차원 축소: PCA, t-SNE

- 많은 수의 변수(차원)를 가진 데이터에서 정보 손실을 최소화하면서 변수의 개수를 줄이는 과정
- **차원의 저주(Curse of Dimensionality)** 문제를 해결하고, 데이터를 시각화하며, 모델의 학습 속도를 높이는 데 중요한 역할
- **PCA (Principal Component Analysis, 주성분 분석)**
    - 차원 축소에 가장 널리 사용되는 **선형** 기법
    - 데이터의 분산(variance)을 가장 잘 설명하는 새로운 축(주성분, Principal Component)들을 찾고, 원본 데이터를 이 축들에 투영(projection)하여 차원을 축소
    - 첫 번째 주성분(PC1)은 데이터의 분산을 가장 많이 설명하는 축이며, 두 번째 주성분(PC2)은 PC1과 직교하면서 나머지 분산을 가장 많이 설명하는 축이다.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding)**
    - 고차원 공간에서 데이터 포인트 간의 유사도(거리)를 저차원 공간(주로 2D 또는 3D)에서도 최대한 보존하도록 점들을 배치하는 **비선형** 시각화 기법
    - PCA가 데이터의 전체적인 구조(분산)를 보존하는 데 초점을 맞춘다면, t-SNE는 국소적인 이웃 관계(local neighborhood)를 보존하는 데 중점
    - 고차원 데이터의 복잡한 구조를 시각화하는 데 매우 뛰어난 성능

적용 가능한 상황

- **PCA**:
    - 수백, 수천 개의 변수를 가진 데이터(e.g., 이미지 픽셀 데이터, 유전자 데이터)의 차원을 줄여 머신러닝 모델의 입력으로 사용할 때 (노이즈 감소 및 과적합 방지 효과).
    - 변수들 간에 높은 상관관계가 존재할 때, 이를 제거하고 서로 독립적인 주성분들로 데이터를 표현하고 싶을 때.
    - 데이터의 주요한 분산 방향을 찾아 데이터의 전반적인 구조를 이해하고 싶을 때.
- **t-SNE**:
    - 고차원 데이터셋에 숨겨진 군집(cluster)이나 매니폴드(manifold) 구조를 2D 또는 3D 산점도(scatter plot)로 시각화하여 탐색하고 싶을 때.
    - 딥러닝 모델의 마지막 은닉층(hidden layer)의 출력을 2D로 시각화하여, 모델이 클래스를 얼마나 잘 구분하고 있는지 확인할 때.
    - **주의**: t-SNE는 주로 시각화를 위한 탐색적 분석 도구이며, PCA처럼 모델 학습을 위한 전처리 단계로 사용하는 것은 일반적이지 않습니다. 또한, t-SNE 결과의 군집 간 거리나 크기는 특별한 의미를 갖지 않습니다.

예제 데이터 생성 (Iris 데이터셋)

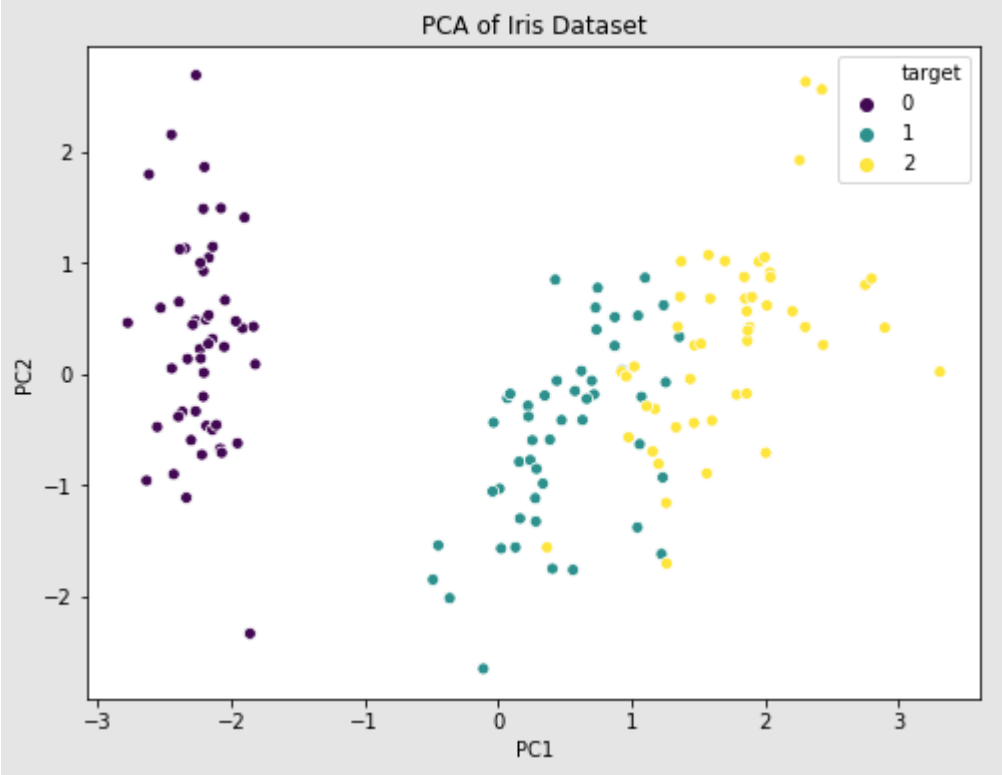
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# 데이터 로드
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# PCA와 t-SNE는 스케일링에 민감하므로 표준화 수행
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.iloc[:, :-1])
```

1. PCA (Principal Component Analysis)

- **용도**: 데이터의 분산을 최대한 보존하는 새로운 저차원 특징 공간을 만듭니다.
- **주의사항**
  - PCA는 스케일에 민감하므로, 적용 전에 스케일링(`StandardScaler` 등) 수행 필수
  - `n_components`를 통해 축소할 차원의 수를 결정
- **코드 예시**

```
from sklearn.decomposition import PCA

# 1. PCA 객체 생성 (2차원으로 축소)
pca = PCA(n_components=2)

# 2. 데이터에 fit_transform 적용
principal_components = pca.fit_transform(scaled_features)

# 3. 결과를 데이터프레임으로 변환
df_pca = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
df_pca['target'] = df['target']

# 설명된 분산 비율 확인
print(f"Explained variance ratio: {pca.explained_variance_ratio_}") #
[0.72962445 0.22850762]
print(f"Total explained variance: {sum(pca.explained_variance_ratio_)}") #
0.9581320720000164

# 시각화
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue='target', data=df_pca,
palette='viridis')
plt.title('PCA of Iris Dataset')
plt.show()
```



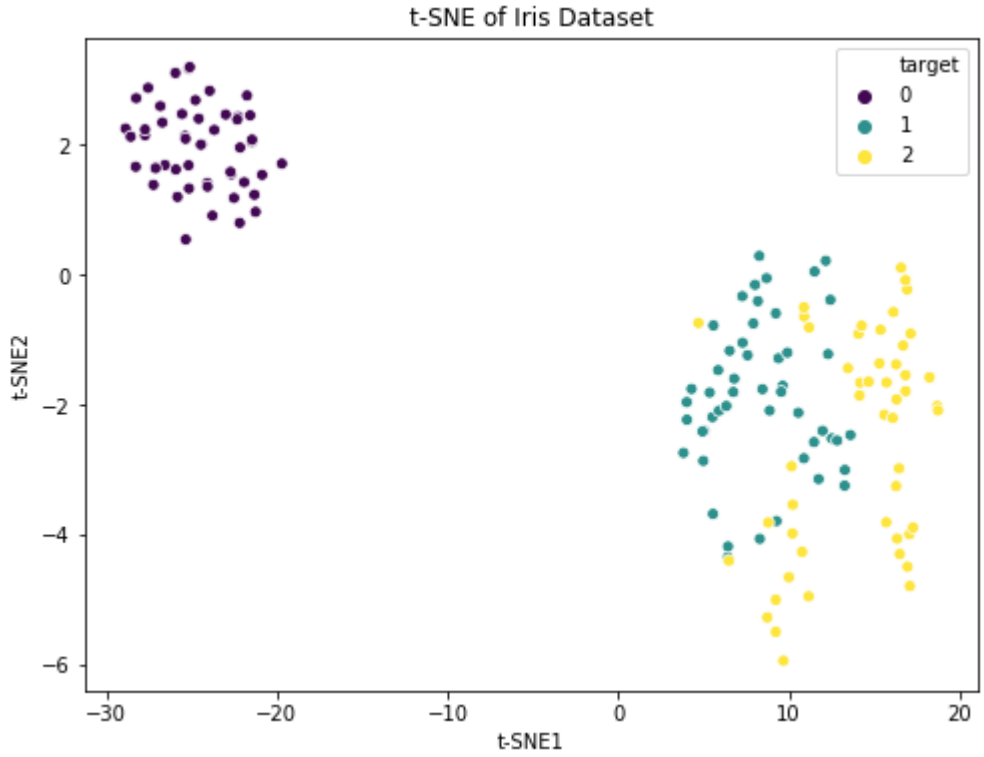
- **결과 해석**
  - 4개의 원본 변수(4차원)가 2개의 주성분(2차원)으로 축소되었습니다.
  - `explained_variance_ratio_`는 각 주성분이 원본 데이터의 분산을 얼마나 설명하는지를 나타냅니다. 예를 들어 `[0.729, 0.228]`이라면, PC1이 약 73%, PC2가 약 23%의 분산을 설명하며, 두 주성분만으로 전체 분산의 약 96%를 설명할 수 있음을 의미합니다.
  - 시각화 결과, 2개의 주성분만으로도 3개의 붓꽃 품종이 잘 구분되는 것을 확인할 수 있습니다.

2. t-SNE

- **용도**: 고차원 데이터의 국소적 구조를 보존하며 저차원으로 시각화합니다.
- **주의사항**
  - 계산 비용이 매우 높아 대용량 데이터에는 적용이 느릴 수 있음
  - PCA로 먼저 50차원 정도로 축소한 후 t-SNE를 적용하는 것이 일반적
  - `perplexity`와 같은 하이퍼파라미터에 결과가 민감하므로, 여러 값으로 시도 필요
- **코드 예시**

```
from sklearn.manifold import TSNE

# 1. t-SNE 객체 생성 (2차원으로 축소)
# perplexity: 각 점이 고려하는 이웃의 수. 보통 5~50 사이 값 사용.
tsne = TSNE(n_components=2, perplexity=30, random_state=42)

# 2. 데이터에 fit_transform 적용
tsne_features = tsne.fit_transform(scaled_features)

# 3. 결과를 데이터프레임으로 변환
df_tsne = pd.DataFrame(data=tsne_features, columns=['t-SNE1', 't-SNE2'])
df_tsne['target'] = df['target']

# 시각화
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='t-SNE1', y='t-SNE2', hue='target', data=df_tsne,
palette='viridis')
plt.title('t-SNE of Iris Dataset')
plt.show()
```



- **결과 해석**
  - t-SNE는 PCA보다 각 품종의 군집을 더 명확하고 뚜렷하게 분리하여 시각화함
    - t-SNE가 데이터 포인트 간의 국소적인 관계를 잘 보존하기 때문
  - 군집 간의 상대적인 거리나 군집의 크기는 실제 데이터의 특성을 반영하는 것이 아니므로 해석에 주의 필요

장단점 및 대안

기법	장점	단점	대안
PCA	선형 변환으로 계산이 빠르고, 결과의 해석이 비교적 용이함 (설명된 분산 등). 변환된 축들이 서로 직교하여 다중공선성 문제를 해결함.	데이터의 비선형 구조를 잘 파악하지 못함. 주성분이 원본 변수들의 선형 결합으로 이루어져 있어, 각 주성분의 의미를 직관적으로 해석하기 어려울 수 있음.	**Kernel PCA**: 커널 트릭을 사용하여 비선형 차원 축소를 수행. **LDA (Linear Discriminant Analysis)**: 분류(classification) 문제에서 클래스를 가장 잘 구분하는 축을 찾는 지도학습 기반의 차원 축소 기법.

기법	장점	단점	대안
t-SNE	복잡한 매니폴드나 군집 구조 등 데이터의 비선형적 패턴을 시각화하는 데 매우 뛰어남.	계산 비용이 높고, 대용량 데이터에 적용하기 어려움. 하이퍼파라미터에 민감하며, 결과의 재현성이 보장되지 않을 수 있음 (`random_state` 지정 필요). 결과(군집 간 거리, 크기)를 해석할 때 주의가 필요함.	**UMAP (Uniform Manifold Approximation and Projection)**: t-SNE와 유사한 목적을 가지지만, 계산 속도가 훨씬 빠르고 데이터의 전역적인 구조도 더 잘 보존하는 경향이 있어 최근 많이 사용됨.

Select KBest

`sklearn.feature_selection.SelectKBest(score_func=<function f_classif>, *, k=10)`
가장 중요한 K개의 특징(feature) 을 자동으로 선택하는 기능입니다. 즉, 입력 데이터(X)의 여러 특성 중에서 통계적으로 유의미한 상위 K개만 남기는 역할을 합니다.

- 하이퍼 파라미터
  - `f_classif`: ANOVA F-value between label/feature for classification tasks.
  - `mutual_info_classif`: Mutual information for a discrete target.
  - `chi2`: Chi-squared stats of non-negative features for classification tasks.
  - `f_regression`:F-value between label/feature for regression tasks.
  - `mutual_info_regression`: Mutual information for a continuous target.
  - `SelectPercentile`: Select features based on percentile of the highest scores.
  - `SelectFpr`: Select features based on a false positive rate test.
  - `SelectFdr`: Select features based on an estimated false discovery rate.
  - `SelectFwe`: Select features based on family-wise error rate.
  - `GenericUnivariateSelect`: Univariate feature selector with configurable mode.
- 지원 메소드
  - `get_support`: Get a mask, or integer index, of the features selected
  - `fit(X, y)`: Run score function on (X, y) and get the appropriate features.

```
# 파이프라인 예시
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif

pipe = Pipeline([
    ('select', SelectKBest(score_func=f_classif, k=3)),
    ('model', SVC())
])

pipe.fit(X, y)
```