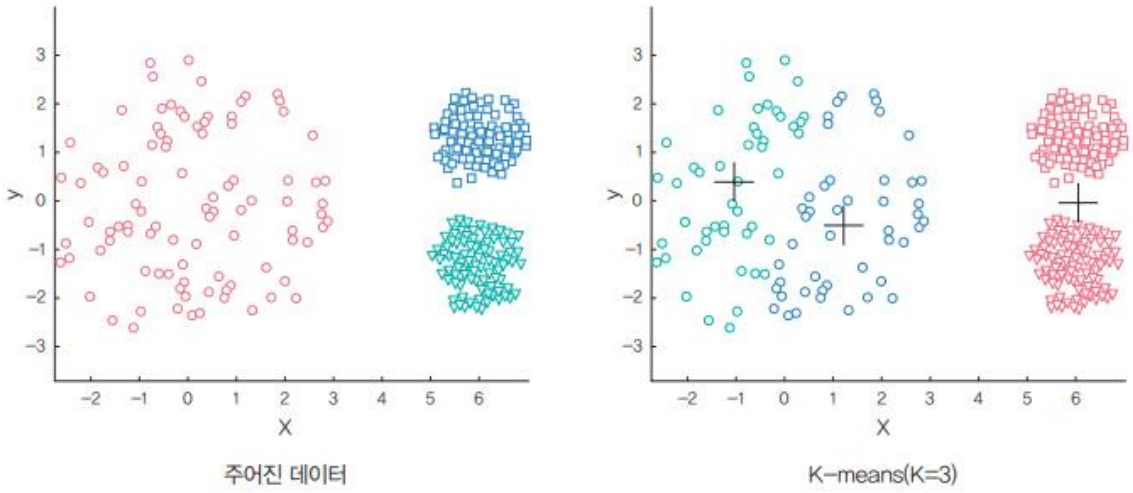
군집 크기가 다를 때

군집마다 밀집도(density)와 거리가 다를 때

▼ 그림 29 서로 다른 군집 크기



▼ 그림 3-30 밀집도와 거리가 다른 군집

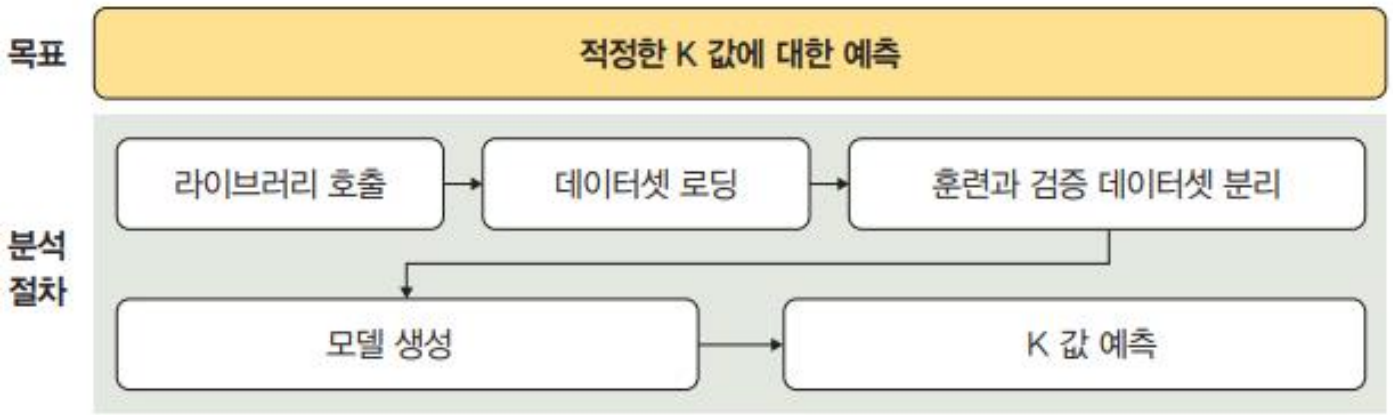


2. 비지도 학습

❖ 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 보자

▼ 그림 3-31 K-평균 군집화 예제



2. 비지도 학습

❖ K-평균 군집화

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

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

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

	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

2. 비지도 학습

❖ K-평균 군집화

- 불러온 데이터셋은 도매 유통업체의 고객 데이터로 신선한 제품, 유제품, 식료품 등에 대한 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)

2. 비지도 학습

❖ K-평균 군집화

자료 유형

- 데이터 형태에 따라 다음과 같은 유형으로 구분할 수 있음

▼ 표 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

2. 비지도 학습

❖ K-평균 군집화

- 데이터 형태에 따라 연속형 데이터와 명목형 데이터로 분류

```
categorical_features = ['Channel', 'Region'] # 명목형 데이터
# 연속형 데이터
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']

for col in categorical_features:
    # 명목형 데이터는 판다스의 get_dummies()를 사용하여 숫자(0과 1)로 변환
    dummies = pd.get_dummies(data[col], prefix=col)
    data = pd.concat([data, dummies], axis=1)
    data.drop(col, axis=1, inplace=True)
data.head()
```

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

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
- 일정한 범위를 유지하도록 사이킷런의 MinMaxScaler() 메서드를 사용

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

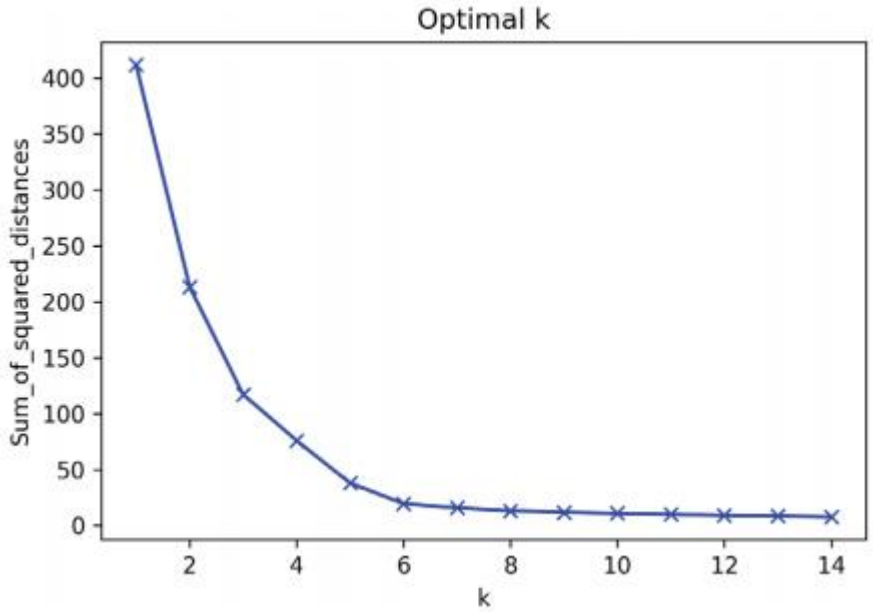
2. 비지도 학습

❖ K-평균 군집화

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

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

Sum_of_squared_distances = []
K = range(1,15)
for k in K:
    km = KMeans(n_clusters=k) # K에 1~14까지 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()
```

그림 34 코드를 실행결과 => K 값이 출력



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 수)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

2. 비지도 학습

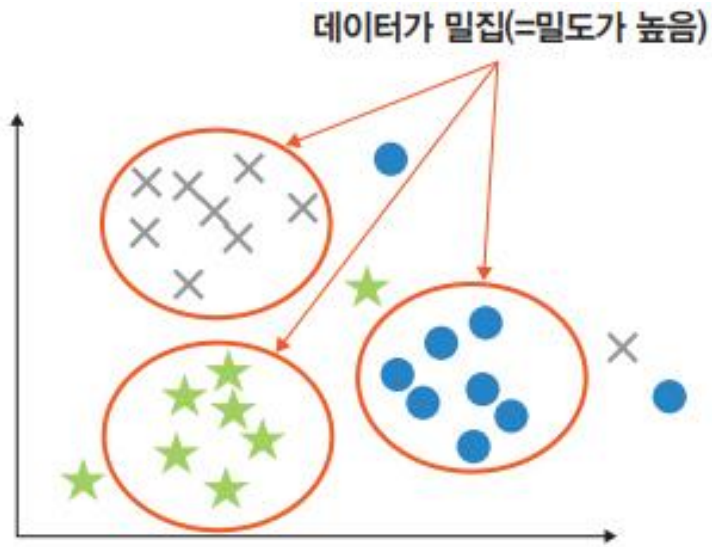
❖ 밀도 기반 군집 분석

▼ 표 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.

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

▼ 그림 35 밀도 기반 군집 분석의 밀집도

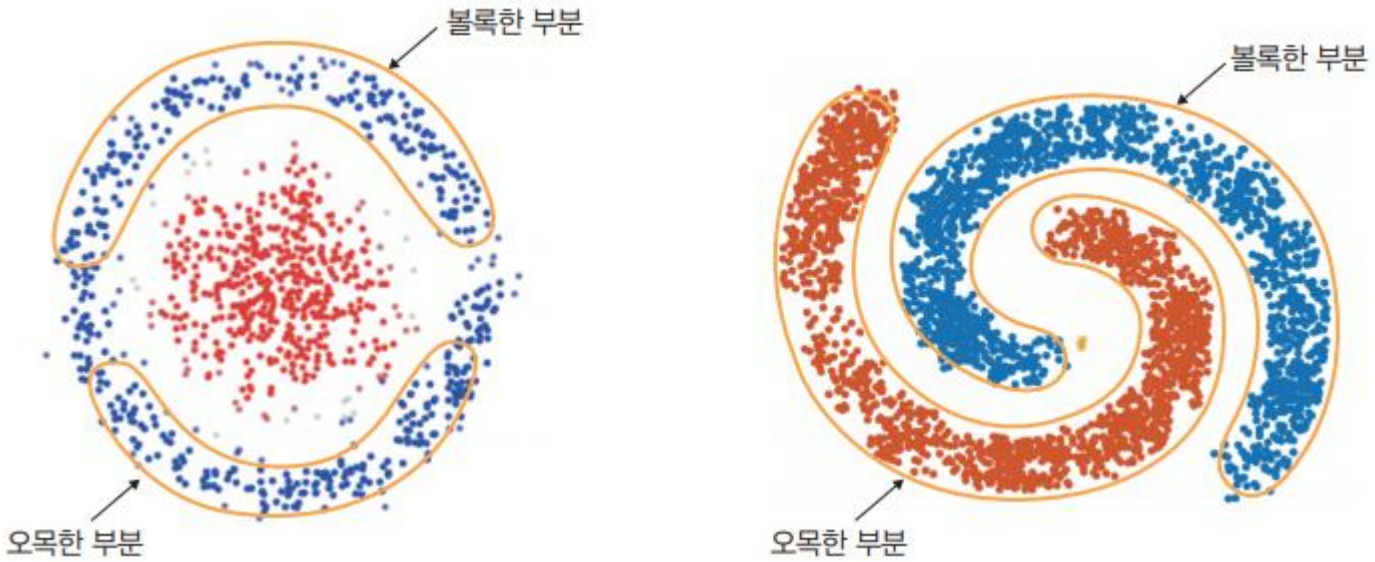


2. 비지도 학습

❖ 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



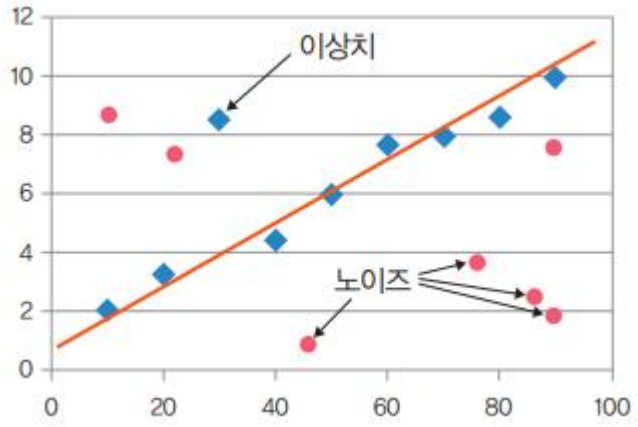
2. 비지도 학습

❖ 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 37 노이즈와 이상치



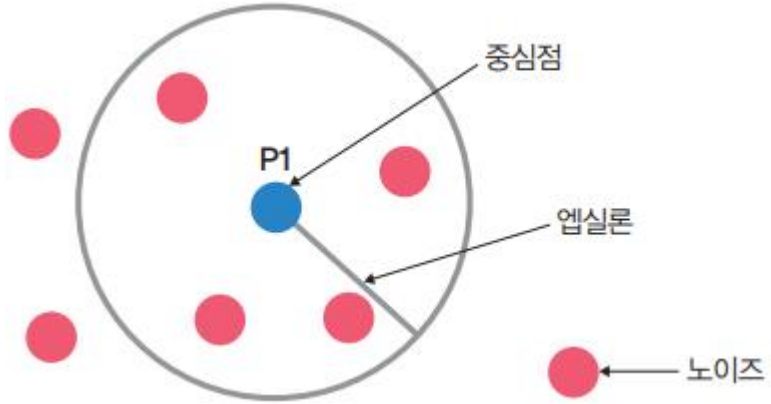
2. 비지도 학습

❖ 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 개 있기 때문에 하나의 군집이 되고, P1은 중심점이 됨

▼ 그림 38 중심점과 엡실론



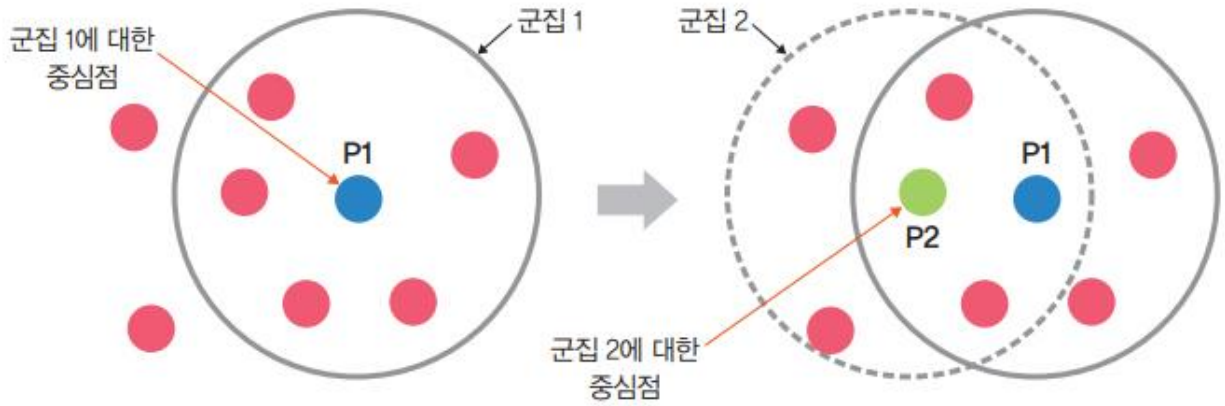
2. 비지도 학습

❖ 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 minPts=3을 만족하기 때문에 새로운 군집을 생성할 수 있음

▼ **그림 39 군집 확장**

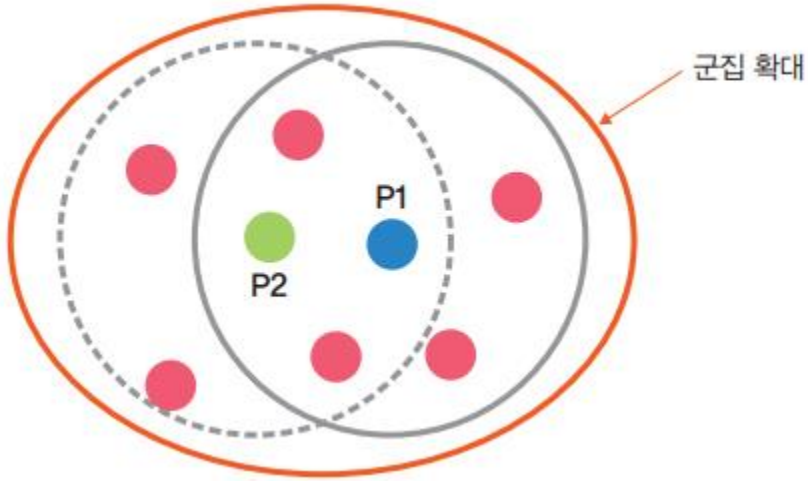


2. 비지도 학습

❖ 밀도 기반 군집 분석

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

▼ 그림 3-40 군집 확대



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

2. 비지도 학습

❖ 주성분 분석(PCA)

▼ 표 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.

- 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

2. 비지도 학습

❖ 주성분 분석(PCA)

- 차원 축소 방법은 다음과 같음

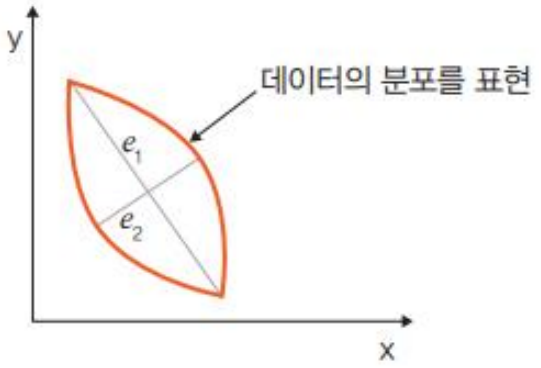
데이터들의 분포 특성을 잘 설명하는 벡터를 두 개 선택

- 다음 그림에서 e_1과 e_2 두 벡터는 데이터 분포를 잘 설명
- e_1의 방향과 크기, e_2의 방향과 크기를 알면 데이터 분포가 어떤 형태인지 알 수 있기 때문임

벡터 두 개를 위한 적정한 가중치를 찾을 때까지 학습을 진행

- 즉, PCA는 데이터 하나하나에 대한 성분을 분석하는 것이 아니라,
- 여러 데이터가 모여 하나의 분포를 이룰 때 이 분포의 주성분을 분석하는 방법

▼ 그림 41 2D에서 PCA 예시

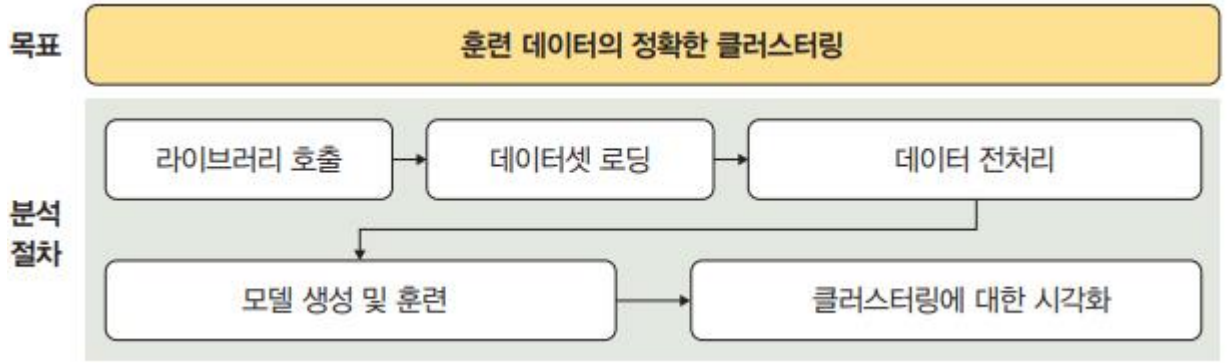


2. 비지도 학습

❖ 주성분 분석(PCA)

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

▼ 그림 3-42 밀도 기반 군집 분석과 PCA 예제



2. 비지도 학습

❖ 주성분 분석(PCA)

- 필요한 라이브러리를 호출 및 데이터 불러오기

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

from sklearn.cluster import DBSCAN  # 밀도기반 군집 분석
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
```

```
X = pd.read_csv('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
```

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	

	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	

2. 비지도 학습

❖ 주성분 분석(PCA)

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

▼ 그림 43 df.fillna() 메서드

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

	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

2. 비지도 학습

❖ 주성분 분석(PCA)

- 데이터 전처리 및 차원 축소를 진행

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 평균 :0, 표준편차:1이 되도록 데이터 크기를 조절
X_normalized = normalize(X_scaled)  # 데이터가 가우스 분포를 따르도록 정규화
X_normalized = pd.DataFrame(X_normalized)
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())
```

차원축소 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

2. 비지도 학습

❖ 주성분 분석(PCA)

- 훈련된 모델에 대해 시각화

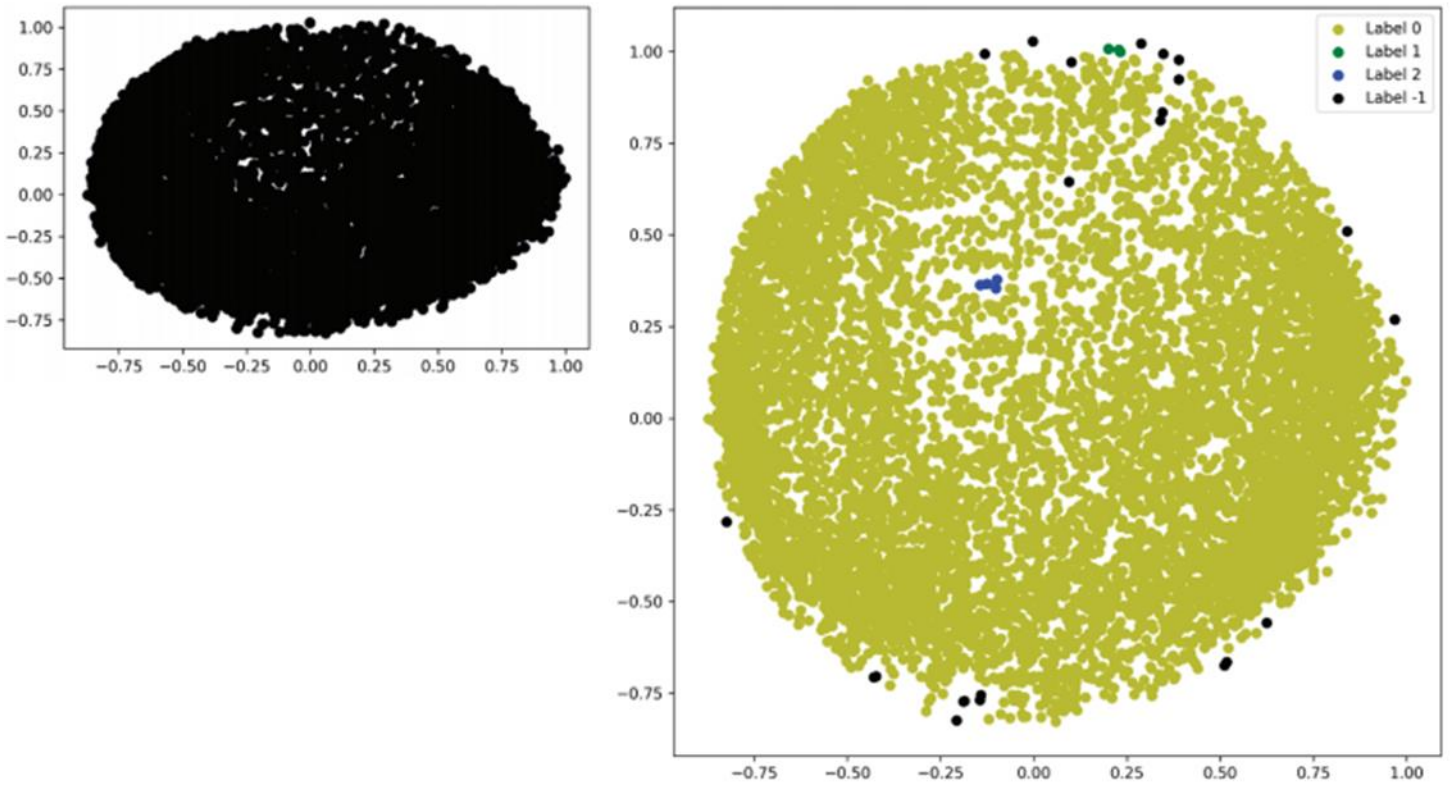
```
db = DBSCAN(eps = 0.0375, min_samples = 3).fit(X_principal)
labels1 = db.labels_

colours1 = {}
colours1[0] = 'r'
colours1[1] = 'g'
colours1[2] = 'b'
colours1[-1] = 'k'

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])
k = plt.scatter(X_principal['P1'], X_principal['P2'], marker ='o', color = colors1[6])

plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((r, g, b, k),
           ('Label 0', 'Label 1', 'Label 2', 'Label -1'))
plt.show()
```

2. 비지도 학습

❖ 주성분 분석(PCA)

- 다음 그림은 DBSCAN 모델을 실행하여 시각화한 결과

▼ 그림 44 밀도 기반 군집 분석과 PCA 예제 실행 결과



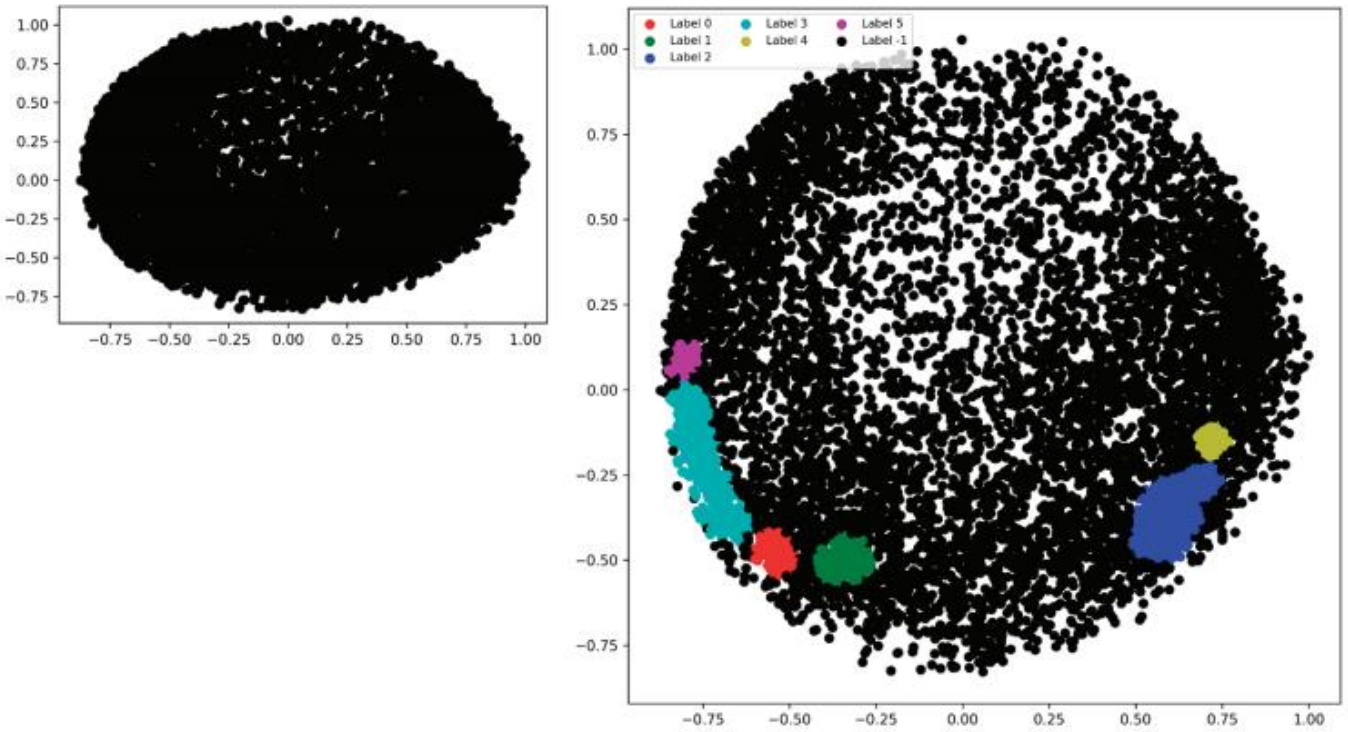
2. 비지도 학습

❖ 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 수정

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

▼ 그림 45 밀도 기반 군집 분석과 PCA 예제 튜닝 결과



2. 비지도 학습

❖ 주성분 분석(PCA)

```
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'

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])

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()
```

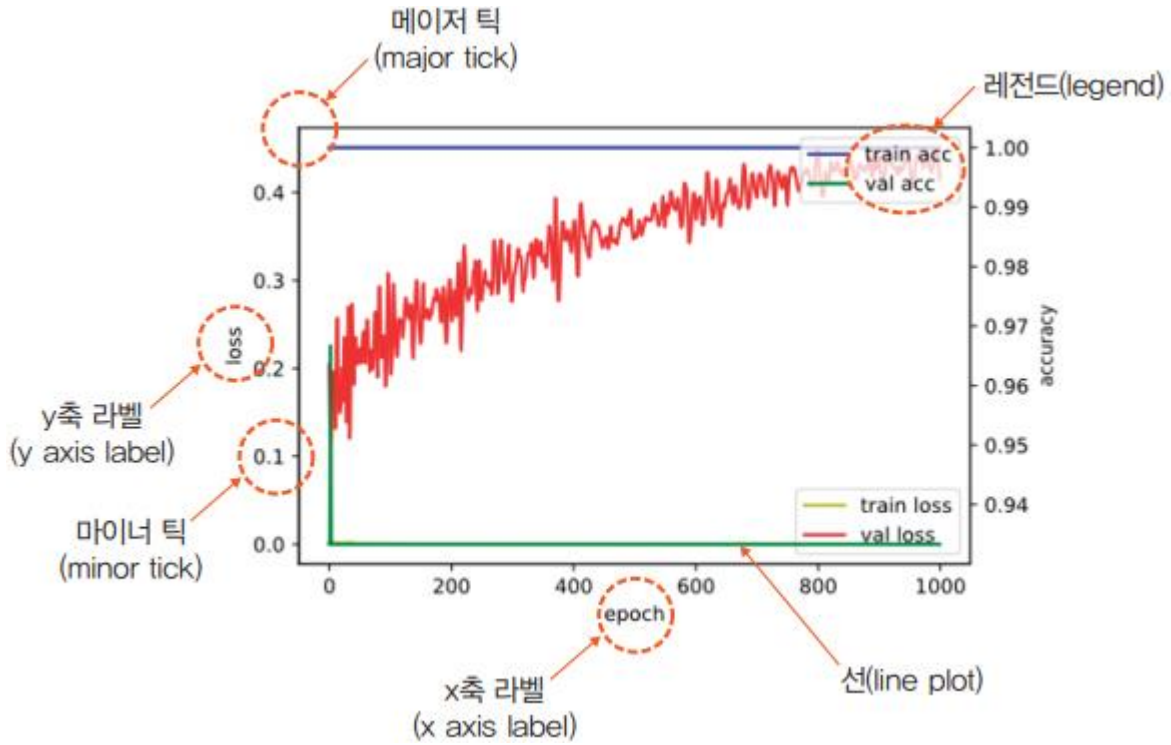
2.비지도 학습

❖ 주성분 분석(PCA)

맷플롯립 용어

- 다음 그림은 맷플롯립(matplotlib) 라이브러리로 출력하는 그림에서 사용되는 용어들을 정리한 것

▼ 그림 46 맷플롯립



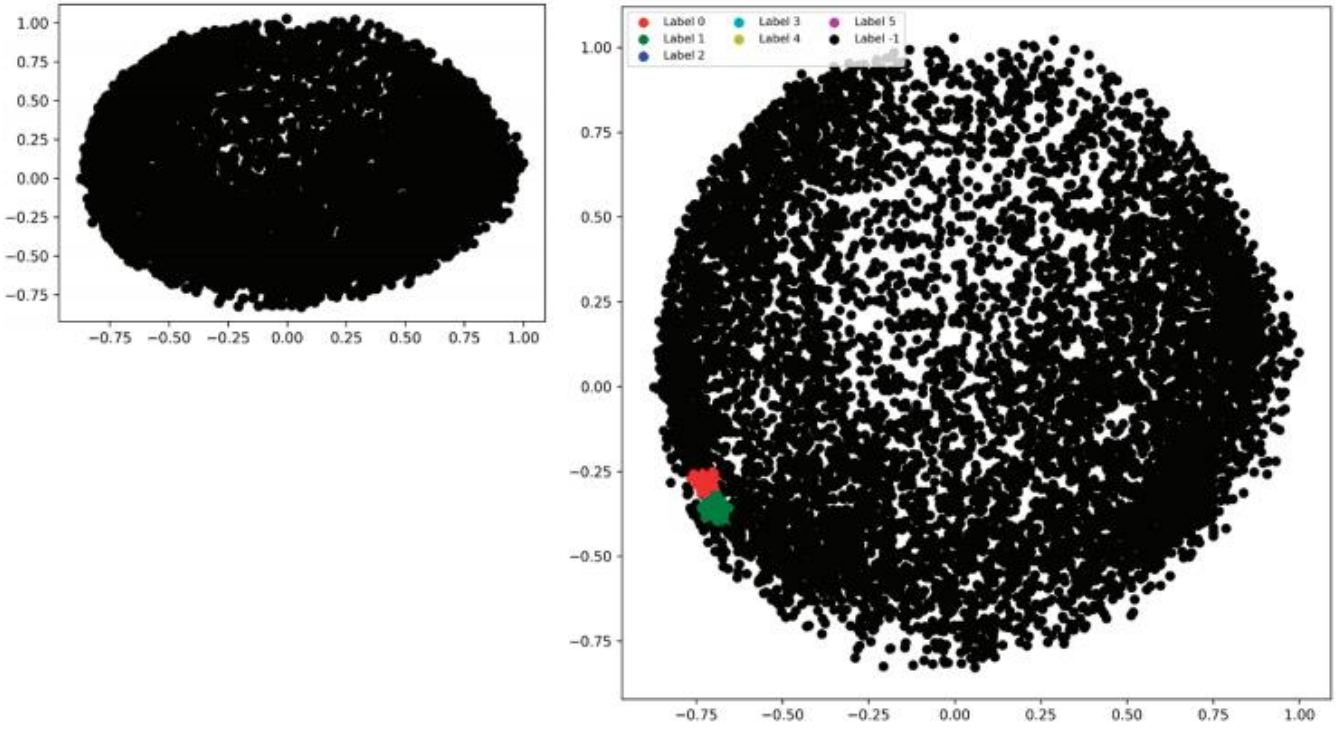
2. 비지도 학습

❖ 주성분 분석(PCA)

- 추가적으로 밀도 기반 군집 분석 모델의 하이퍼파라미터 인자 min_samples를 50에서 100으로 변경해 보면 그림 47과 같은 그래프를 출력

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

▼ 그림 47 밀도 기반 군집 분석과 PCA 예제에서 잘못된 하이퍼파라미터를 적용할 때의 결과



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 중요