AI Programming

데이터 마이닝 / 생성형 AI

05. 머신러닝 기초 - 비지도학습

first coding

first coding

비지도학습 소개

머신러닝 비지도학습의 정의와 활용 분야

지도학습과 비지도학습의 차이점

비지도학습의 주요 기법: 군집화와 차원 축소

머신러닝 비지도학습의 정의와 활용 분야

- **비지도학습**

  - 비지도학습은 정답(레이블)이 없는 데이터를 기반으로 학습하는 머신러닝의 한 방법
  - 입력 데이터에 포함된 숨겨진 패턴이나 데이터의 구조를 파악하는 데 초점을 맞춤

    → 즉, 모델이 데이터를 보고 스스로 규칙을 발견하거나 데이터를 특정 기준에 따라 그룹으로 나누는 것

    - 지도학습 : "이 데이터는 고양이야" 같은 정답이 주어진 상태에서 학습.
    - 비지도학습: "이 데이터들이 어떻게 나뉠 수 있을까?"를 스스로 학습.

머신러닝 비지도학습의 정의와 활용 분야

- **비지도학습의 활용 분야**

  - **군집화 (Clustering)** : 데이터들을 비슷한 특성을 가진 그룹으로 묶는 작업
    - 고객 세분화, 이미지 분류, 문서 분류
  - **차원 축소 (Dimensionality Reduction)** : 데이터를 더 간결하고 유용한 형태로 변환
    - 데이터 시각화, 노이즈 제거, 데이터 압축
  - **이상치 탐지 (Anomaly Detection)** : 정상적인 패턴에서 벗어나는 데이터(이상치)를 탐지
    - 신용카드 사기 탐지, 기계 고장 예측
  - **추천 시스템 (Recommendation System)** : 비슷한 데이터를 기반으로 사용자에게 추천을 제공
    - 영화 추천, 상품 추천

지도학습과 비지도학습의 차이점

- **지도학습과 비지도학습의 차이점**

  - 학습 데이터
    - 지도학습 : 데이터에 정답(레이블)이 포함되어 있음.
      - 예를 들어, 사진 데이터에 "고양이", "강아지"라는 라벨이 붙어 있음.
      - 목적 : 예측이 목적.
        입력 데이터와 정답 사이의 관계를 학습하여, 새 데이터를 예측.

    - 비지도학습 : **데이터에 레이블이 없음.**
      - 데이터만 제공되고, 모델이 데이터를 스스로 분석하고 **숨겨진 패턴을 발견.**
      - 목적 : 탐색이 목적.
        입력 데이터의 구조나 그룹화를 파악.

지도학습과 비지도학습의 차이점

- **지도학습과 비지도학습의 차이점**

  – 활용 관점에서 본 차이점

구분	지도학습	비지도학습
레이블 유무	정답(레이블)이 제공됨	레이블 없이 입력 데이터만 제공됨
데이터 구조 이해	데이터-정답 관계 학습	데이터 간의 패턴과 관계를 스스로 파악
결과물	새로운 데이터의 정답 예측	데이터의 그룹, 축소된 데이터 표현.
적용 사례	예측 모델 (스팸 이메일 분류, 가격 예측)	데이터 탐색 (고객 세분화, 시각화).

비지도학습의 주요 기법

- 비지도학습의 주요 기법

  - 군집화 (Clustering)
    - K-Means, Hierarchical Clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
  - 차원 축소 (Dimensionality Reduction)
    - PCA (Principal Component Analysis), t-SNE (t-Distributed Stochastic Neighbor Embedding), UMAP (Uniform Manifold Approximation and Projection)
  - 이상치 탐지 (Anomaly Detection)
    - Isolation Forest, LOF (Local Outlier Factor), Autoencoder

first coding

군집화 (Clustering) 기초

군집화의 개념과 활용 (고객 세분화, 이미지 세그멘테이션 등)

K-Means 알고리즘 원리와 적용

클러스터 수(k) 선택 방법

군집화의 개념과 활용

- 군집화(Clustering)
  - 군집화는 주어진 데이터에서 유사한 특성을 가진 데이터 포인트들을 그룹으로 묶는 것
    - 이때 각 그룹을 클러스터(Cluster)라고 함
    - 데이터를 사전에 분류하지 않고, 데이터를 살펴보며 그룹을 형성
  - 군집화의 주요 목표
    - 데이터 구조 파악 : 데이터를 시각화 하거나 그룹화하여 숨겨진 패턴을 발견
    - 유사성 기반 그룹화 : 데이터 포인트들 간의 유사성을 기준으로 분류하여 각 그룹이 내포하는 의미를 이해
  - 군집화의 작동 방식
    - 유사성 측정 : 데이터 포인트 간의 유사성을 측정하기 위해 거리 계산(예: 유클리드 거리)을 사용
    - 그룹화 : 유사한 데이터 포인트들을 같은 그룹으로 묶음

군집화의 개념과 활용

- **군집화(Clustering)의 활용**
  - 고객 세분화(Customer Segmentation)
    - 고객 데이터를 분석하여 유사한 구매 패턴, 관심사, 행동을 가진 그룹으로 나눔.
    - 효과 : 각 그룹에 맞는 마케팅 전략을 세우고, 고객 만족도를 높일 수 있음.
      - 자주 구매하는 고객 그룹(충성 고객)
      - 할인에 민감한 고객 그룹
      - 특정 상품에 집중하는 고객 그룹
  - 이미지 세그멘테이션(Image Segmentation)
    - 이미지의 픽셀을 비슷한 색상이나 질감을 가진 영역으로 나눔.
    - 효과 : 물체 탐지, 의료 영상 분석, 자율주행차의 객체 인식 등에 활용.
      - CT 스캔 이미지에서 장기와 조직 분리
      - 위성 이미지에서 건물, 숲, 강 구분

군집화의 개념과 활용

- **군집화(Clustering)의 활용**
  - 문서 분류 및 정보 검색
    - 문서를 주제별로 그룹화하여 검색 및 추천 시스템에 활용.
    - 효과 : 효율적인 데이터 관리와 검색 결과 개선.
      - 뉴스 기사 데이터를 스포츠, 정치, 기술 등으로 자동 분류.
  - 이상치 탐지(Anomaly Detection)
    - 대부분의 데이터와 크게 다른 이상 데이터를 감지.
    - 효과 : 금융 사기 탐지, 네트워크 보안 등에 활용.
      - 정상적인 거래와 사기 거래 구분네트워크에서 비정상적인 패킷 탐지
  - 추천 시스템(Recommendation System)
    - 유사한 관심사나 행동 패턴을 가진 사용자들을 그룹화하여 추천 항목을 생성
    - 효과 : 사용자 맞춤형 추천으로 사용자 만족도 증가
      - 음악 추천(스포티파이, 유튜브)상품 추천(아마존, 쿠팡)

K-Means 알고리즘 원리와 적용

- K-Means 알고리즘

  - K-Means는 데이터 군집화에서 가장 널리 사용되는 알고리즘 중 하나
  - 주어진 데이터를 K개의 클러스터로 나누는 비지도 학습 방법

    → 반복적이고 간단한 방식으로 데이터를 그룹화하며, 각 클러스터는 하나의 중심(centroid)으로 설정

  - 활용
    - **고객 세분화** : 고객 데이터를 K-Means로 클러스터링 하여, 비슷한 소비 행동을 가진 그룹으로 분류
      - 예: 충성 고객, 신규 고객, 할인 선호 고객 그룹화.
    - **이미지 압축** : K-Means로 픽셀 색상을 클러스터링 하여 주요 색상만 유지.
    - **문서 분류** : 텍스트 데이터를 클러스터링 하여 문서를 주제별로 분류.

K-Means 알고리즘 원리와 적용

- K-Means 알고리즘의 원리

① 초기 클러스터 중심(Centroid) 설정

- 클러스터의 개수 K를 사용자가 미리 설정, 초기 클러스터 중심은 무작위로 설정하거나 데이터를 기반으로 선택

② 데이터 포인트 할당

- 각 데이터 포인트를 가장 가까운 클러스터 중심에 할당 → 가까운 중심에 속하는 데이터 포인트들이 하나의 클러스터를 형성
- 데이터 포인트와 클러스터 중심 사이의 거리는 일반적으로 유클리드 거리(Euclidean Distance)를 사용하여 계산

③ 클러스터 중심 재계산

- 각 클러스터에 속한 데이터 포인트들의 평균 위치를 계산하여 새로운 클러스터 중심으로 설정

$$\text{새로운 중심} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

여기서 xi 는 클러스터 내 데이터 포인트, n은 데이터 포인트의 개수

④ 반복

- 데이터 포인트를 다시 가장 가까운 클러스터 중심에 할당하고, 클러스터 중심을 재계산
- 클러스터 중심이 더 이상 변하지 않거나(수렴), 최대 반복 횟수에 도달하면 알고리즘이 종료

K-Means 알고리즘 원리와 적용

- K-Means 알고리즘의 적용

① 데이터 준비

⬇

- 군집화를 수행할 데이터 준비 및 정규화(데이터의 크기가 클 경우 정규화가 중요).

② K 값 선택

⬇

- 원하는 클러스터 개수 K를 결정.

③ 알고리즘 실행

⬇

- K-Means 알고리즘을 적용하여 데이터 클러스터링.

④ 결과 평가 및 시각화

- 각 클러스터의 품질을 평가하고, 시각적으로 확인.

클러스터 수(k) 선택 방법

- K 값 선택 방법
  - 엘보우 방법(Elbow Method)를 통해 최적의 K 값을 선택
    - 여러 K 값에 대해 K-Means를 실행하고,
      각 경우의 클러스터 내 데이터의 거리 합(WCSS, Within-Cluster Sum of Squares)을 계산
    - K 값에 따른 WCSS를 그래프로 그려, 그래프의 기울기가 급격히 완화되는 지점(엘보우 포인트)에서 최적의 K를 선택.

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

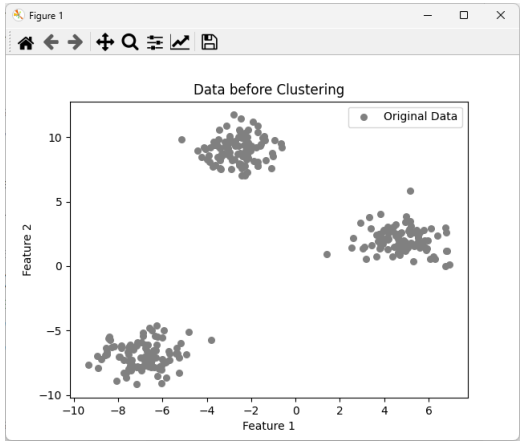
- C_i: 클러스터 i
- μ_i: 클러스터 중심
- x: 데이터 포인트

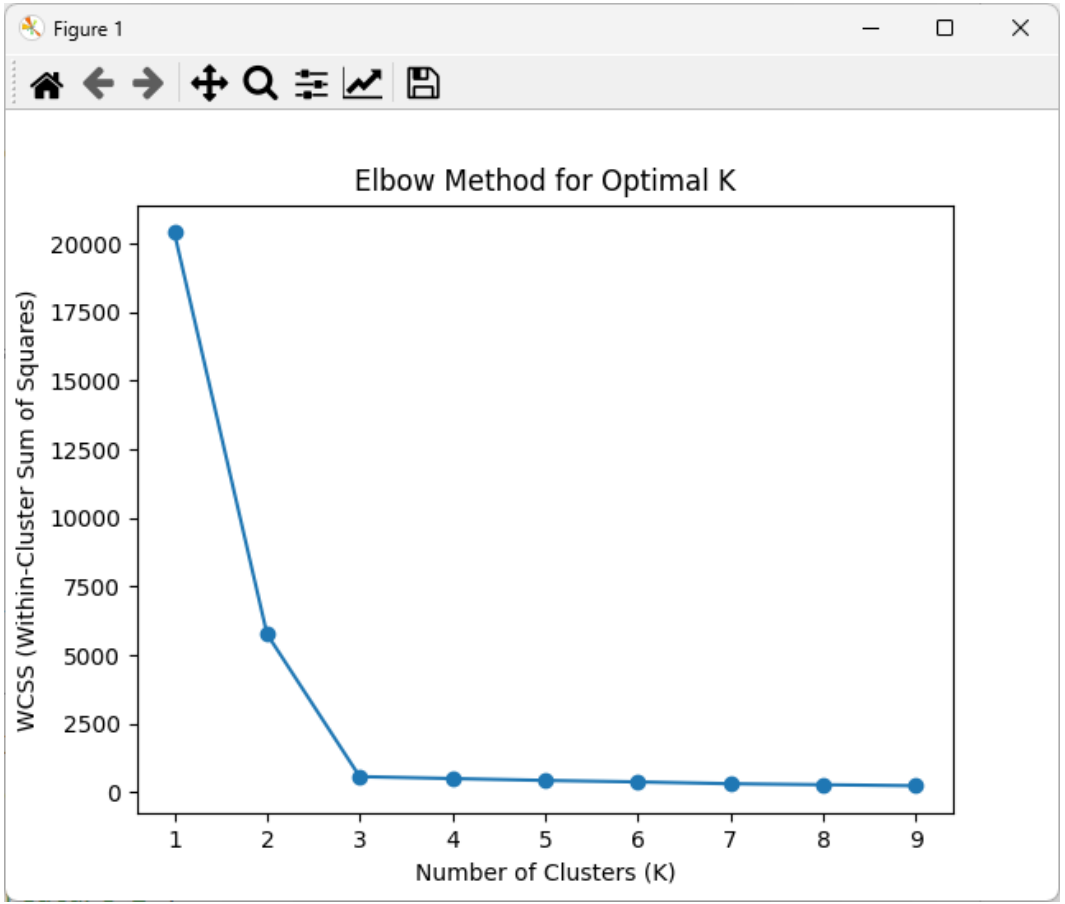
K-Means 알고리즘 예제

```python
# 필요한 라이브러리 import
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# 1. 데이터 생성
# make_blobs: 군집화에 적합한 샘플 데이터를 생성하는 함수
# n_samples: 데이터 포인트 수, centers: 클러스터 수, random_state: 재현성을 위한 시드값
n_samples = 300
n_clusters = 3
X, y = make_blobs(n_samples=n_samples, centers=n_clusters, cluster_std=1.0, random_state=42)

# 2. 데이터 시각화 (군집화 전)
plt.scatter(X[:, 0], X[:, 1], s=30, c='gray', label="Original Data")
plt.title("Data before Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# 3. K-Means 모델 생성 및 학습
# KMeans 클래스: 사이킷런에서 제공하는 K-Means 구현
# n_clusters: 클러스터 개수, random_state: 결과 재현성을 위한 시드값
kmeans = KMeans(n_clusters=n_clusters, random_state=42)

# fit_predict: 데이터를 클러스터링하고 각 데이터 포인트의 클러스터 레이블을 반환
cluster_labels = kmeans.fit_predict(X)
```

```python
# 4. 군집화 결과 시각화
# 각 데이터 포인트의 클러스터에 따라 색을 지정
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis', s=30, label="Clustered Data")
# 클러스터 중심 시각화
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', label="Centroids", marker='X')
plt.title("Data after Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# 5. 모델 세부 정보 확인
# kmeans.cluster_centers_: 클러스터 중심 좌표
# kmeans.inertia_: 클러스터 내 거리 합 (WCSS)
print("Cluster Centers:\n", kmeans.cluster_centers_)
print("Inertia (WCSS):", kmeans.inertia_)

# 6. 최적의 K 값을 찾기 위한 엘보우 방법 (추가 분석)
# 다양한 K 값에 대한 Inertia 계산
wcss = []
K_range = range(1, 10)
for k in K_range:
    kmeans_temp = KMeans(n_clusters=k, random_state=42)
    kmeans_temp.fit(X)
    wcss.append(kmeans_temp.inertia_)

# 엘보우 그래프 그리기
plt.plot(K_range, wcss, marker='o')
plt.title("Elbow Method for Optimal K")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS (Within-Cluster Sum of Squares)")
plt.show()
```

K-Means 알고리즘 예제





first coding

군집화 응용 및 성능 평가

클러스터링 결과의 해석

클러스터 품질 평가: 실루엣 점수(Silhouette Score)

K-Means 외의 군집화 알고리즘 소개 (Hierarchical Clustering, DBSCAN)

클러스터링 결과의 해석

- **클러스터링 결과를 시각화**

  - 클러스터링 결과를 해석하는 첫 단계는 데이터를 시각적으로 분석하는 것
  - 2D 또는 3D 플롯데이터가 2차원이나 3차원으로 변환 가능하다면 각 클러스터를 색으로 구분하여 시각화
    - 주로 사용하는 도구
      - **Matplotlib** : 산점도(scatter plot)로 표현
      - **Seaborn** : 페어 플롯(pair plot) 등
      - **PCA** : 고차원 데이터를 2차원으로 축소하여 클러스터를 시각화
  - 클러스터 중심 확인
    - 각 클러스터의 중심(centroid)을 확인
    - 중심점은 클러스터 내 데이터를 대표하는 점으로, 클러스터의 특성을 요약

클러스터링 결과의 해석

- **클러스터의 특징 이해**

  - 클러스터 내 데이터의 평균, 분포 확인
    - 클러스터에 속한 데이터의 주요 속성(특징)의 평균값과 분포를 분석
      예: 고객 세그먼트를 분류했다면, 각 클러스터의 평균 구매 금액, 구매 빈도 등을 비교
    - 이를 통해 각 클러스터가 어떤 특성을 가지고 있는지 이해할 수 있음

  - 클러스터 간의 차이점 파악
    - 클러스터를 서로 비교하여 공통점과 차이점을 확인
      예: 하나의 클러스터는 구매 빈도가 높고 금액이 낮은 고객, 다른 클러스터는 구매 빈도는 낮지만 금액이 높은 고객일 수 있음

클러스터링 결과의 해석

- 클러스터의 적절성 평가

  - **데이터 응집도**
    - 클러스터 내의 데이터가 얼마나 가까이 모여 있는지를 분석

      → 가까이 모여 있을수록 해당 클러스터는 응집도가 높다고 평가

  - **데이터 분리도**
    - 다른 클러스터와 얼마나 잘 분리되어 있는지를 분석

      → 클러스터 간의 거리가 멀수록 분리도가 높다고 평가

클러스터링 결과의 해석

- 실질적인 의미 부여

  - **도메인 지식을 활용한 해석**
    - 클러스터링은 자동으로 클러스터를 나누지만, 해석은 도메인 지식이 필요

      예: 의료 데이터에서 클러스터링을 했다면, 각 클러스터가 특정 질병에 걸린 환자 집단일 수 있음

  - **이름 붙이기**
    - 각 클러스터에 적절한 이름을 붙여 줌

      예: 고객 데이터 클러스터링에서 "고액 구매 고객", "저빈도 구매 고객" 등으로 명명

클러스터 품질 평가

- **클러스터 품질 평가**

  - 클러스터 품질 평가는 클러스터링 결과가 얼마나 잘 나왔는지 평가하는 과정
    K-Means 알고리즘에서 클러스터 품질을 평가하려면
    → 클러스터 내 데이터 응집도와 클러스터 간 분리도를 확인해야 함

  - **실루엣 점수(Silhouette Score)**
    - 는 클러스터의 응집도와 분리도를 동시에 평가할 수 있는 강력한 지표
    - K-Means의 품질을 판단하는 데 유용

클러스터 품질 평가

- **클러스터 품질 평가**

  - 실루엣 점수(Silhouette Score)
    - 실루엣 점수는 데이터 포인트가 자신의 클러스터와 얼마나 잘 응집되어 있고, 다른 클러스터와 얼마나 잘 분리되어 있는지를 나타냄
    - 계산 공식 (각 데이터 포인트 i에 대해)

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

      - $a(i)$: 데이터 포인트 i와 같은 클러스터 내 다른 포인트 간의 평균 거리 (응집도)
      - $b(i)$: 데이터 포인트 i와 가장 가까운 다른 클러스터 포인트들 간의 평균 거리 (분리도)
      - $s(i)$의 값 범위 : $-1 \leq s(i) \leq 1$

클러스터 품질 평가

- **클러스터 품질 평가**

  - 실루엣 점수(Silhouette Score)의 의미 $$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$
    - $s(i)$ 값 해석
      - **$s(i)$가 1에 수렴** : 데이터 포인트가 자신의 클러스터에 잘 속해 있고, 다른 클러스터와 명확히 분리됨
      - **$s(i)$가 0에 수렴** : 데이터 포인트가 경계에 가까워져, 어느 클러스터에 속했 어야 하는지 불확실
      - **$s(i)$가 음수** : 데이터 포인트가 잘못된 클러스터에 속했을 가능성이 높음.
    - 전체 실루엣 점수
      - 데이터셋 전체의 실루엣 점수는 모든 데이터 포인트의 $s(i)$ 값 평균으로 계산됨

$$S = \frac{1}{n} \sum_{i=1}^{n} s(i)$$

S 가 1에 가까울수록 클러스터링이 잘 되었다고 평가.

- **클러스터 품질 평가**

  - 실루엣 점수를 활용한 K 값 선택
    - K-Means의 가장 큰 과제는 적절한 클러스터 수 k를 선택하는 것

      → 실루엣 점수는 이를 돕는 데 매우 유용함
    - 방법

      ① 여러 k값에 대해 K-Means를 실행

      ② 각 k값에 대해 실루엣 점수를 계산

      ③ 실루엣 점수 시각화

      ④ 실루엣 점수가 가장 높은 k값을 선택

클러스터 품질 평가 예제

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score, silhouette_samples

# 1. 데이터 생성
# - make_blobs를 사용하여 군집화에 적합한 샘플 데이터를 생성합니다.
# - centers: 생성할 클러스터의 수
# - n_samples: 생성할 데이터 포인트 수
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.6,
random_state=42)

# 데이터 시각화 (원본 데이터)
plt.scatter(data[:, 0], data[:, 1], s=30, color='gray', label="Original
Data")
plt.title("Generated Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# 2. K-Means 알고리즘 적용
# - n_clusters: 생성할 클러스터 수
# - random_state: 결과 재현성을 위한 랜덤 시드
kmeans = KMeans(n_clusters=3, random_state=42)

# K-Means 모델 학습 및 클러스터 예측
# - fit_predict: 데이터를 학습시키고 각 데이터 포인트의 클러스터 할당을 반환
cluster_labels = kmeans.fit_predict(data)
```

```
# 클러스터링 결과 시각화
plt.scatter(data[:, 0], data[:, 1], c=cluster_labels, cmap='viridis', s=30)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroids')  # 클러스터 중심
plt.title("K-Means Clustering Results")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# 3. 클러스터 품질 평가 - 실루엣 점수 계산
# - silhouette_score: 전체 데이터에 대한 평균 실루엣 점수를 계산
silhouette_avg = silhouette_score(data, cluster_labels)
print(f"Overall Silhouette Score: {silhouette_avg:.2f}")

# 4. 실루엣 점수 시각화 - 각 데이터 포인트의 실루엣 점수를 계산
silhouette_values = silhouette_samples(data, cluster_labels)

# 시각화 준비
y_lower = 10
for i in range(4):  # 각 클러스터에 대해 반복
    ith_cluster_silhouette_values = silhouette_values[cluster_labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    # 클러스터별 막대 그리기
    plt.fill_betweenx(np.arange(y_lower, y_upper), 0,
ith_cluster_silhouette_values)
    plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))  # 클러스터 번호
    y_lower = y_upper + 10  # 다음 클러스터로 이동

# 그래프 설정
plt.axvline(x=silhouette_avg, color="red", linestyle="--")  # 평균 실루엣 점수
plt.title("Silhouette Plot for K-Means Clustering")
plt.xlabel("Silhouette Score")
plt.ylabel("Cluster")
plt.show()

# 5. 결과 요약 - 실루엣 점수와 클러스터 중심 출력
print(f"Cluster Centers:\n{kmeans.cluster_centers_}")
print(f"Silhouette Score for each point stored as silhouette_values array.")
```

클러스터 품질 평가 예제

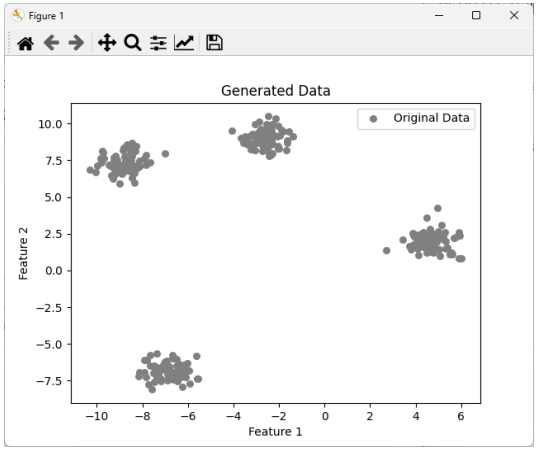


Figure 1
Generated Data
Original Data
Feature 1
Feature 2

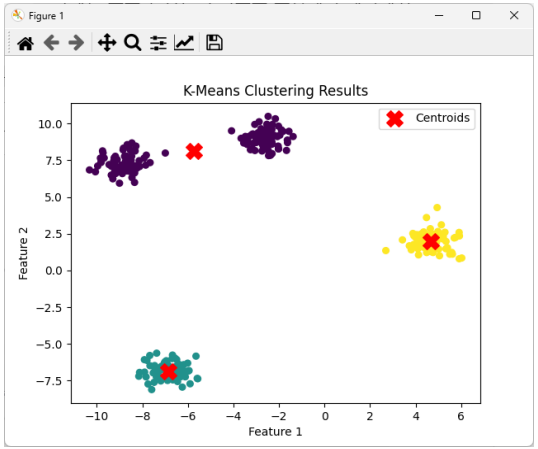


Figure 1
K-Means Clustering Results
Centroids
Feature 1
Feature 2

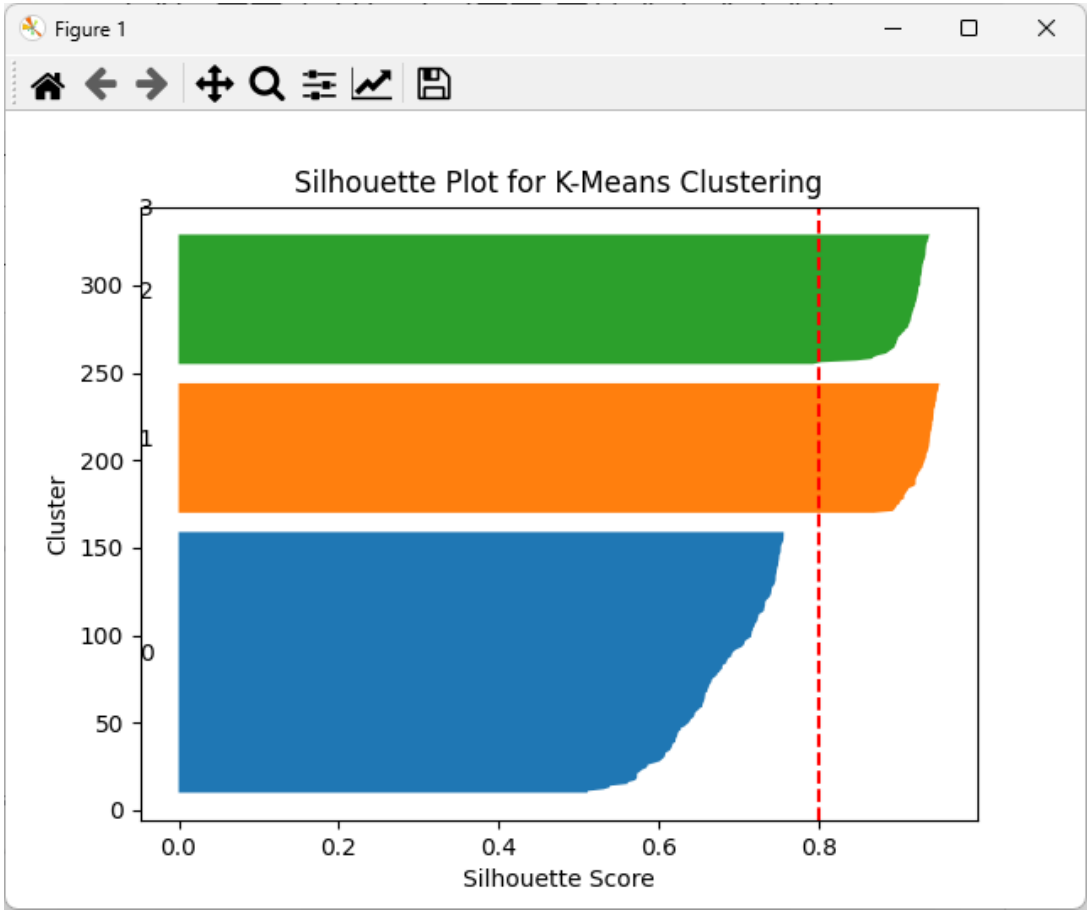


Figure 1
Silhouette Plot for K-Means Clustering
Cluster
Silhouette Score

K-Means 외의 군집화 알고리즘 소개

- **Hierarchical Clustering (계층적 군집화)**

  - 데이터 포인트 간의 계층적 관계를 기반으로 클러스터를 형성
    - 데이터가 트리 형태로 그룹화
    - 결과는 덴드로그램(Dendrogram)이라는 시각화로 표현
  - 두 가지 방법
    - 병합형(Agglomerative)
      - 각 데이터 포인트를 개별 클러스터로 시작하여 가까운 클러스터를 병합.
    - 분할형(Divisive)
      - 모든 데이터를 하나의 클러스터로 시작하여 점차 분리.

K-Means 외의 군집화 알고리즘 소개

- Hierarchical Clustering (계층적 군집화)

  - 특징
    - 장점
      - 클러스터 수(k)를 미리 지정할 필요 없음
      - 덴드로그램을 통해 클러스터 구조를 시각적으로 확인 가능
    - 단점
      - 큰 데이터셋에서는 계산 비용이 높음
      - 클러스터 병합 또는 분할이 초기 단계에서 잘못되면 이후 결과에 영향을 줄 수 있음.

K-Means 외의 군집화 알고리즘 소개

- Hierarchical Clustering (계층적 군집화)

  - 작동 방식 (병합형 기준)
    ① 각 데이터 포인트를 하나의 클러스터로 간주
    ② 두 클러스터 간의 거리(또는 유사도)를 계산
    ③ 가장 가까운 두 클러스터를 병합
    ④ 클러스터가 하나가 될 때까지 반복

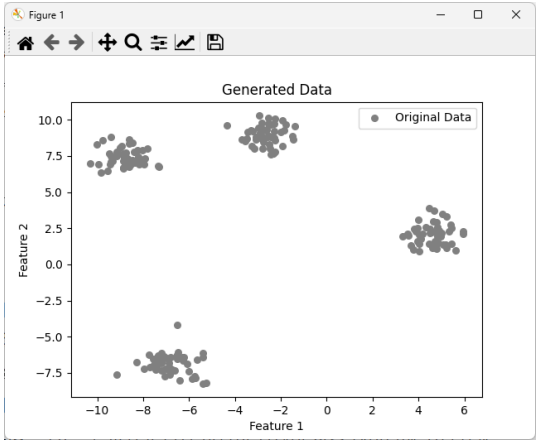
  - 클러스터 간 거리 계산 방법
    ① 최단 연결법 (Single Linkage) : 두 클러스터 간 가장 가까운 두 점 사이의 거리
    ② 최장 연결법 (Complete Linkage) : 두 클러스터 간 가장 먼 두 점 사이의 거리
    ③ 평균 연결법 (Average Linkage) : 두 클러스터 간 모든 점의 평균 거리.

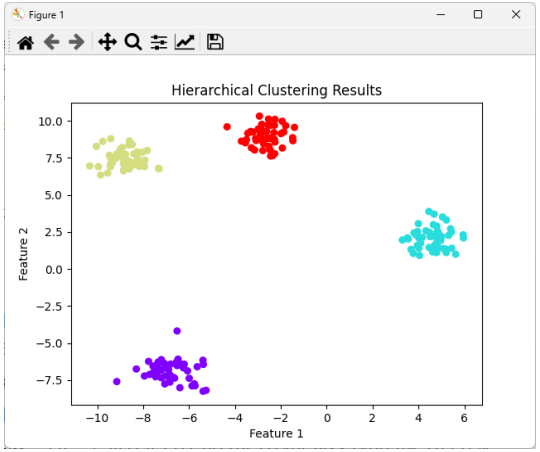
Hierarchical Clustering (계층적 군집화) 예제

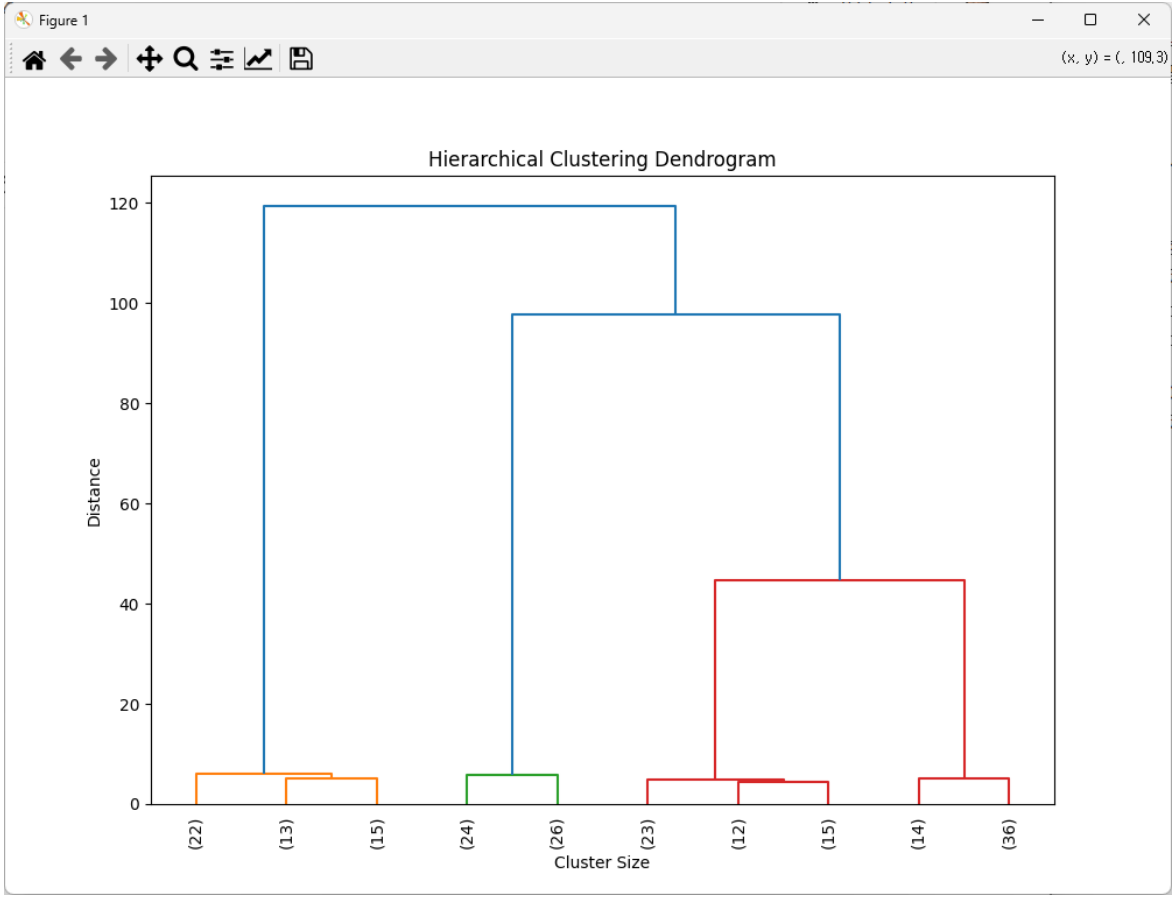
```
# 필요한 라이브러리 임포트
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib.pyplot as plt

# 1. 데이터 생성
# - 군집화 실험에 적합한 샘플 데이터를 생성합니다.
# - make_blobs: 클러스터가 명확히 구분되는 데이터셋 생성
data, _ = make_blobs(n_samples=200, centers=4, cluster_std=0.7,
random_state=42)

# 데이터 시각화 (원본 데이터)
plt.scatter(data[:, 0], data[:, 1], s=30, color='gray', label="Original
Data")
plt.title("Generated Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# 2. 계층적 군집화 적용
# - linkage 함수를 사용하여 클러스터 간의 거리 계산
# - method='ward': Ward's method로 데이터 간 거리 기반 병합
linked = linkage(data, method='ward')
```

```
# 3. 덴드로그램 시각화
# - linkage 결과를 기반으로 덴드로그램을 생성합니다.
plt.figure(figsize=(10, 7))
dendrogram(linked, truncate_mode='lastp', p=10, leaf_rotation=90,
leaf_font_size=10)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Cluster Size")
plt.ylabel("Distance")
plt.show()

# 4. 클러스터 형성
# - fcluster: 덴드로그램의 특정 거리 기준으로 클러스터를 형성
# - t=7: 클러스터를 나누는 임계값 (거리 기준)
cluster_labels = fcluster(linked, t=7, criterion='distance')

# 5. 결과 시각화
# - 각 클러스터를 색상으로 구분하여 시각화
plt.scatter(data[:, 0], data[:, 1], c=cluster_labels, cmap='rainbow', s=30)
plt.title("Hierarchical Clustering Results")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Hierarchical Clustering (계층적 군집화) 예제







K-Means 외의 군집화 알고리즘 소개

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

  - 밀도를 기반으로 데이터 포인트를 클러스터링 하는 알고리즘
    - 데이터가 밀집된 영역을 클러스터로 정의
    - 밀도가 낮은 데이터는 노이즈로 간주 → 이상치(노이즈)를 처리 가능
    - 클러스터 수(k)를 미리 지정할 필요 없음
    - 밀도가 다른 데이터나 비구형 데이터를 효과적으로 처리하며, 이상치 검출에 강점이 있음.
    - 고차원 데이터에서는 거리 계산이 왜곡될 수 있음

  - 주요 매개변수
    - Epsilon (ϵ) : 포인트가 밀집되었다고 판단할 거리의 임계값
    - MinPts : 한 포인트가 핵심 포인트가 되기 위한 최소 이웃 데이터 포인트 수.

K-Means 외의 군집화 알고리즘 소개

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

  - **동작 방식**

    ① 각 데이터 포인트에 대해 ϵ-이웃을 계산

    ② ϵ-이웃 내의 데이터 포인트 수가 MinPts 이상이면 해당 포인트는 **핵심(Core)** 포인트로 간주

    ③ 핵심 포인트와 연결된 이웃 데이터는 같은 클러스터로 그룹화

    ④ 밀도가 낮은 포인트는 노이즈로 분류.

DBSCAN 예제

```
# 필요한 라이브러리 임포트
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

# 1. 데이터 생성
# - make_moons: 비구형(non-linear) 데이터 구조를 가진 샘플 데이터를 생성
# - noise=0.05: 데이터에 노이즈 추가
data, _ = make_moons(n_samples=300, noise=0.05, random_state=42)

# 데이터 시각화
plt.scatter(data[:, 0], data[:, 1], s=30, color='gray', label="Original
Data")
plt.title("Generated Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# 2. DBSCAN 모델 생성 및 학습
# - eps: 포인트 간 거리가 이 값 이하일 때 이웃으로 간주 (밀도의 기준)
# - min_samples: 핵심 포인트가 되기 위한 최소 이웃 데이터 수
dbscan = DBSCAN(eps=0.2, min_samples=5)
```

```
# 데이터에 대한 클러스터 예측
cluster_labels = dbscan.fit_predict(data)

# 3. 클러스터링 결과 분석
# 클러스터링 결과 시각화
plt.scatter(data[:, 0], data[:, 1], c=cluster_labels, cmap='viridis', s=30)
plt.title("DBSCAN Clustering Results")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.colorbar(label="Cluster Label")
plt.show()

# 4. 클러스터링 결과 요약
# - 클러스터 레이블: 클러스터 번호 (노이즈는 -1로 표시)
unique_labels = set(cluster_labels)
print(f"Unique Clusters: {unique_labels}")

# 클러스터별 데이터 포인트 수 출력
for label in unique_labels:
    count = (cluster_labels == label).sum()
    if label == -1:
        print(f"Noise points: {count}")
    else:
        print(f"Cluster {label}: {count} points")
```

DBSCAN 예제

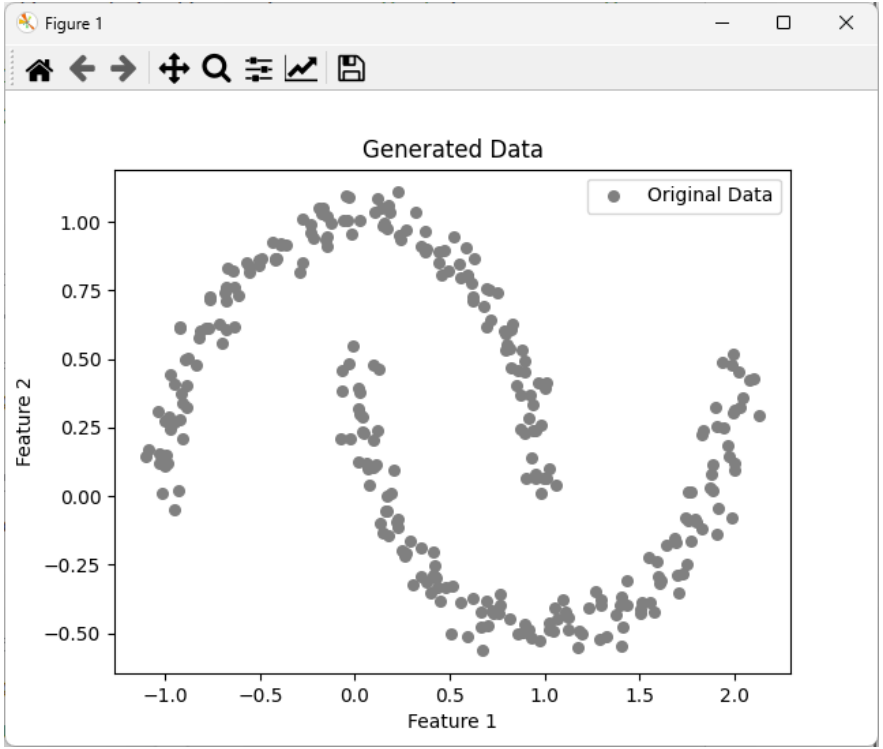


Figure 1
Generated Data
Original Data
Feature 2
Feature 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

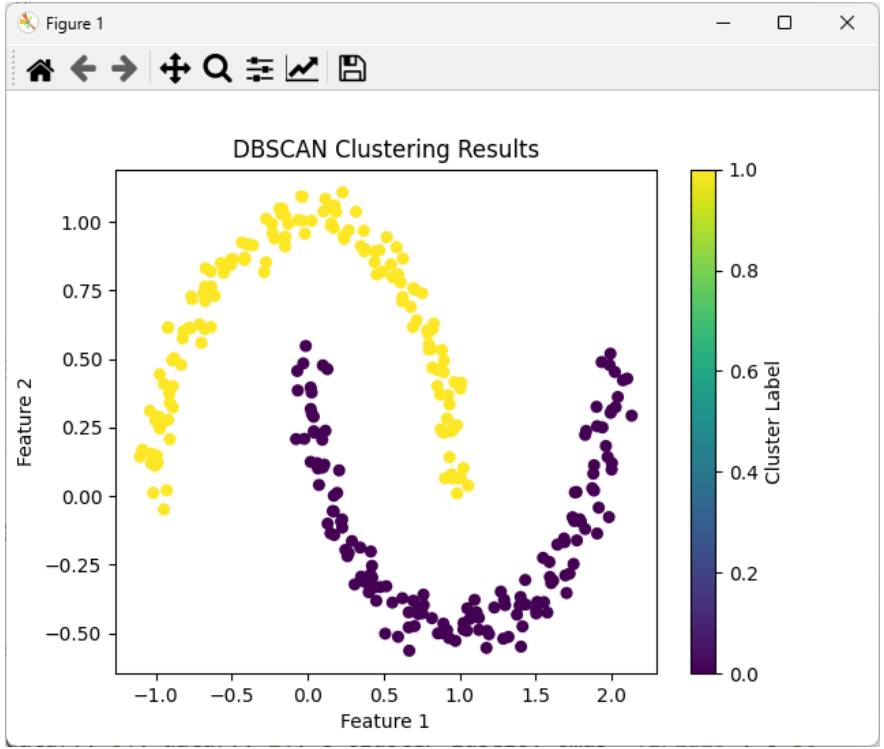


Figure 1
DBSCAN Clustering Results
Feature 2
Feature 1
Cluster Label
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
1.0
0.8
0.6
0.4
0.2
0.0

first coding

차원 축소 (Dimensionality Reduction) 기초

차원 축소의 필요성 (고차원의 문제, 시각화)

PCA(Principal Component Analysis) 개념 및 원리

주성분의 의미와 데이터 투영

차원 축소의 필요성

- **차원 축소(Dimensionality Reduction)**

  – 데이터가 고차원일수록(즉, 피처(feature)의 개수가 많을수록) 모델을 학습시키거나 분석할 때 여러 가지 문제가 발생할 수 있음 → 이 **문제들을 해결하기 위해 차원 축소가 필요**

① **고차원의 문제 (Curse of Dimensionality)**

- 데이터의 차원이 증가하면 데이터가 차지하는 공간이 기하급수적으로 커짐 → 이로 인해 발생하는 문제를 "차원의 저주"라 함
- **데이터가 희소해짐**

  고차원 공간에서는 데이터 포인트들이 서로 멀리 떨어져 있고, 유사성을 찾기가 어려워 짐
- **학습 속도 저하**

  고차원 데이터는 계산 비용이 증가하여 모델 학습이 느려짐
- **과적합(Overfitting)**

  피처가 너무 많으면 모델이 학습 데이터에 지나치게 맞춰져 일반화 능력이 떨어질 수 있음

차원 축소의 필요성

- 차원 축소(Dimensionality Reduction)

② 데이터 시각화

- 고차원의 데이터는 직관적으로 이해하거나 시각화하기 어려움 → 10차원 이상의 데이터를 사람이 시각적으로 이해 ?
- 차원 축소를 통해 데이터를 2차원 또는 3차원으로 변환 → 그래프를 통해 데이터의 분포나 패턴을 쉽게 분석할 수 있음

③ 노이즈 제거 : 이는 학습 모델의 성능 향상과 안정성을 제공

- 고차원 데이터에는 중요한 정보 뿐만 아니라 불필요한 정보(노이즈)가 포함될 수 있음
- 차원 축소는 데이터의 가장 중요한 특징(Variance, 변동성)을 유지하면서 노이즈를 줄여 줌

④ 모델 효율성 향상

- 데이터 차원을 줄이면 계산량이 줄어들어 모델 학습과 예측 속도가 빨라지고, 메모리 사용량도 감소
- 차원이 줄어들면 중요하지 않은 피처가 제거되므로 모델이 학습에만 필요한 정보에 집중할 수 있음

차원 축소의 필요성

- 차원 축소(Dimensionality Reduction)의 필요성 사례

  - 이미지 데이터
    - 이미지 한 장은 수천, 수만 개의 픽셀로 구성
    - 모든 픽셀을 사용하면 계산량이 너무 많아지므로, 중요한 정보만 추출하여 차원을 줄여야 함

  - 유전자 데이터
    - 유전자 분석 데이터는 수천 개의 피처로 구성되지만, 실제로 중요한 것은 몇 가지 피처에 불과함

  - 추천 시스템
    - 수천 명의 사용자가 수천 개의 상품에 대해 평가를 매겼다고 가정, 이 데이터를 모두 다루는 것은 비효율적
      → 중요한 사용 패턴만 추출하여 차원을 줄일 필요가 있음

PCA(Principal Component Analysis) 개념 및 원리

- PCA(Principal Component Analysis, 주성분 분석)

  – 고차원 데이터를 저차원으로 축소하면서도 데이터의 중요한 정보를 최대한 보존하는 차원 축소 기법
  → 데이터의 주요 특징을 파악하고, 노이즈를 제거하거나 시각화를 용이하게 함

  – PCA는 데이터를 선형 변환하여 새로운 축(Principal Components, 주성분)을 생성하는 과정
    - **주성분(Principal Components)** : 데이터의 분산을 가장 잘 설명하는 새로운 축
    - **데이터의 주요 패턴 찾기** : 데이터의 분산이 가장 큰 방향을 기준으로 데이터를 분석
    - **축 변경** : 기존의 좌표축(특징 공간)을 새로운 주성분 축으로 변환
    - **차원 축소** : 중요하지 않은 주성분을 제거하여 데이터의 차원을 줄임

PCA(Principal Component Analysis) 개념 및 원리

- **PCA(Principal Component Analysis, 주성분 분석)**

  - PCA의 작동 원리
    - PCA의 원리는 데이터를 새로운 좌표축으로 변환하여 주요 특징을 유지하면서 차원을 줄이는 데 있음

    ① 데이터는 처음에 여러 차원(피처)으로 구성

    ② PCA는 각 차원의 분산(정보량)을 측정하고, 분산이 가장 큰 방향(축)을 찾음

    ③ 분산이 가장 큰 방향으로 데이터를 정렬하여 새로운 축을 정의

    ④ 새로운 축을 기준으로 데이터를 변환(투영)하여, 적은 차원에서도 데이터를 잘 설명할 수 있도록 처리

PCA(Principal Component Analysis) 개념 및 원리

- PCA의 적용 단계

① 데이터 정규화 (Standardization)

- PCA를 적용하기 전에 **데이터를 정규화(Standardization)하거나 스케일링(Scaling)**
- 각 피처의 값이 서로 다른 범위를 갖고 있으면 결과에 영향을 미치기 때문에, **평균이 0이고 분산이 1인 데이터로 변환**

② 주성분 축을 찾기 위한 공분산 행렬 계산

- 공분산 행렬(Covariance Matrix)은 각 피처 간의 상관관계를 나타냄
- 예를 들어, 두 피처가 양의 상관관계에 있으면 공분산 값이 양수
- 공분산 행렬은 데이터의 분산 구조를 파악하는 데 필수적

③ 주성분 축을 찾기 위한 고유값(Eigenvalues)과 고유벡터(Eigenvectors) 계산

- 공분산 행렬을 분해하여 고유값과 고유벡터를 계산
  - 고유값 : 주성분이 설명하는 분산의 크기 (중요도)
  - 고유벡터 : 주성분의 방향 (새로운 축).

PCA(Principal Component Analysis) 개념 및 원리

- PCA의 적용 단계

④ 주성분 선택

- 고유값이 큰 순서대로 주성분을 정렬
- 중요한 정보를 설명하는 주성분(변동성을 많이 설명하는 축)만 선택
  예: 전체 분산의 95% 이상을 설명하는 주성분만 사용.

⑤ 데이터 투영 (Projection)

- 원본 데이터를 선택된 주성분 축으로 투영
- 이렇게 하면 데이터의 차원이 축소

주성분의 의미와 데이터 투영

- **주성분의 의미**
  - 주성분이란?
    - 주성분(Principal Components)은 데이터의 분산(정보량)이 가장 큰 방향을 나타내는 축
    - 데이터를 분석할 때 중요한 정보는 데이터의 분산(변동성) 속에 포함되어 있다고 가정
    - PCA는 데이터의 분산을 최대화하는 방향을 찾아 새로운 좌표축을 정의하고 이를 주성분이라고 함
  - 주성분의 특징
    - **첫 번째 주성분(PC1)**
      - 데이터의 분산이 가장 큰 방향. 데이터의 변동성을 가장 잘 설명하는 축.
    - **두 번째 주성분(PC2)**
      - 첫 번째 주성분에 수직(직교)한 축. 두 번째로 큰 분산을 설명하는 방향
    - 추가 주성분
      - 이전 주성분들과 직교하면서 분산이 큰 방향을 순서대로 찾음
    - 주성분의 수

      데이터의 차원 수와 동일한 수의 주성분이 존재. 예를 들어, 데이터가 4차원이라면 4개의 주성분이 있음

주성분의 의미와 데이터 투영

- **데이터 투영(Projection)의 의미**

  - PCA의 핵심 작업 중 하나는 데이터를 주성분 축으로 투영(Projection)하여 차원을 축소하는 것
  - **데이터 투영**이란?
    - 데이터를 기존의 좌표축 대신 주성분 축으로 변환하는 과정
    - 투영 후 데이터는 새로운 축(주성분) 상에서 표현. 불필요한 차원(변동성이 적은 축)은 제거 됨
  - 왜 투영이 필요한가?
    - **중요한 정보만 유지**
      - 데이터의 분산이 큰 축(주성분)에 투영하면 정보량을 최대한 유지할 수 있음
      - 분산이 작은 축은 주로 노이즈로 간주되어 제거
    - **차원 축소** : 데이터가 주성분 축으로 투영되면, 더 적은 차원에서도 데이터를 잘 설명할 수 있음
    - **시각화** : 투영된 데이터를 2차원이나 3차원으로 변환하여 시각적으로 이해할 수 있음

PCA 예제

```
# 필요한 라이브러리 import
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# 1. Iris 데이터셋 로드
# - Iris 데이터셋은 꽃잎 길이, 꽃잎 너비 등 4개의 피처를 가진 데이터로 구성
iris = load_iris()
X = iris.data  # 특성 데이터 (4차원)
y = iris.target  # 타깃 데이터 (0, 1, 2의 세 가지 클래스)

# 2. 데이터 정규화 (평균=0, 표준편차=1로 변환)
# - PCA는 데이터의 스케일에 민감하므로 정규화가 필수
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 3. PCA 객체 생성 및 적합
# - n_components=2로 설정하여 데이터를 2차원으로 축소
pca = PCA(n_components=2)  # 2개의 주성분으로 축소
X_pca = pca.fit_transform(X_scaled)
```

```
# 4. PCA 결과 출력
# - 각 주성분이 데이터의 분산을 얼마나 설명하는지 출력
print("Explained variance ratio:", pca.explained_variance_ratio_)
# - 첫 번째 주성분과 두 번째 주성분이 전체 분산의 약 95% 이상을 설명

# 5. PCA 결과 시각화
# - 두 개의 주성분을 사용해 데이터를 2D 플롯으로 시각화
plt.figure(figsize=(8, 6))
for target, color, label in zip([0, 1, 2], ['red', 'blue', 'green'],
iris.target_names):
    plt.scatter(X_pca[y == target, 0],  # 첫 번째 주성분
                X_pca[y == target, 1],  # 두 번째 주성분
                color=color, label=label, alpha=0.7)
plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid()
plt.show()
```

PCA 예제

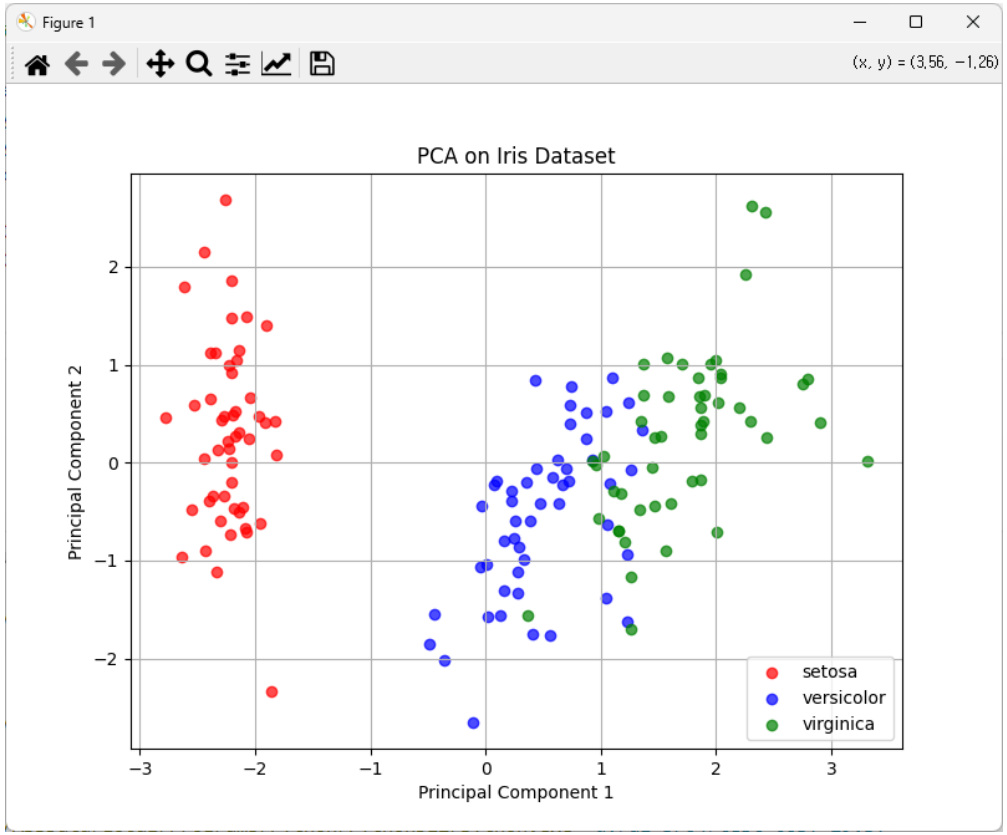


Figure 1
(x, y) = (3.56, −1.26)
PCA on Iris Dataset
Principal Component 2
Principal Component 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
setosa
versicolor
virginica

first coding

고급 차원 축소

t-SNE(T-Distributed Stochastic Neighbor Embedding) 소개

t-SNE를 활용한 데이터 시각화

PCA와 t-SNE의 비교

t-SNE(T-Distributed Stochastic Neighbor Embedding) 소개

- t-SNE(T-Distributed Stochastic Neighbor Embedding)

  - t-SNE는 고차원의 데이터를 저차원(일반적으로 2D 또는 3D) 공간으로 변환하여
    데이터의 구조를 시각화 하는 데 사용되는 비선형 차원 축소 방법
  - 특히, 데이터의 군집(클러스터)이나 복잡한 구조를 시각적으로 분석할 때 매우 유용

  - 왜 t-SNE가 필요한가?
    - 고차원 데이터(이미지 픽셀 데이터, 유전자 표현 데이터 등)는 사람이 직접 이해하거나 시각화하기 어렵다
    - PCA 같은 선형 차원 축소 방법은 고차원 데이터의 선형적인 특성을 잘 다루지만,
      데이터 간의 비선형적인 관계를 포착하기 어렵다
    - t-SNE는 데이터 간의 비선형적 관계를 유지하면서 시각적으로 유의미한 구조를 제공하는 데 강점

t-SNE(T-Distributed Stochastic Neighbor Embedding) 소개

- t-SNE(T-Distributed Stochastic Neighbor Embedding)

  - t-SNE의 핵심 과정

    ① 고차원 공간에서 데이터 간의 유사성 계산

    - 데이터 포인트 i와 j 간의 유사성은 확률로 표현
    - 가우시안 분포를 사용하여 j가 i의 이웃일 확률 pij 를 계산

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}$$

    ② 저차원 공간에서 데이터 간의 유사성 정의

    - 저차원 공간에서는 t-분포(t-distribution)를 사용하여 유사성 확률 qij 를 계산
    - t-분포는 가우시안보다 긴 꼬리를 가지므로 저차원 공간에서 데이터 간의 거리를 더 잘 표현

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

    ③ 고차원과 저차원 공간의 유사성 차이를 줄이기

    - 두 확률 분포 pij 와 qij 의 차이를 줄이는 것이 목표
    - 이 차이를 KL 다이버전스(Kullback-Leibler divergence)로 측정하고, 이를 최소화하는 방식으로 학습

$$KL(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

t-SNE(T-Distributed Stochastic Neighbor Embedding) 소개

- t-SNE(T-Distributed Stochastic Neighbor Embedding)

  - t-SNE의 특징
    - **비선형 차원 축소**
      - 데이터의 비선형적인 구조를 효과적으로 포착
      - 복잡한 데이터의 군집(클러스터)이나 패턴을 시각화하기에 적합
    - **국소 구조 유지**
      - 데이터의 가까운 이웃 간의 관계를 보존하는 데 중점
      - 저차원 공간에서 군집이 명확하게 나타남.
    - **저차원 시각화 최적화**
      - 결과는 저차원 공간에서 데이터 포인트 간의 상대적인 위치를 기반으로 구조를 해석

t-SNE(T-Distributed Stochastic Neighbor Embedding) 소개

- t-SNE(T-Distributed Stochastic Neighbor Embedding)

  - t-SNE의 단점
    - **계산 비용이 높음**
      - 모든 데이터 포인트 간의 유사성을 계산해야 하므로 데이터 포인트 수가 많아질수록 계산 비용이 크게 증가
      - 최근에는 이 문제를 해결하기 위해 GPU와 병렬 처리가 지원되는 개선된 버전들이 사용
    - **전역 구조 표현의 한계**
      - 데이터의 전역적인 구조(전체적인 관계) 는 잘 표현하지 못할 수 있음
      - 데이터의 전체 분포보다는 국소적 관계를 분석할 때 적합
    - **결과 해석의 주관성**
      - 같은 데이터라도 초기값 설정에 따라 다른 결과를 도출할 수 있음

t-SNE(T-Distributed Stochastic Neighbor Embedding) 소개

- t-SNE(T-Distributed Stochastic Neighbor Embedding)

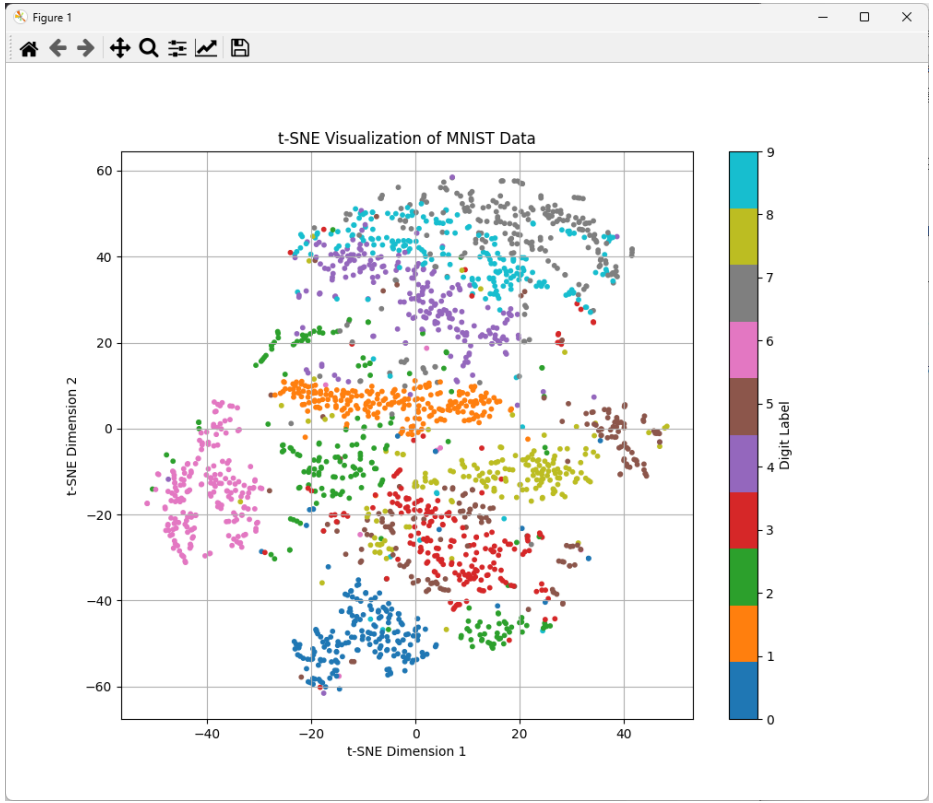
  - t-SNE의 주요 활용 사례
    - **이미지 분석**
      - 고차원 픽셀 데이터를 시각화 하여 이미지의 패턴을 파악
    - **유전자 데이터 분석**
      - 생물학적 데이터의 군집화와 패턴 탐지
    - **자연어 처리**
      - 단어 임베딩(Word Embedding) 벡터를 시각화 하여 단어 간의 의미적 관계 분석

t-SNE 예제

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# 1. MNIST 데이터셋 불러오기
# MNIST 데이터는 28x28 픽셀의 손글씨 숫자 이미지입니다.
print("데이터 로드 중...")
mnist = fetch_openml('mnist_784', version=1)
X, y = mnist.data, mnist.target  # X: 이미지 벡터, y: 레이블

# 2. 데이터 전처리
# t-SNE는 데이터 스케일에 민감하므로 표준화(정규화)가 필요합니다.
print("데이터 전처리 중...")
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# 3. t-SNE 모델 생성
# 데이터 차원을 2D로 줄여 시각화할 예정입니다.
print("t-SNE 모델 생성 중...")
tsne = TSNE(n_components=2, random_state=42, n_iter=1000,
perplexity=30)
X_tsne = tsne.fit_transform(X_scaled[:2000])  # 계산 속도를 위해
2000개 샘플만 사용

# 4. 시각화
# t-SNE로 변환된 결과를 시각화합니다.
print("시각화 중...")
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1],
c=y[:2000].astype(int), cmap='tab10', s=10)
plt.colorbar(scatter, label="Digit Label")
plt.title("t-SNE Visualization of MNIST Data")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.grid(True)
plt.show()
```

t-SNE 예제



- **목적**
  고차원 데이터를 2차원으로 축소해 데이터 간 유사성과 패턴을 시각적으로 표현.

- **구성 요소**
  - 축: 데이터의 상대적 관계를 나타냄
    (값 자체는 해석 불가).
  - 점: MNIST 데이터의 샘플.색상: 숫자 레이블(0~9)을 구분.

- **클러스터 특징**
  - 같은 숫자 레이블은 군집화.
  - 일부 숫자는 겹치는 경우도 있음 (예: 4와 9).

- **활용**
  - 데이터 분포와 패턴 분석.모델 평가 및 데이터 구조 이해.

- **제한점**
  - 고차원 데이터 구조를 완벽히 표현하지 못함.

first coding

이상치 탐지 (Outlier Detection)

이상치 탐지의 개념과 중요성

비지도학습 기반 이상치 탐지 (Isolation Forest, DBSCAN)

응용 분야 (금융 사기 탐지, 장비 이상 감지 등)

이상치 탐지의 개념과 중요성

- **이상치 탐지의 개념**

  - 이상치 탐지(Outlier Detection)
    - 데이터에서 일반적인 패턴과 크게 벗어나는 관측값(이상치, outlier)을 식별하는 과정
    - 데이터 분포와 동떨어져 있거나, 예상 범위를 벗어난 데이터 포인트를 나타냄
      → 다른 데이터와 비교했을 때 비정상적이고, 통계적으로 드문 데이터 포인트

    - 목적
      - 이상치 탐지의 주된 목표는 데이터의 품질을 개선
      - 비정상적인 행동을 사전에 발견하여 문제를 예방

이상치 탐지의 개념과 중요성

- 이상치 탐지의 중요성

  - 금융
    - 신용카드 사용 내역에서 사기 거래 탐지.
    - 일반적인 소비 패턴과 다른 과도한 결제 금액이나 지역 외 사용 패턴은 금융 사기의 징후로 간주
  - 산업 장비 유지보수
    - 장비의 센서 데이터에서 비정상적인 동작 탐지.
    - 정상적인 동작과 다른 패턴을 가진 이상값은 기계 고장의 초기 신호일 수 있음
  - 의료
    - 환자의 건강 데이터에서 비정상적인 생체 신호 탐지.
    - 심박수, 혈압 등에서 극단적인 수치는 질병의 징후일 가능성이 있음

비지도학습 기반 이상치 탐지 (Isolation Forest, DBSCAN)

- Isolation Forest
  - Isolation Forest는 이상치 탐지를 위해 설계된 비지도학습 알고리즘
    - 이 알고리즘은 데이터를 반복적으로 나누는 방식으로 이상치를 탐지
    - 이상치는 일반적인 데이터보다 나누기 쉽다는 특징에 기반
    - 데이터 공간을 무작위로 분할하며, 나눌수록 나머지 데이터에서 이상치가 더 빠르게 분리되는 경향이 있음
  - 작동 원리

    ① **무작위 샘플링**

    데이터에서 임의의 특징(feature)을 선택하고, 임의의 분할 값(split value)을 선택해 데이터를 나눔

    ② **분리 깊이 계산**

    이상치 일수록 다른 데이터로부터 분리되는 데 필요한 단계(depth)가 작음

    ③ **여러 트리 구축**

    데이터를 무작위로 분할하여 여러 개의 이진 트리(isolation tree)를 생성 → 각 데이터 포인트가 나뉘는 평균 깊이를 계산

    ④ **이상치 점수 산출**

    이상치 점수는 분리 깊이에 따라 정해지며, 높은 점수를 받을수록 이상치일 가능성이 높음

비지도학습 기반 이상치 탐지 (Isolation Forest, DBSCAN)

- Isolation Forest

  - 장점
    - 효율성: 데이터 크기와 관계없이 선형 시간복잡도 ($O(n)$)
    - 확장성: 대규모 데이터 세트에서도 작동 가능
    - 비지도학습: 레이블이 없어도 작동.
  - 단점
    - 고차원 데이터에서 효율이 떨어질 수 있음
    - 분포가 매우 복잡한 데이터에는 성능이 제한될 수 있음
  - 주요 응용 분야
    - 금융 사기 탐지 (예: 신용카드 이상 거래)
    - 네트워크 보안 (예: 비정상 트래픽 탐지)

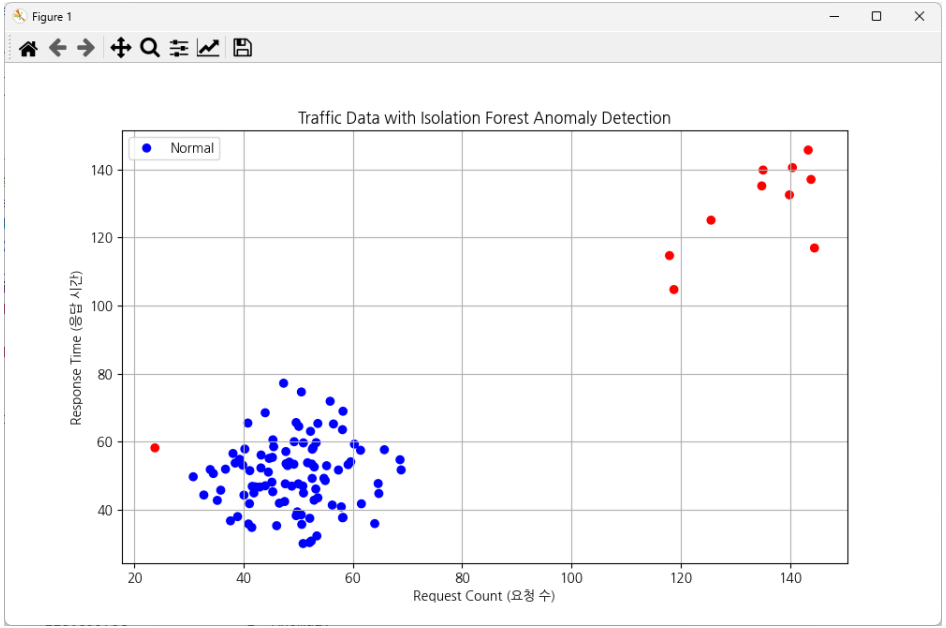
Isolation Forest 예제

```
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt

# 1. 데이터 생성
# 시드 고정: 재현성을 위해 난수 시드를 고정
np.random.seed(42)

# 정상 트래픽 데이터 생성
# 평균(loc=50), 표준편차(scale=10)를 따르는 정규분포에서 데이터 생성 (100개의 샘플, 2차원)
normal_traffic = np.random.normal(loc=50, scale=10, size=(100, 2))

# 비정상 트래픽 데이터 생성
# 일정 범위(low=100, high=150)에서 균등분포로 비정상적으로 높은 값 생성 (10개의 샘플, 2차원)
anomalous_traffic = np.random.uniform(low=100, high=150, size=(10, 2))

# 정상 데이터와 비정상 데이터를 결합
traffic_data = np.vstack((normal_traffic, anomalous_traffic))

# 데이터를 pandas DataFrame으로 변환
traffic_df = pd.DataFrame(traffic_data, columns=["Request Count", "Response Time"])
```

```
# 2. Isolation Forest 모델 생성 및 학습
# Isolation Forest 초기화
# - n_estimators: 트리의 개수 (모델 복잡도에 영향을 미침)
# - contamination: 이상치로 간주할 데이터의 비율 (10%로 설정)
# - random_state: 난수 시드 (재현성을 위해 설정)
model = IsolationForest(n_estimators=100, contamination=0.1, random_state=42)

# 모델 학습 후 예측 수행
# - fit_predict: 데이터를 학습하면서 이상치를 탐지 (-1: 이상치, 1: 정상)
traffic_df['Anomaly Score'] = model.fit_predict(traffic_data)

# 이상 탐지 결과를 해석하여 "Normal" 또는 "Anomaly"로 표시
traffic_df['Anomaly'] = traffic_df['Anomaly Score'].apply(lambda x: "Normal" if x == 1 else "Anomaly")

# 3. 비정상 데이터의 범위 확인 - 이상 데이터만 필터링
anomalous_data = traffic_df[traffic_df['Anomaly'] == "Anomaly"]

# 이상 범위를 출력
min_request_count = anomalous_data["Request Count"].min()
max_request_count = anomalous_data["Request Count"].max()
min_response_time = anomalous_data["Response Time"].min()
max_response_time = anomalous_data["Response Time"].max()
print(f"비정상 데이터 범위:")
print(f"- 요청 수 (Request Count): {min_request_count:.2f} ~ {max_request_count:.2f}")
print(f"- 응답 시간 (Response Time): {min_response_time:.2f} ~ {max_response_time:.2f}")
```

Isolation Forest 예제

```
# 4. 시각화
# 정상과 비정상을 다른 색으로 표시하여 트래픽 데이터 시각화
# 한글 폰트 설정
plt.rcParams['font.family'] = 'NanumGothic'
plt.figure(figsize=(10, 6))
plt.scatter(
    traffic_df["Request Count"], traffic_df["Response Time"],
    c=traffic_df['Anomaly'].map({"Normal": "blue", "Anomaly":
"red"}),  # 색상 매핑
    label='Traffic'
)
plt.title("Traffic Data with Isolation Forest Anomaly Detection")
plt.xlabel("Request Count (요청 수)")
plt.ylabel("Response Time (응답 시간)")
plt.legend(["Normal", "Anomaly"])
plt.grid()
plt.show()

# 5. 결과 출력
# 전체 데이터프레임을 출력하여 정상/비정상 데이터 확인
print("전체 데이터와 이상 탐지 결과:")
print(traffic_df)
```



비지도학습 기반 이상치 탐지 (Isolation Forest, DBSCAN)

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
  - DBSCAN
    - 밀도 기반 클러스터링 알고리즘
    - 데이터를 밀도가 높은 영역(군집)과 밀도가 낮은 영역(이상치)으로 나눔
  - 기본 아이디어
    - 데이터의 밀도가 충분히 높은 영역을 클러스터로 간주
    - 밀도가 낮은 데이터 포인트는 이상치로 분류
  - 주요 개념
    - Epsilon (ε) : 두 데이터 포인트가 같은 클러스터에 속하기 위해 허용되는 최대 거리
    - MinPts : 클러스터가 형성되기 위해 필요한 최소 데이터 포인트 수
    - Core Points : ε 반경 안에 MinPts 이상의 데이터가 포함된 포인트
    - Border Points : ε 반경 내에 Core Point에 포함되지만 MinPts 조건을 만족하지 않는 포인트
    - Noise (이상치) : Core Point나 Border Point에 속하지 않는 포인트

비지도학습 기반 이상치 탐지 (Isolation Forest, DBSCAN)

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
  - 작동 원리
    - 각 데이터 포인트에서 ε 반경 내 데이터 밀도를 계산.
    - 밀도가 높은 데이터 포인트(Core Point)를 중심으로 클러스터 형성.
    - 밀도가 낮은 포인트는 Noise(이상치)로 간주.
  - 장점
    - 클러스터의 모양에 제약이 없음(비구조적 데이터에서도 사용 가능).
    - 사전 클러스터 개수 지정이 불필요.
    - 이상치 탐지와 클러스터링을 동시에 수행.
  - 단점
    - ε와 MinPts 파라미터의 선택이 성능에 크게 영향을 미침.
    - 고차원 데이터에 적용할 경우 효율성 저하.

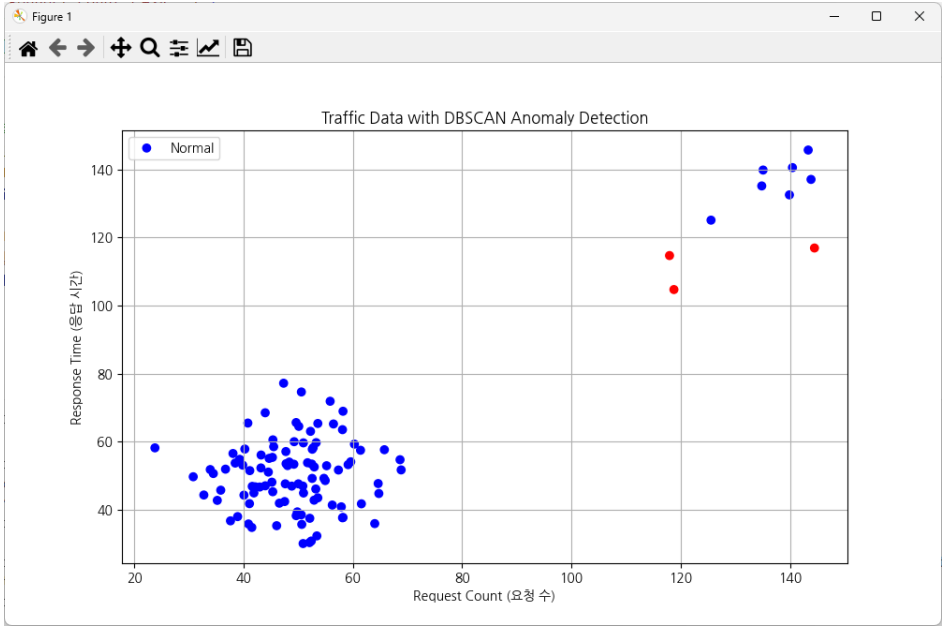
Isolation Forest 예제

```
import numpy as np
import pandas as pd
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# 1. 데이터 생성
# 난수 시드 고정
np.random.seed(42)

# 정상 트래픽 데이터 생성 (평균: 50, 표준편차: 10)
normal_traffic = np.random.normal(loc=50, scale=10, size=(100, 2))

# 비정상 트래픽 데이터 생성 (비정상적으로 높은 값)
anomalous_traffic = np.random.uniform(low=100, high=150, size=(10,
2))

# 데이터 결합
traffic_data = np.vstack((normal_traffic, anomalous_traffic))

# 데이터프레임으로 변환
traffic_df = pd.DataFrame(traffic_data, columns=["Request Count",
"Response Time"])

# 2. DBSCAN 모델 생성
# eps: 클러스터의 반경 (밀도를 정의)
# min_samples: 한 클러스터를 구성하기 위한 최소 샘플 수
dbscan = DBSCAN(eps=15, min_samples=5)
```

```
# DBSCAN 모델 적용
# fit_predict: 클러스터링 수행 (-1은 이상치, 나머지는 클러스터 ID)
traffic_df['Cluster'] = dbscan.fit_predict(traffic_data)

# 3. 이상치 탐지
# 이상치는 Cluster 값이 -1인 데이터
traffic_df['Anomaly'] = traffic_df['Cluster'].apply(lambda x:
"Anomaly" if x == -1 else "Normal")

# 이상치 범위 확인
anomalous_data = traffic_df[traffic_df['Anomaly'] == "Anomaly"]
min_request_count = anomalous_data["Request Count"].min()
max_request_count = anomalous_data["Request Count"].max()
min_response_time = anomalous_data["Response Time"].min()
max_response_time = anomalous_data["Response Time"].max()

print(f"비정상 데이터 범위:")
print(f"- 요청 수 (Request Count): {min_request_count:.2f} ~
{max_request_count:.2f}")
print(f"- 응답 시간 (Response Time): {min_response_time:.2f} ~
{max_response_time:.2f}")
```

Isolation Forest 예제

```
# 4. 시각화
plt.rcParams['font.family'] = 'NanumGothic' # 한글 폰트 설정
plt.figure(figsize=(10, 6))
plt.scatter(
    traffic_df["Request Count"], traffic_df["Response Time"],
    c=traffic_df['Anomaly'].map({"Normal": "blue", "Anomaly":
"red"}),  # 색상 매핑
    label="Traffic"
)
plt.title("Traffic Data with DBSCAN Anomaly Detection")
plt.xlabel("Request Count (요청 수)")
plt.ylabel("Response Time (응답 시간)")
plt.legend(["Normal", "Anomaly"])
plt.grid()
plt.show()

# 5. 결과 출력
# 전체 데이터프레임 출력
print("전체 데이터와 DBSCAN 이상 탐지 결과:")
print(traffic_df)
```




```
비정상 데이터 범위:
- 요청 수 (Request Count): 117.97 ~ 144.50
- 응답 시간 (Response Time): 104.70 ~ 116.90
비정상 데이터:
     Request Count  Response Time  Cluster  Anomaly
102     117.974558     114.679592       -1  Anomaly
108     144.500267     116.899758       -1  Anomaly
109     118.779148     104.699097       -1  Anomaly
```

비지도학습 기반 이상치 탐지 (Isolation Forest, DBSCAN)

- Isolation Forest와 DBSCAN 비교

특징	Isolation Forest	DBSCAN
기본 접근 방식	데이터 분리를 통해 이상치 탐지	밀도 차이를 기반으로 클러스터와 이상치 탐지
적용 데이터	대규모 데이터, 일반적인 분포	밀도 기반의 분포 데이터
확장성	매우 높은 확장성	고차원 데이터에서 성능 저하
설정 파라미터	최소 설정 필요	ε와 MinPts 설정 필요
응용 분야	금융, 보안, 장비 감지	지리 데이터, 이상 행동 탐지

first coding

실습 프로젝트 - 군집화와 차원 축소

비지도학습의 전체 과정을 실습하며 종합적으로 이해

주어진 데이터셋에서 패턴 분석 및 시각화