

# 차원 축소

PCA · t-SNE · UMAP 원리와 적용

변수가 많을 때 데이터를 이해하고 압축하는 방법을 다룹니다.  
선형회귀와 표준화 개념을 알고 있다고 전제합니다.

# 1. 들어가며: 차원의 저주

변수가 많아질수록 데이터 분석이 어려워집니다. 30 개 변수가 있다면 각 변수 쌍의 관계를 보는 산점도는 435 개, 세 변수 조합은 4060 개나 됩니다. 모두 직접 살펴볼 수는 없습니다. 더 심각한 문제는 고차원에서는 데이터 포인트 간 거리가 의미를 잃는다는 것입니다. 차원이 늘어날수록 모든 점이 서로 비슷한 거리에 있게 되어, 거리 기반 알고리즘(KNN, 클러스터링 등)의 성능이 저하됩니다. 이를 차원의 저주(curse of dimensionality)라 부릅니다.

차원 축소는 데이터가 담고 있는 핵심 정보를 최대한 보존하면서 변수의 수를 줄이는 방법입니다. 목적에 따라 크게 두 가지로 나뉩니다.

- 전처리로서의 차원 축소: 이후 모델(분류, 회귀, 클러스터링)의 성능을 높이거나 계산 속도를 빠르게 하기 위해. PCA 가 주로 사용됩니다.
- 시각화로서의 차원 축소: 고차원 데이터를 2D 또는 3D 로 압축해 구조를 눈으로 확인하기 위해. t-SNE, UMAP 이 주로 사용됩니다.

## 예제 데이터셋

이 가이드는 시리즈 전체와 동일한 Breast Cancer Wisconsin 데이터셋(569 명, 30 개 변수, 악성=1 로 뒤집어 사용)을 사용합니다. 30 개 변수를 더 적은 수의 성분으로 압축할 수 있는지, 압축 후에도 분류 성능이 유지되는지 확인합니다.

# 2. 선형 차원 축소: PCA

## 2-1. PCA 란 무엇인가

주성분분석(PCA, Principal Component Analysis)은 데이터의 분산을 최대한 보존하는 새로운 축(주성분, PC)을 찾는 방법입니다. 직관적으로 설명하면 이렇습니다. 데이터가 퍼져 있는 방향 중 가장 퍼진 방향을 첫 번째 축(PC1)으로 정하고, 그에 수직인 방향 중 다음으로 가장 퍼진 방향을 PC2 로 정합니다. 이렇게 정한 축들은 서로 직교하며, 앞쪽 축일수록 더 많은 정보(분산)를 담습니다.

수학적으로는 공분산 행렬의 고유벡터(eigenvector)를 찾는 문제이지만, 이 가이드에서는 직관과 코드에 집중합니다.

## 2-2. 왜 표준화가 필요한가

PCA 는 분산이 큰 방향을 우선합니다. 변수의 스케일이 다르면 단위가 큰 변수(예: 면적은 mm<sup>2</sup> 단위)가 분산도 크게 나타나 PC1 을 혼자 독점합니다. 규제 선형회귀 가이드에서 다룬 것과 같은 이유입니다.

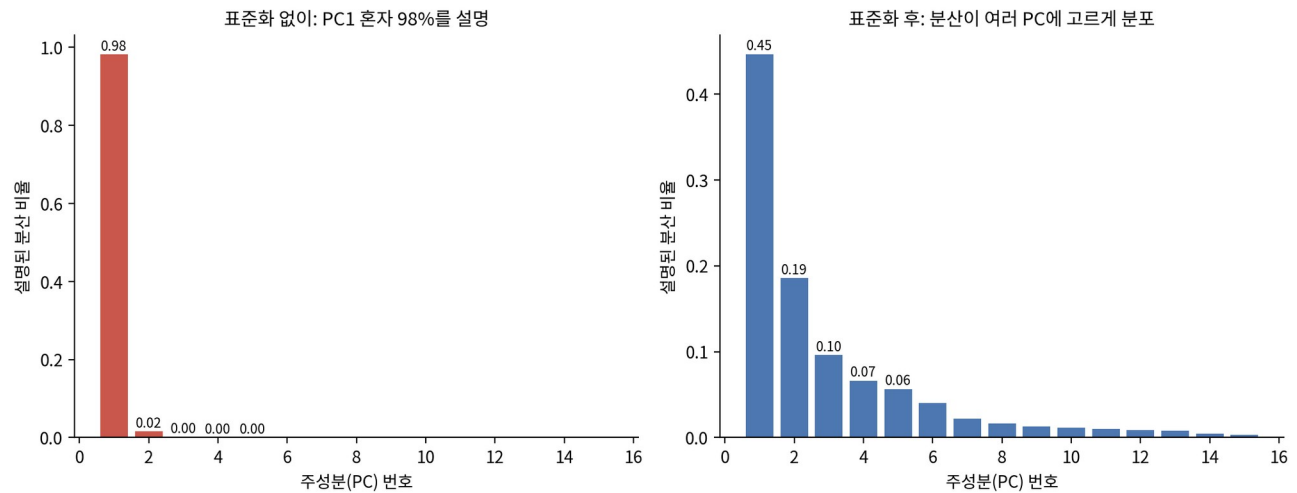


그림 1. Breast Cancer 데이터에 표준화 없이(왼쪽) vs 표준화 후(오른쪽) PCA를 적용한 결과. 표준화 없이는 PC1 하나가 98%를 독점해 나머지 PC가 무의미해진다. 표준화 후에는 분산이 여러 PC에 고르게 분포한다.

### 변수 단위가 서로 다르면 PCA 전 표준화는 필수입니다

표준화 없이 PCA를 돌리면 측정 단위가 큰 변수가 결과를 지배합니다.

StandardScaler(평균 0, 표준편차 1)로 모든 변수를 동일한 스케일로 맞추고 PCA를 적용해야 합니다. 반대로 모든 변수가 같은 단위이고 분산의 크기 차이 자체가 의미 있는 경우(예: 같은 센서의 측정값)에는 표준화하지 않기도 합니다.

규제 선형회귀 가이드에서 다룬 것과 같은 이유입니다: 페널티(또는 분산)가 계수(또는 변수)의 단위에 좌우되면 안 됩니다.

## 2-3. 몇 개의 주성분을 선택할 것인가

PCA 후 가장 중요한 결정은 몇 개의 PC를 사용할지 정하는 것입니다. 두 가지 도구를 함께 씁니다.

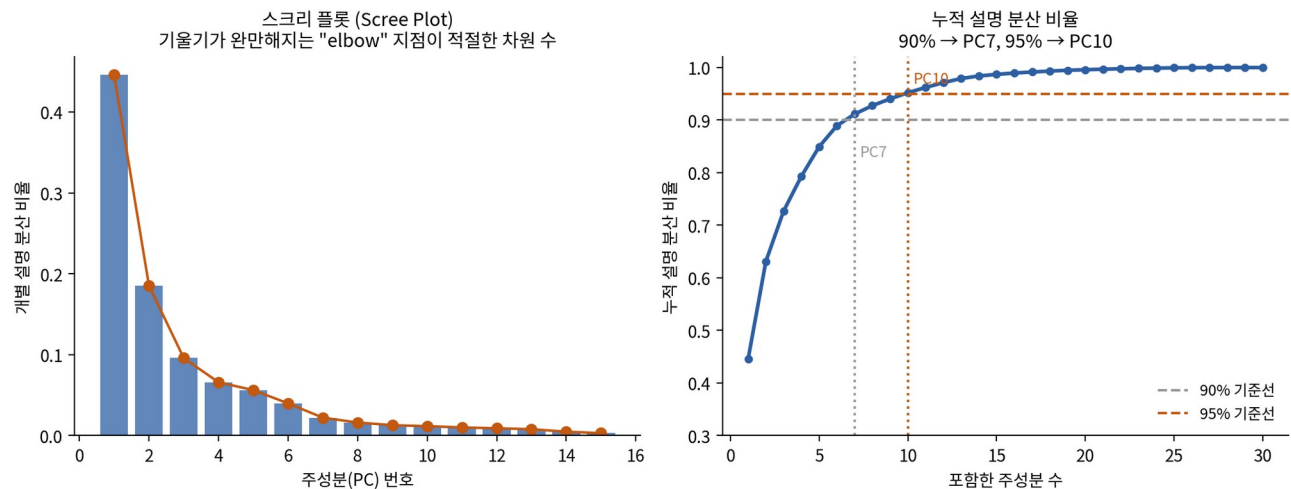


그림 2. 스크리 플롯(왼쪽): 각 PC가 설명하는 분산 비율. 기울기가 꺾이는 'elbow' 지점(약 PC3~4)이 자연스러운 절단점을 제시한다. 누적 분산 비율(오른쪽): 90%를 달성하는 데 PC7, 95%를 달성하는 데 PC10이 필요하다.

- 스크리 플롯(Scree Plot): 각 PC의 설명 분산을 막대로 그린 것. 기울기가 완만해지는 'elbow' 지점이 적절한 차원 수의 힌트를 줍니다.



```
X_test_pca <- as.matrix(X_test_scaled) %*% pca_fit$rotation[, 1:n_comp]
```

## Python

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 1. 표준화 (훈련 데이터로만 fit)
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# 2. 누적 분산 확인 후 PC 수 결정
pca_full = PCA().fit(X_train_s)
import numpy as np
cumvar = np.cumsum(pca_full.explained_variance_ratio_)
n_comp = np.argmax(cumvar >= 0.95) + 1 # 95% 기준 자동 선택
print(f'선택된 PC 수: {n_comp}')
```

```
# 3. PCA 적용
pca = PCA(n_components=n_comp)
X_train_pca = pca.fit_transform(X_train_s)
X_test_pca = pca.transform(X_test_s) # fit 없이 transform 만
```

## 3. 비선형 차원 축소: t-SNE 와 UMAP

### 3-1. PCA 의 한계와 비선형 방법의 필요성

PCA 는 선형 변환입니다. 즉 새 축이 원래 변수들의 선형 결합이어야 합니다. 데이터의 구조가 비선형적이라면 (예: 동심원 모양, 나선 모양의 클러스터) PCA 로는 잘 분리되지 않을 수 있습니다. 또한 PCA 의 2D 투영에서 점들이 섞여 보이더라도, 실제로는 휘어진(비선형) 저차원 구조가 숨어 있을 수 있습니다.

t-SNE 와 UMAP 은 이런 경우에 유용합니다. 이 방법들은 고차원에서 가까운 점들을 저차원에서도 가깝게 유지하는 것을 목표로 합니다.

### 3-2. t-SNE 의 작동 원리

t-SNE(t-Distributed Stochastic Neighbor Embedding)는 두 단계로 작동합니다. 먼저 고차원에서 모든 점 쌍의 유사도를 확률로 변환합니다(가까운 점 = 높은 확률). 그다음 저차원에서의 점 배치를 반복적으로 최적화해, 저차원의 유사도 분포가 고차원과 최대한 비슷해지도록 만듭니다.

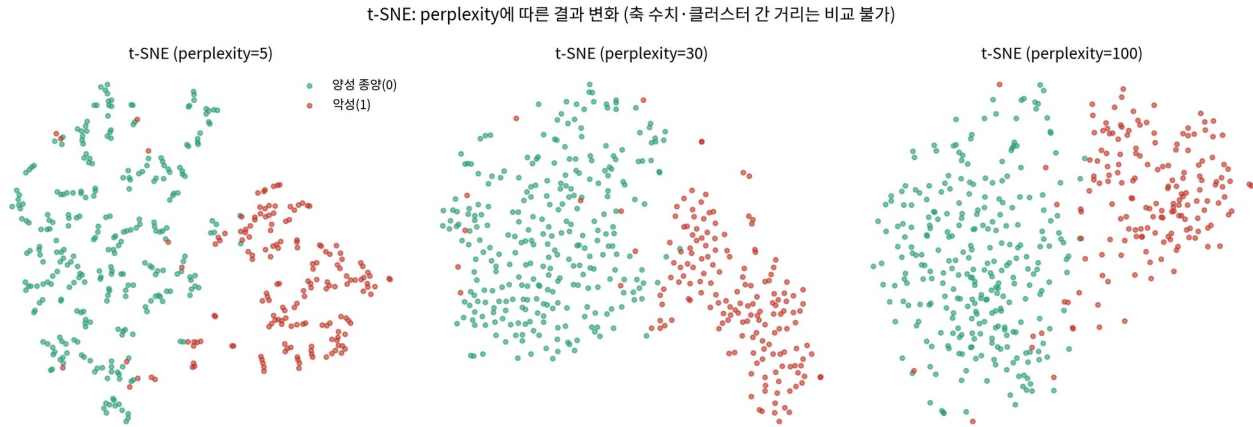


그림 4. Breast Cancer 훈련 데이터에 t-SNE 를 적용한 결과(perplexity 5, 30, 100). perplexity 가 작으면 지역 구조(작은 클러스터)가 뚜렷하고, 크면 전체 구조(두 클래스의 분리)가 드러난다. 세 결과에서 좌표의 절대적 수치나 클러스터 간 거리는 의미가 없고, 클러스터의 형태와 분리 여부만 참고해야 한다.

#### t-SNE 사용 시 반드시 알아야 할 제약

축 간 거리나 클러스터 간 거리는 해석할 수 없습니다. '두 클러스터가 멀리 떨어져 있다'는 것이 실제로 멀다는 의미가 아닙니다.

결과가 무작위 초기값(random\_state)에 따라 달라집니다. 여러 번 실행해서 일관된 구조만 신뢰하세요.

새 데이터를 기존 결과에 투영할 수 없습니다. 예측 파이프라인에 t-SNE 를 쓰면 안 됩니다.

계산이 느립니다. 샘플이 수만 개 이상이면 PCA 로 50~100 차원으로 먼저 줄인 뒤 t-SNE 를 적용하는 것이 일반적입니다.

### 3-3. UMAP: t-SNE 보다 빠르고 구조를 더 잘 보존

UMAP(Uniform Manifold Approximation and Projection)은 t-SNE 와 비슷한 목적을 가지지만 다른 수학적 기반(위상수학과 리만 기하학)을 사용합니다. 실용적인 관점에서의 차이는 다음과 같습니다.

- 속도: t-SNE 보다 훨씬 빠릅니다. 대용량 데이터에서 t-SNE 가 수 시간 걸릴 때 UMAP 은 몇 분 안에 됩니다.
- 전역 구조 보존: t-SNE 는 지역 구조(작은 클러스터 내부)는 잘 보존하지만 클러스터 간의 전역 관계는 잘 보존하지 못합니다. UMAP 은 두 가지를 더 균형 있게 보존하는 편입니다. 다만 초기화와 파라미터에 따라 달라지므로, UMAP 에서도 클러스터 간 거리를 그대로 해석해서는 안 됩니다.
- 변환 가능: UMAP 은 훈련 후 새 데이터를 같은 저차원 공간에 투영할 수 있습니다. 이 때문에 t-SNE 와 달리 전처리 파이프라인에 포함할 수 있습니다.

#### UMAP 설치

UMAP 은 scikit-learn 에 내장되어 있지 않아 별도 설치가 필요합니다.

pip install umap-learn 또는 conda install -c conda-forge umap-learn

R: install.packages('uwot') 또는 install.packages('umap')

### 3-4. R / Python 코드

R

```
# t-SNE
library(Rtsne)
```

```

set.seed(42)
# PCA로 먼저 줄인 뒤 t-SNE 권장 (대용량 데이터)
tsne_fit <- Rtsne(X_scaled, dims=2, perplexity=30,
                 verbose=FALSE, max_iter=1000)
plot(tsne_fit$Y, col=y_train+1, pch=19, cex=0.6,
     main='t-SNE 2D 투영')

# UMAP
library(uwot)
umap_fit <- umap(X_scaled, n_components=2)
plot(umap_fit, col=y_train+1, pch=19, cex=0.6,
     main='UMAP 2D 투영')

```

## Python

```

from sklearn.manifold import TSNE

# t-SNE (대용량이면 PCA 먼저 적용 권장)
tsne = TSNE(n_components=2, perplexity=30,
            random_state=42, max_iter=1000)
X_tsne = tsne.fit_transform(X_train_s)

import matplotlib.pyplot as plt
plt.scatter(X_tsne[:,0], X_tsne[:,1], c=y_train, cmap='RdYlGn', s=10)
plt.title('t-SNE 2D 투영'); plt.show()

# UMAP (별도 설치 필요: pip install umap-learn)
import umap
reducer = umap.UMAP(n_components=2, random_state=42)
X_umap = reducer.fit_transform(X_train_s)
# 새 데이터 투영 가능 (t-SNE와 다름)
X_test_umap = reducer.transform(X_test_s)

```

## 4. PCA vs Lasso: 차원 축소와 변수 선택의 차이

둘 다 "변수가 너무 많을 때 쓰는 방법"으로 묶이기 쉽지만, 접근 방식이 근본적으로 다릅니다.

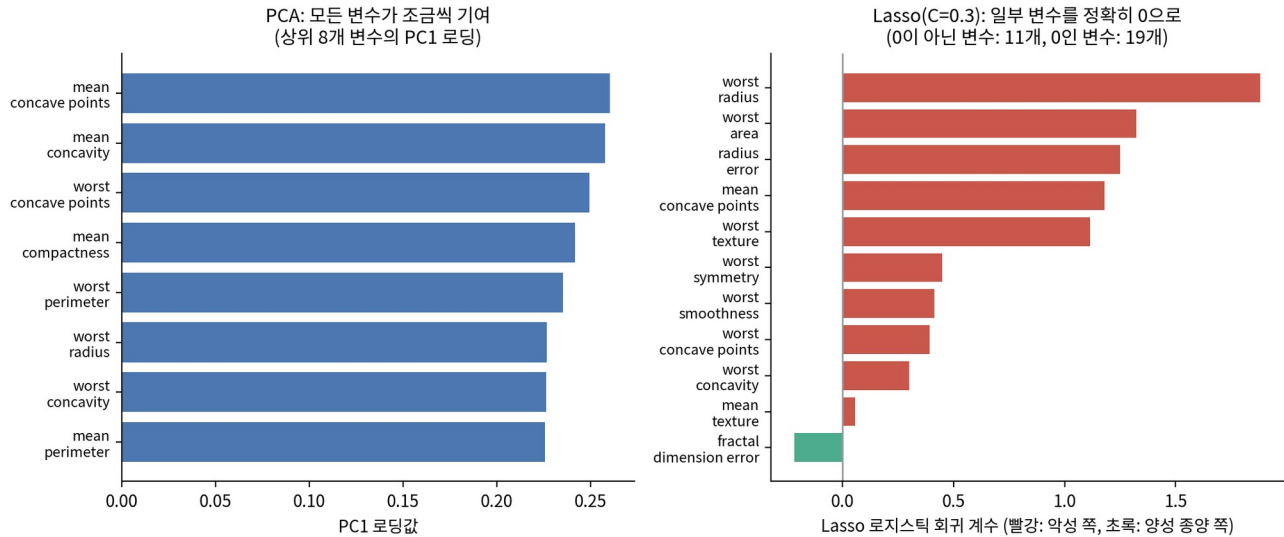


그림 5. PCA의 PC1 로딩(왼쪽): 30개 변수 중 상위 8개만 보여줬지만, 실제로는 모든 변수가 조금씩 기여한다. Lasso 로지스틱 회귀(오른쪽, C=0.3): 30개 중 19개 변수의 계수를 정확히 0으로 만들고, 나머지 11개만 남겼다. 빨강은 악성 쪽, 초록은 양성 종양 쪽으로 작용하는 변수다.

|             | PCA                 | Lasso (변수 선택)        |
|-------------|---------------------|----------------------|
| 새 변수를 만드는가  | 예: 원래 변수의 선형 결합(PC) | 아니오: 원래 변수 일부를 선택    |
| 해석 가능성      | PC의 의미 파악이 어려움      | 남은 변수의 의미를 그대로 해석    |
| 정보 손실       | 설명 분산 기준으로 최소화      | 버린 변수의 정보는 완전히 손실    |
| 목적          | 차원 압축, 노이즈 제거, 시각화  | 변수 선택, 모델 해석, 과적합 방지 |
| 목표 변수(y) 사용 | 사용하지 않음 (비지도)       | 사용함 (지도)             |

선택 기준은 의외로 간단합니다. '어떤 변수가 중요한지 알고 싶다'면 Lasso, '데이터의 구조를 압축하거나 시각화하고 싶다'면 PCA입니다. 두 방법을 결합할 수도 있지만, PCA 성분에 Lasso를 적용하면 선택되는 것이 원래 변수가 아니라 성분이므로 Lasso의 장점인 "어떤 변수가 중요한가"라는 해석이 사라진다는 점에 주의해야 합니다.

이 가이드는 PCA(선형 차원 축소, 분산 보존), t-SNE·UMAP(비선형, 시각화 중심), PCA와 Lasso의 차이를 다루었습니다. 커널 PCA, Autoencoder 기반 차원 축소, Factor Analysis 등은 이 가이드의 범위를 벗어나며 후속 자료에서 다룰 수 있습니다.