차원축소

Sang Yup Lee

차원축소

차원 축소

- 차원의 저주 (curse of dimensionality)
  - High dimensions => Large number of features (or IVs)
  - Possible problems
    - 과적합
    - 군집화 결과가 좋지 않다. <= 데이터포인트들 간의 거리가 유사하게 되는 문제
- 차원 축소의 방법
  - Feature selection
    - 원래의 features 들 중에서 일부만 선택
  - Feature extraction
    - 원래 feature들을 그대로 사용하는 것이 아니라, 원래 feature들이 가지고 있는 정보를 사용하여 새로운 feature들을 추출
    - 이렇게 새롭게 추출되는 feature들의 수는 원래 feature들의 수보다 작다.

차원축소

- Feature selection
  - 데이터셋에 존재하는 원래의 feature 들 중에서 문제를 푸는데 있어 중요한 역할을 하는 몇 개의 feature를 선택하여 최종 분석에서 사용
  - Example
    - 전체 features: age, experience, gender, height, weight → 이중에서 age, experience, gender를 선택하여 최종 분석에서 사용
    - 단점: 선택되지 않은 features (예, height, weight 등)가 갖고 있는 정보를 최종 분석에서 사용하지 못한다.

차원축소

- Feature extraction
  - 데이터셋에 존재하는 원래의 feature들의 정보를 사용하지만, 원래 feature들 그대로를 최종분석에서 사용하지는 않는다.
  - 원 feature들이 가지고 있는 정보 (분산 정보)를 활용해서 새로운 종류의 feature 를 생성해서 (이를 extraction이라고 함) 최종 분석에서 사용한다.
  - 장점: 데이터셋에 존재하는 feature들을 버리지 않고, feature들이 가지고 있는 많은 정보를 사용할 수 있다.
  - Example
    - 전체 features: age, experience, gender, height, weight
    - 각 feature들이 가지고 있는 정보를 사용해서 feature 들의 정보를 담고 있는 새로운 feature 2개를 사용
    - 이렇게 새롭게 생성된 feature들은 실제로 존재하는 어떠한 변수가 아님

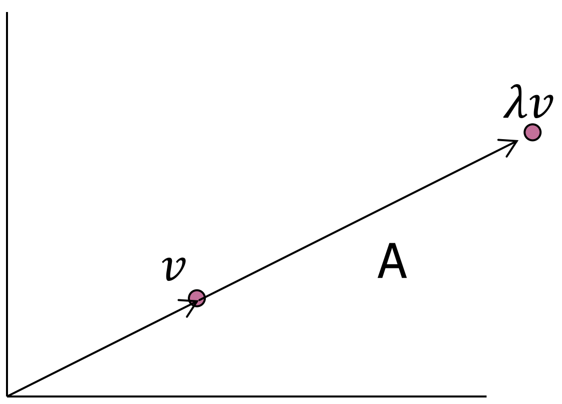
차원축소

- Principal Component Analysis (주성분분석)
  - Feature extraction 방법
  - Principal component
    - 원 데이터가 가지고 있는 정보, 즉, 원 독립변수들이 가지고 있는 정보 (분산으로 표현)를 설명하는 축들
    - 전체 PC의 수 = 전체 독립변수의 수
    - 각 PC에 의해서 설명되는 분산의 크기 (즉, 정보의 양)이 다르다.
    - 설명하는 분산의 크기에 따라 정렬 가능
      - 첫번째 PC = 분산을 가장 많이 설명하는 축
      - 두번째 PC = 분산을 두번째로 많이 설명하는 축
      - …
    - 각 PC는 서로 수직

고유값과 고유벡터

고유값과 고유벡터

- 고유값과 고유벡터 (eigenvalues, eigenvectors)
- 정의
  - $Av = \lambda v$
  - A는 **<u>nxn 행렬</u>**, v는 nx1 벡터 (≠영벡터), λ는 스칼라값
  - 위의 식을 만족하는v를 A의 고유벡터 ($v \neq 0$) , λ를 A의 고유값
- 기하학적 의미
  - 고유벡터는 행렬 A에 의해 선형변환되는 경우 방향은 바뀌지 않고, 길이만 달라지는 벡터
  - $Av = \lambda v$는 고유벡터에 대한 A의 사상은 고유벡터를 스칼라배한 것과 같다
  - **<u>즉, Av는v 벡터의 방향은 바꾸지 않고, 크기만 변경시킨다는 것</u>**
  - 고유벡터는 A 행렬의 고유한 특성을 나타내는 벡터



고유값과 고유벡터

- 고유값과 고유벡터 (손으로) 구하기
  - $Av = \lambda v$은 아래와 같이 변형
  - (A-λI)v= 0
    - 여기서 I는 nxn 단위행렬
    - 이식이 0이 아닌 v에 대해서 만족하기 위해서는 (A-λI)의 역행렬이 존재하지 않아야 함
    - $(A - \lambda I)$의 역행렬이 존재하는 경우에는 위의 식을 만족하는 v는 0벡터밖에 없음
  - $\det(A - \lambda I) = 0$

고유값과 고유벡터

- 고유값과 고유벡터 (손으로) 구하기 (cont'd)
  - 예) $A = \begin{bmatrix} 5 & 1 \\ 3 & 3 \end{bmatrix}$, $(A - \lambda\mathbf{1}) = \begin{bmatrix} 5-\lambda & 1 \\ 3 & 3-\lambda \end{bmatrix}$, $det(A - \lambda\mathbf{1}) = (5-\lambda)(3-\lambda) - 3 = 0$
  - λ = 2, 6
  - 고유벡터: $(A - \lambda\mathbf{1})v = \begin{bmatrix} 5-\lambda & 1 \\ 3 & 3-\lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$
    - λ = 2 의 경우: $\begin{bmatrix} 3 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$, 3x + y = 0을 만족하는 모든 x, y가 λ = 2 에 대한 고유벡터가 됨
    - 즉, 고유벡터는 여러개 나올 수 있음. 왜냐하면 고유벡터는 방향성만이 중요하기 때문. 보통은 여러개의 고유벡터들 중에서 그 길이가 1인 고유벡터를 선택.
    - λ = 6 의 경우, x = y

고유값과 고유벡터

- Numpy를 이용해서 고유값과 고유벡터 구하기
  - See "eigen_examples.ipynb"

```
A = np.array([[5, 1],
              [3, 3]])
eigVals, eigVecs = np.linalg.eig(A)
```

  - eigVecs
    - Unit vectors
      - 방향이 중요하기 때문에
    - v1 = eigVecs[:, 0]
    - v2 = eigVecs[:, 1]
    - Check out if v1 satisfies x = y and v2 satisfies y = -3x

고유값과 고유벡터

- 고유값의 특성
  - 행렬식 (determinant) = 고유값들의 곱
    - 예) $A = \begin{bmatrix} 5 & 1 \\ 3 & 3 \end{bmatrix}$
      - 고유값 = 6, 2
      - 행렬식 = 12 = 5*3 – 3*1
  - tr(A) = 고유값들의 합
    - tr(A) = 대각 성분의 합
    - 5+3 = 6+2

고유값과 고유벡터

- 고유값의 특성
  - $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$
  - $Av = \lambda v$
  - How to find eigenvalues?
  - $det(A - \lambda\mathbf{1}) = (a - \lambda)(d - \lambda) - bc = 0$
  - $\lambda^2 - \lambda(a + d) + ad - bc = 0$
  - 두 근의 합과 곱은?
  - 위의 식을 만족하는 고유값: λ_1, λ_2
    - Then, $(\lambda - \lambda_1)(\lambda - \lambda_2) = 0$

고유값과 고유벡터

- 행렬식과의 관계
  - 행렬식 (determinant) = 고유값들의 곱
    - What if a matrix is not full rank
  - 예) $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$
    - 고유값 = 0, 5
    - 행렬식 = 0 = 5*0
    - => 역행렬이 존재하지 않는다!

고유값과 고유벡터

- 대칭행렬의 고유벡터
  - 서로 수직이다.
    - 즉, 사이각 = 90도
    - cosθ = 0
  - 예)
    - $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$

고유분해 (Eigendecomposition)

Eigendecomposition

- Eigendecomposition (고유 분해)
  - $\mathrm{A} = \mathrm{V}\Lambda\mathrm{V}^{-1}$
    - A: nxn 정사각행렬
    - V: A의 고유벡터들을 열로 갖는 행렬
    - Λ: 고유값들을 대각성분으로 갖는 대각행렬
      - 즉, $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$
    - Example)
      - A: 2x2 행렬, then $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2)$,
      - v1 = $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$, v2 = $\begin{bmatrix} 1 \\ -3 \end{bmatrix}$, then V = $\mathrm{V} = \begin{bmatrix} 1 & 1 \\ 2 & -3 \end{bmatrix}$

Eigendecomposition

- How to derive?
  - $\mathrm{AV} = \mathrm{V\Lambda}$
  - Example
    - $\mathrm{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$
    - $v_1 = \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix}, v_2 = \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix}, then\ V = \begin{bmatrix} v_{11} & v_{21} \\ v_{12} & v_{22} \end{bmatrix}$
    - $\mathrm{AV} = [Av_1 \quad Av_2]$
    - $\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, V = [v_1 \quad v_2], where\ v_1$: λ_1에 대한 eigenvector
    - $v_1 = \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix}, v_2 = \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix}$, then $\mathrm{V\Lambda} = [\lambda_1 v_1 \quad \lambda_2 v_2]$

Eigendecomposition

- How to derive? (cont'd)
  - $\mathrm{AV} = \mathrm{V\Lambda}$
    - V must be full rank, that is the eigenvectors must be linearly independent
    - Then, V^{-1} 존재
  - $\mathrm{AVV^{-1}} = \mathrm{V\Lambda V^{-1}}$

Eigendecomposition

- Example
  - For $A = \begin{bmatrix} 5 & 1 \\ 3 & 3 \end{bmatrix}$
  - $V = \begin{bmatrix} 0.707106 & -0.31622 \\ 0.707106 & 0.94868 \end{bmatrix}, \Lambda = \begin{bmatrix} 6 & 0 \\ 0 & 2 \end{bmatrix}$
  - Python code: See “eigen_examples.ipynb”

Eigendecomposition

- eigendecomposition
  - $A = V\Lambda V^{-1}$은 언제 사용할 수 있는가?
  - 1) A 변환이 여러번 수행되는 경우를 간단하게 계산 가능
    - $A^2 = AA = V\Lambda V^{-1}V\Lambda V^{-1} = V\Lambda^2 V^{-1}$, $\Lambda^2 = \begin{bmatrix} {\lambda_1}^2 & 0 \\ 0 & {\lambda_2}^2 \end{bmatrix}$
  - 2) PCA 차원축소
    - 정확하게는 eigendecomposition이 사용되기 보다는 eigenvalues와 eigenvectors가 사용됨

차원축소

차원 축소

- 차원의 저주 (curse of dimensionality)
  - High dimensions => Large number of features (or IVs)
  - Possible problems
    - 과적합
    - 군집화 결과가 좋지 않다. <= 데이터포인트들 간의 거리가 유사하게 되는 문제
- 차원 축소의 방법
  - Feature selection
    - 원래의 features 들 중에서 일부만 선택
  - Feature extraction
    - 원래 feature들을 그대로 사용하는 것이 아니라, 원래 feature들이 가지고 있는 정보를 사용하여 새로운 feature들을 추출
    - 이렇게 새롭게 추출되는 feature들의 수는 원래 feature들의 수보다 작다.

차원축소

- Feature selection
  - 데이터셋에 존재하는 원래의 feature 들 중에서 문제를 푸는데 있어 중요한 역할을 하는 몇 개의 feature를 선택하여 최종 분석에서 사용
  - Example
    - 전체 features: age, experience, gender, height, weight → 이중에서 age, experience, gender를 선택하여 최종 분석에서 사용
    - 단점: 선택되지 않은 features (예, height, weight 등)가 갖고 있는 정보를 최종 분석에서 사용하지 못한다.

차원축소

- Feature extraction
  - 데이터셋에 존재하는 원래의 feature들의 정보를 사용하지만, 원래 feature들 그대로를 최종분석에서 사용하지는 않는다.
  - 원 feature들이 가지고 있는 정보 (분산 정보)를 활용해서 새로운 종류의 feature 를 생성해서 (이를 extraction이라고 함) 최종 분석에서 사용한다.
  - 장점: 데이터셋에 존재하는 feature들을 버리지 않고, feature들이 가지고 있는 많은 정보를 사용할 수 있다.
  - Example
    - 전체 features: age, experience, gender, height, weight
    - 각 feature들이 가지고 있는 정보를 사용해서 feature 들의 정보를 담고 있는 새로운 feature 2개를 사용
    - 이렇게 새롭게 생성된 feature들은 실제로 존재하는 어떠한 변수가 아님

차원축소

- Principal Component Analysis (주성분분석)
  - Feature extraction 방법
  - Principal component
    - 원 데이터가 가지고 있는 정보, 즉, 원 독립변수들이 가지고 있는 정보 (분산으로 표현)를 설명하는 축들
    - 전체 PC의 수 = 전체 독립변수의 수
    - 각 PC에 의해서 설명되는 분산의 크기 (즉, 정보의 양)이 다르다.
    - 설명하는 분산의 크기에 따라 정렬 가능
      - 첫번째 PC = 분산을 가장 많이 설명하는 축
      - 두번째 PC = 분산을 두번째로 많이 설명하는 축
      - …
    - 각 PC는 서로 수직

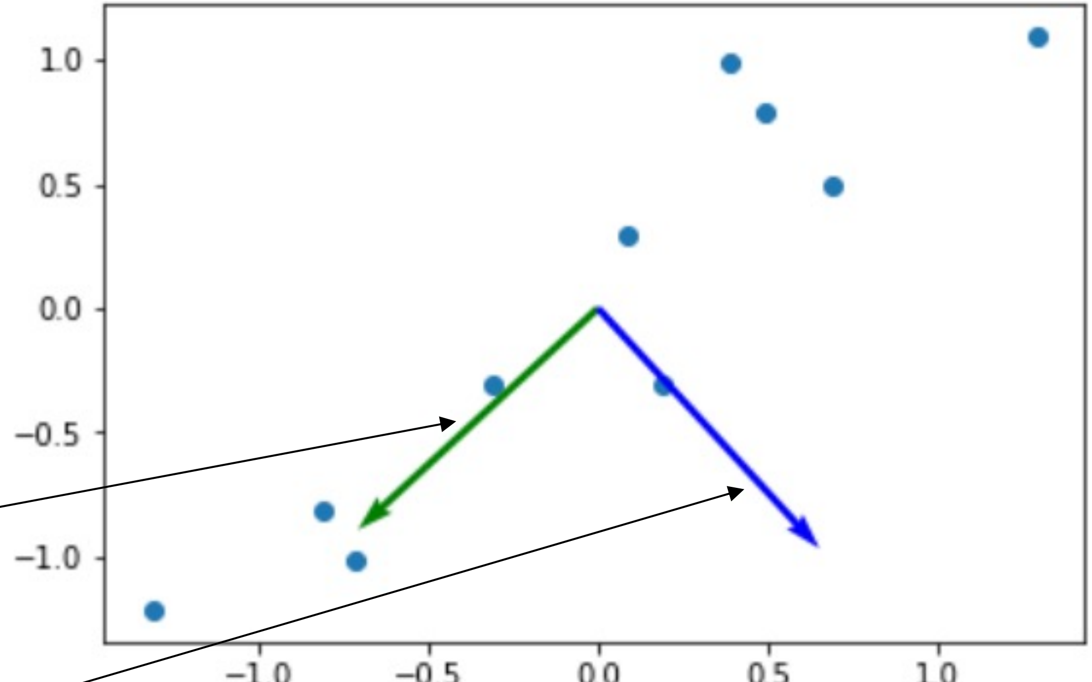
차원축소

- Principal Component Analysis (cont'd)
  - 이러한 PC 들 중에서 분산을 많이 설명하는 상위 몇개의 PC만을 선택하여 사용
    - 이를 이용하여 원데이터를 다시 표현
  - 효과: 원 데이터의 정보 (즉, 분산)은 별로 손실하지 않으면서 feature수를 줄이는 효과

차원축소

- Example

독립변수의 수 = 2
따라서 PC의 수 = 2



첫번째 PC
분산을 가장 많이 설명하는 축

두번째 PC
첫번째 PC가 설명하지 못하는 분산을 설명
첫번째 PC와 서로 수직

차원축소

- PCA (cont'd)
  - 예제 데이터 (# of features = 2, # data points = 10)

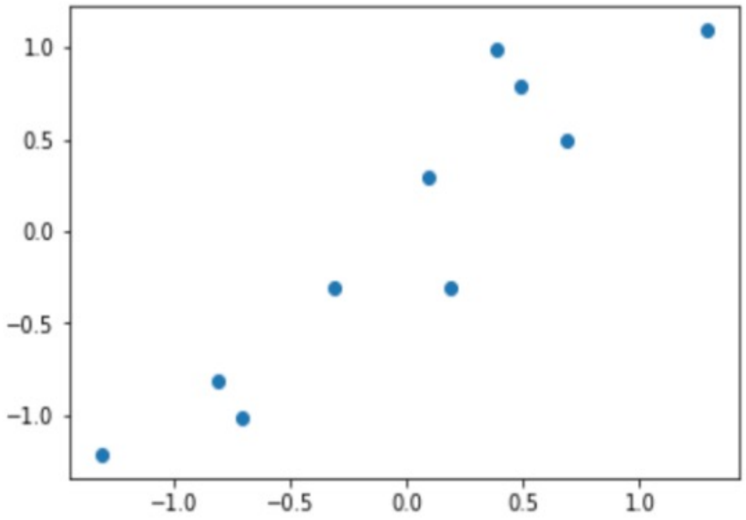
	X1	X2
0	2.5	2.4
1	0.5	0.7
2	2.2	2.9
3	1.9	2.2
4	3.1	3.0
5	2.3	2.7
6	2.0	1.6
7	1.0	1.1
8	1.5	1.6
9	1.1	0.9

Feature를 하나만 선택해서 사용하고자 하는 경우

- X1 또는 X2 중 하나만을 선택하게 되면 (feature selection), 선택되지 않은 변수의 정보를 모두 손실
- 이를 방지하기 위해 PCA 기반의 feature extraction 방법 사용
- 즉, 2개의 PC 중에서 원 데이터의 분산을 더 많이 설명하는 하나만 선택

차원축소

- PCA (cont'd)
  - Mean centering
    - PCA는 원래의 값을 사용하지 않고, mean centering값을 사용
    - Mean centering value = 원값 - 평균



차원축소

- PCA (cont'd)
  - 순서
    - 원데이터의 각 독립변수에 대해서 mean centering 한다 (즉, 원래 값에서 해당 변수의 평균을 뺀다).
    - 원데이터에 대한 공분산 행렬을 만든다.
    - 공분산 행렬에 대해서 고유분해를 수행한다 (즉, 고유값과 고유벡터를 찾는다).
    - 각 고유벡터가 우리가 찾고자 하는 PC가 된다.
    - 우리는 이중에서 설명력이 높은 PC만을 선택한다.
      - 몇개를 선택하는지는 사용자가 결정
      - 이렇게 선택된 PC가 우리가 최종적으로 사용하고자 하는 독립변수가 됨 (즉, feature 가 됨)
    - 새로 구한 PC에 대해 각 관측치의 새로운 값 구하기

차원축소

- PCA (cont'd)

	X1	X2
0	2.5	2.4
1	0.5	0.7
2	2.2	2.9
3	1.9	2.2
4	3.1	3.0
5	2.3	2.7
6	2.0	1.6
7	1.0	1.1
8	1.5	1.6
9	1.1	0.9

	PC1
0	?
1	?
2	?
3	?
4	?
5	?
6	?
7	?
8	?
9	?

PCA

- 원데이터를 이용해서 공분산 행렬 구하기
  - 공분산 행렬
  - Cov(X) = $\begin{bmatrix} Var(X_1) & Cov(X_1,X_2) & \cdots & Cov(X_1,X_K) \\ Cov(X_2,X_1) & Var(X_2) & \cdots & Cov(X_1,X_K) \\ \cdots & \cdots & & \cdots \\ Cov(X_K,X_1) & Cov(X_K,X_2) & \cdots & Var(X_K) \end{bmatrix}$
  - $\mathrm{Var}(\mathrm{X}_1) = {\sum_i^n (x_{1i}-\bar{X}_1)^2}/_{n-1}$
  - $\mathrm{Cov}(X_1, X_2) = {\sum_i^n (x_{1i}-\bar{X}_1)(x_{2i}-\bar{X}_2)}/_{n-1}$
  - 전체 분산 = $Var(X_1) + \cdots + Var(X_K)$
  - Cov(X) is a symmetric matrix
  - Numpy나 pandas를 이용해서 쉽게 구할 수 있다.

PCA

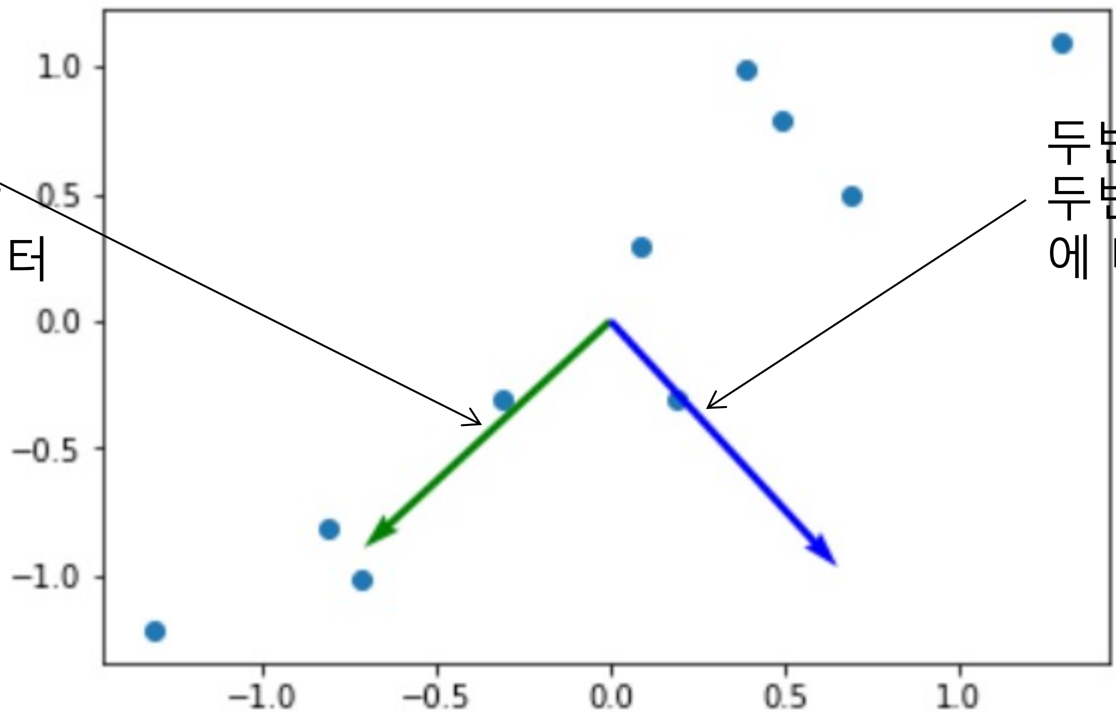
- 주의!
  - PCA에서는 각 독립변수의 원래 값을 사용하지 않고 mean centering 된 값을 사용한다.
  - 즉, 원래값에서 각 변수의 평균을 뺀 값을 사용한다.
    - 그래야지만 일반적인 경우, 모형의 성능이 더 좋다.

PCA

- Cov(X)의 고유값과 고유벡터 구하기
  - 전체 분산 = $Var(X_1) + \cdots + Var(X_K)$ = 고유값의 합
  - 고유값: 전체 분산을 설명하는 정도
  - 고유벡터
    - Principal component를 의미
      - 각 고유값의 방향 (분산의 방향)
    - array([[-0.73517866, -0.6778734],
      [0.6778734 , -0.73517866]])

PCA

첫번째 PC
가장 큰 고유값
에 대한 고유벡터



두번째 PC
두번째 큰 고유값
에 대한 고유벡터

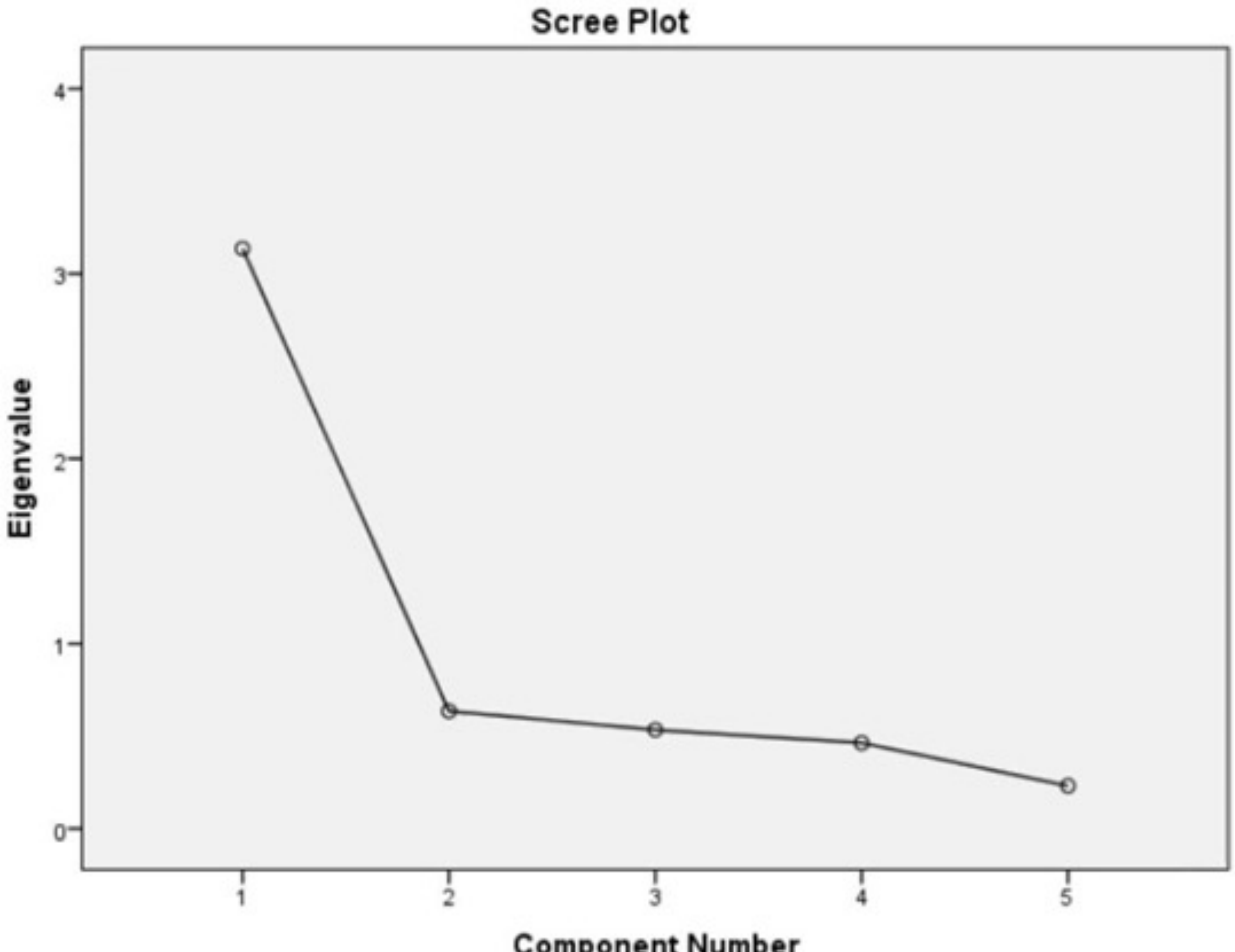
Review

- 순서
  - **원데이터의 각 독립변수에 대해서 mean centering 한다 (즉, 원래 값에서 해당 변수의 평균을 뺀다).**
  - **원데이터에 대한 공분산 행렬을 만든다.**
  - **공분산 행렬에 대해서 고유분해를 수행한다 (즉, 고유값과 고유벡터를 찾는다).**
  - **각 고유벡터가 우리가 찾고자 하는 PC가 된다.**
  - **우리는 이중에서 설명력이 높은 PC만을 선택한다.**
    - **몇개를 선택하는지는 사용자가 결정**
    - **이렇게 선택된 PC가 우리가 최종적으로 사용하고자 하는 독립변수가 됨 (즉, feature 가 됨)**
  - **새로 구한 PC에 대해 각 관측치의 새로운 값 구하기**

PCA

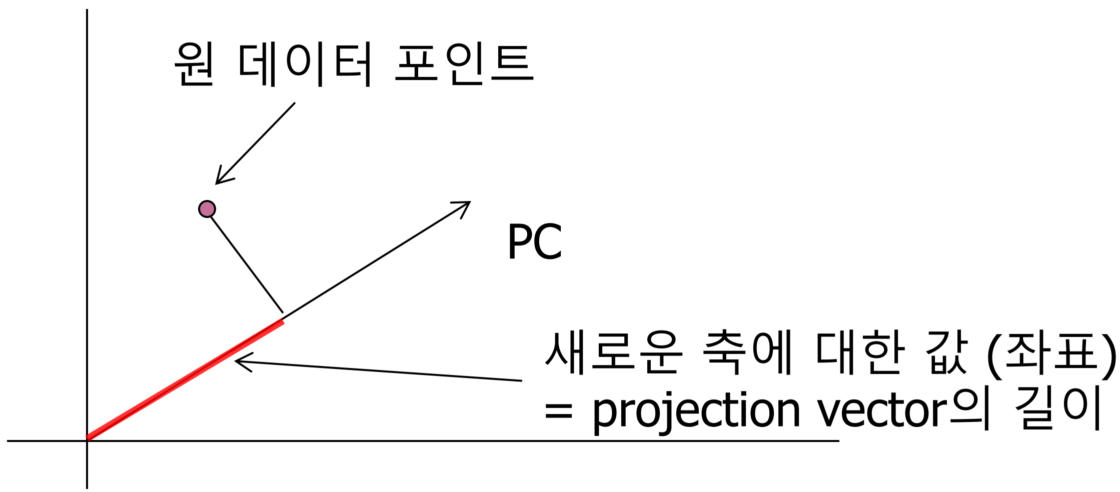
- 우리가 사용하고 싶은 component (eigenvector)만을 고른다.
  - 선택 방법
    - Scree plot
      - Eigenvalues를 사용
- PC를 이용한 데이터 표현
  - 기저의 이동 (change of basis)
    - 축의 이동

Scree plot



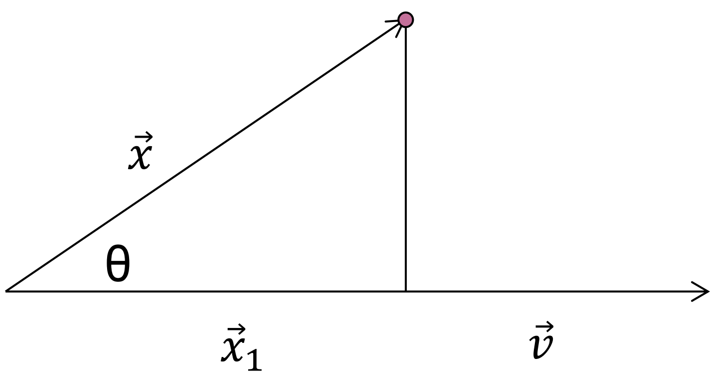
PC를 이용한 데이터 표현

- PC를 이용한 데이터 표현
  - 각 관측치가 각 PC에 대해서 갖는 값을 계산
  - PC로 나타내어지는 새로운 IV에 대한 값 계산으로 간주
  - 이는 원래의 관측치가 갖는 해당 PC로 나타내어지는 축에 대한 좌표 (즉, 길이)



PC를 이용한 데이터 표현

- PC를 이용한 데이터 표현
  - 이를 위해 projection vector에 대해서 먼저 알아야 한다.



$$\vec{x}_1 = |\vec{x}_1| \frac{\vec{v}}{|\vec{v}|}$$

즉, 길이가 $|\vec{x}_1|$ 면서 방향은
$\vec{v}$ 의 단위벡터와 동일한 벡터

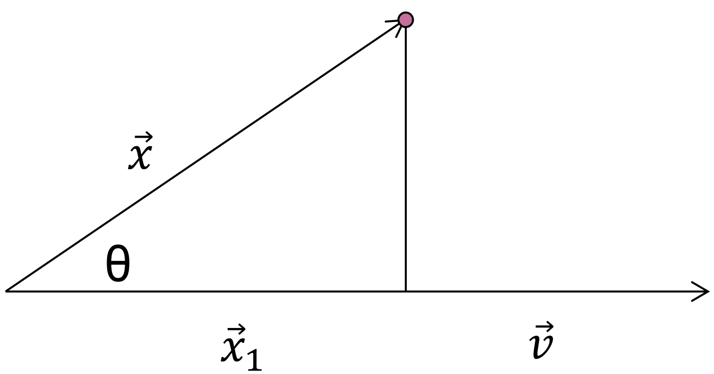
$$cos\theta = \frac{|\vec{x}_1|}{|\vec{x}|}$$

$$|\vec{x}_1| = |\vec{x}|cos\theta = |\vec{x}| \frac{\vec{x} \cdot \vec{v}}{|\vec{x}||\vec{v}|} = \frac{\vec{x} \cdot \vec{v}}{|\vec{v}|}$$

$$\because \vec{x} \cdot \vec{v} = |\vec{x}||\vec{v}|cos\theta$$

PC를 이용한 데이터 표현

- PC를 이용한 데이터 표현



$$|\vec{x}_1| = |\vec{x}|cos\theta = |\vec{x}|\frac{\vec{x}\cdot\vec{v}}{|\vec{x}||\vec{v}|} = \frac{\vec{x}\cdot\vec{v}}{|\vec{v}|}$$

$\vec{v}$ 가 고유벡터인 경우, $|\vec{v}| = 1$

따라서, $|\vec{x}_1| = \vec{x}\cdot\vec{v}$

PCA

- sklearn을 이용하는 경우

```
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
pca.fit_transform(d_np)
```

특이값분해 (Singular values decomposition)

Singular values decomposition

- 다른 종류의 decomposition
  - SVD (Singular values decomposition)
    - eigendecomposition과 비슷, but
    - eigendecomposition 는 square matrix에 대해서만 가능, 다른 형태의 matrix에 대해서는 SVD 사용
    - 다음과 같이 정의
    - $\mathrm{X} = \mathrm{UDV}^{\mathrm{T}}$
    - X는 정사각행렬이 아닌 사각행렬로 mxn 행렬라고 표현
    - U는 XX^{T} 행렬의 고유벡터를 열로 갖는 행렬
    - V는 $\mathrm{X}^{\mathrm{T}}\mathrm{X}$의 고유벡터를 열로 갖는 행렬
    - D은 대각행렬인데, 대각원소는 X^TX 혹은 XX^T의 eigenvalues (λ_i)에 루트를 씌운 값 $(\sqrt{\lambda_i})$, 이를 X의 singular values라고 함

Singular values decomposition

- SVD (cont'd)
  - 파이썬 코드
    - "SVD_example.ipynb" 참고
    - U, D, V = np.linalg.svd(A)
      - 여기서의 V는 위의 공식에서의 V^T

Q & A