Data Mining

차원 축소 (Dimension Reduction)

학습 목표

- 고차원 데이터를 저차원 데이터로 만드는 차원 축소에 대해서 살펴본다.

주요 내용

1. 차원 축소
2. 주성분 분석 (PCA)

1. 차원 축소

고차원 데이터 (High-Dimensional Data)

실생활의 데이터는 대부분 고차원 데이터로 이뤄져 있다.

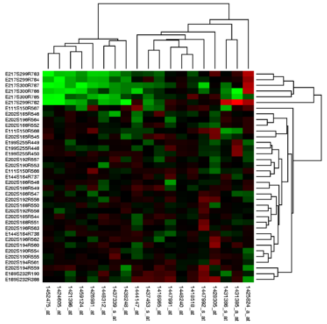
100만 차원의 이미지

1024x1024 이미지 = 1,048,576 차원

NETFLIX

Netflix 추천 시스템 (Recommendation System)

480,189 사용자 x 17,770 영화 행렬



유전자 발현 분석 군집화 (Clustering)

10,000 유전자 x 1,000 조건

고차원 데이터 (High-Dimensional Data)

GPT3 (Generative Pre-trained Transformer 3)

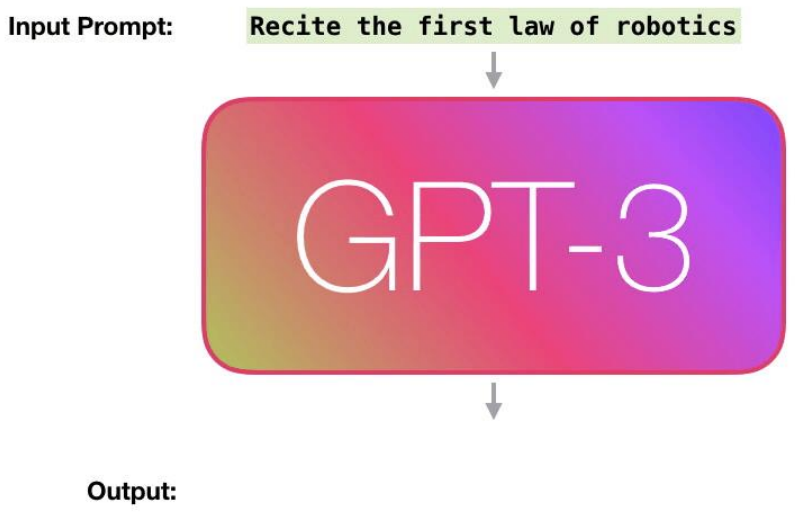


그림 발췌 : https://jalammar.github.io/

2020년 OpenAI가 제안한 최대 규모의 언어 모델
(Language Model)

1750억 파라미터 모델

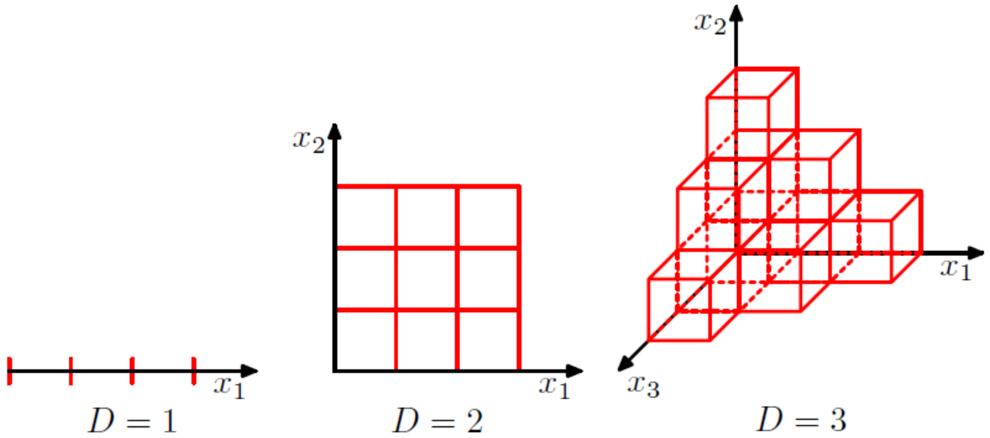
3천억 토큰으로 사전 학습

- **문서 분류** (Text classification)
- **질의 응답** (Question answering)
- **문서 생성** (Text generation)
- **문서 요약** (Text summarization)
- **개체 인식** (Named-entity recognition)
- **언어 번역** (Language translation)

50,257차원을 12,288차원으로 임베딩해서 훈련

차원의 저주 (Curse of dimensionality)

차원이 증가할수록 상황을 설명하는데 필요한 인스턴스가 기하급수적으로 증가하는 현상



각 차원 별로 등간격의 격자를 만들었을 때 기하 급수적으로 증가

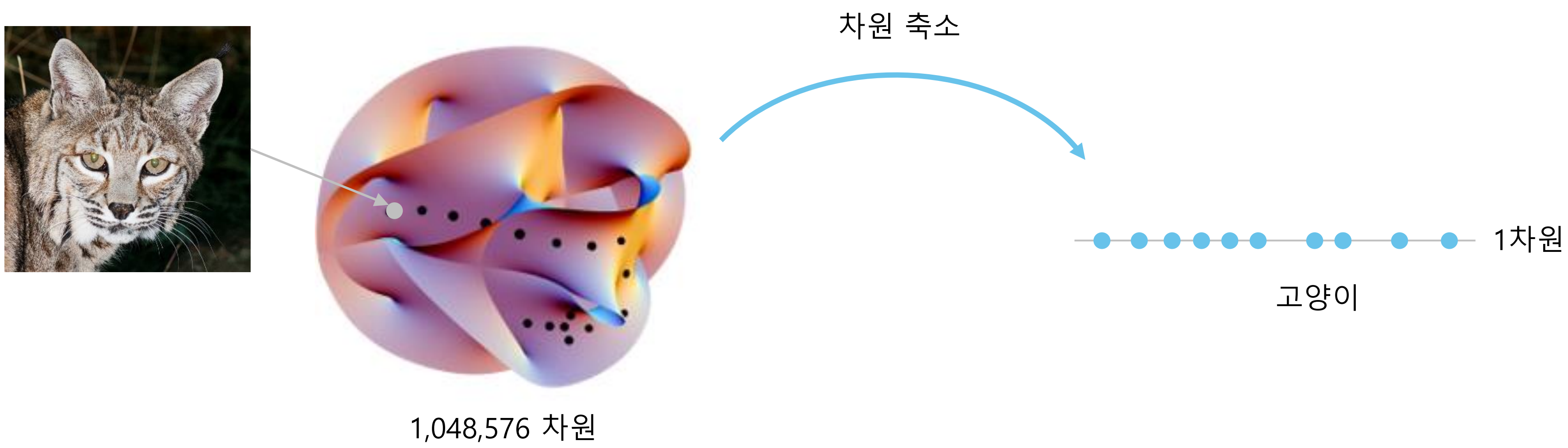
차원이 높아질수록 데이터 사이의 거리가 멀어지고 빈공간이 증가하면서 데이터가 희소해지는 현상이 생김

차원 축소 (Dimensionality Reduction)

대부분의 데이터는 고차원 공간에 있지만 저차원의 잠재 데이터로 표현될 수 있다.

고차원 공간 (Observed Data Space)

저차원 잠재 공간 (Latent Space)



핵심적인 정보는 유지하면서 불필요하거나 중복되는 정보를 제거하는 과정

차원 축소 (Dimensionality Reduction)

특징 추출 (Feature Extraction)

비지도 학습

선형 사영 (Linear Projection)

- 데이터를 분산을 최대화하는 차원을 찾는 방식
- 주성분 분석 (Principal Component Analysis)

비선형 사영 (Nonlinear Projection)

- 고차원 공간에서 이웃의 관계가 저차원에서도 유지되도록 비선형 맵핑을 하는 방식
- 시각화에 많이 사용
- MDS (Multidimensional Scaling), LLE, ISOMAP, t-SNE

오토인코더 (AutoEncoder)

- 딥러닝 이용한 비선형 차원 축소 방식
- 잠재 공간을 학습할 때 사용
- 오토인코더 (AutoEncoder), VAE (Variational AutoEncoder)

특징 선택 (Feature Selection)

지도 학습

성능을 보고 성능이 최대가 되는 방법을 찾음. ⇒ 지도학습

레퍼 방법 (Wrapper Method)

- RFE (Recursive feature elimination), SFS (Sequential feature selection)
- 유전 알고리즘 (Genetic algorithm)
- 단변량 특징 선택 (Univariate selection), Exhaustive
- mRMR (Minimum Redundancy Maximum Relevance)

필터 방법 (Filter Method)

- 상관 계수 (Correlation coefficient), 정보 획득 (information gain)
- 카이스퀘어 테스트 (Chi-square test), 피셔 스코어 (Fisher score), 분산 (variance) 임계치

임베드 방식 (Embedded Method)

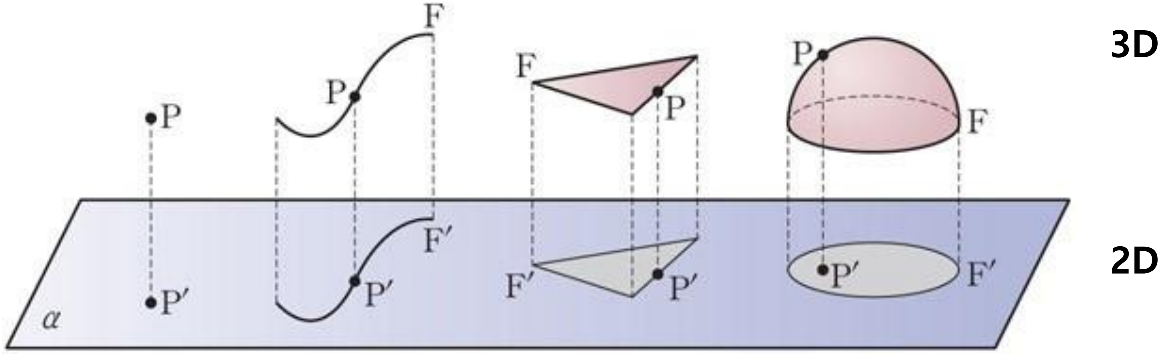
- 라소 회귀 (Lasso Regression), 리지 회귀 (Ridge Regression)
- 의사결정 트리 (Decision Tree)

선형 사영 vs 비선형 사영

사영(Projection)**이란 집합을 부분집합으로 맵핑하는 과정**

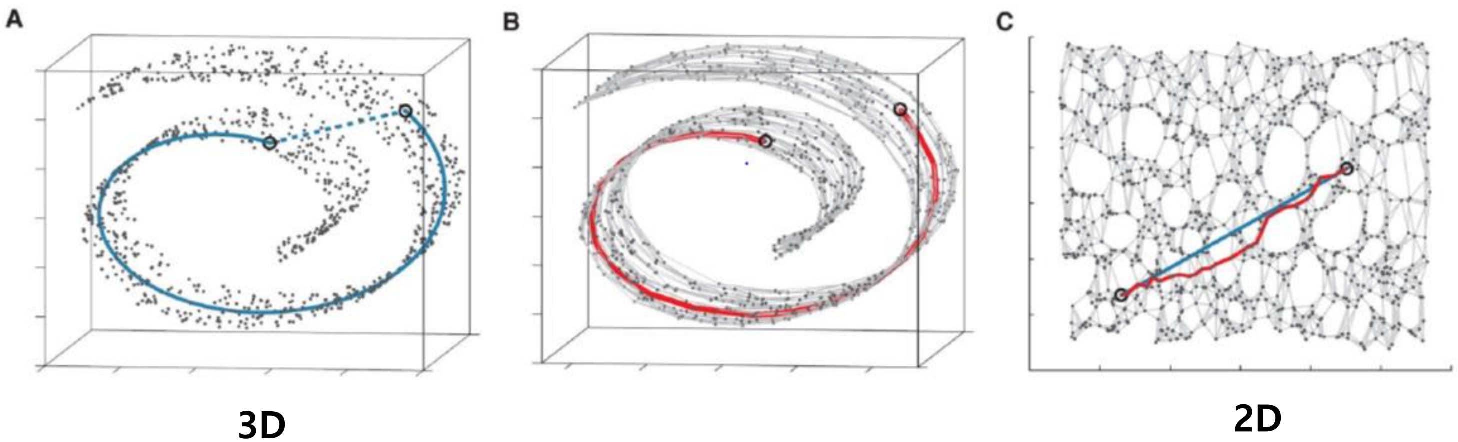
선형 사영 (Linear Projection)

- 선형 맵핑을 통해 저차원의 특징을 추출하는 방식
- 정사영 (orthographic projection) : 도형의 각 점에서 투영할 평면에 수선을 내려서 만들어지는 도형



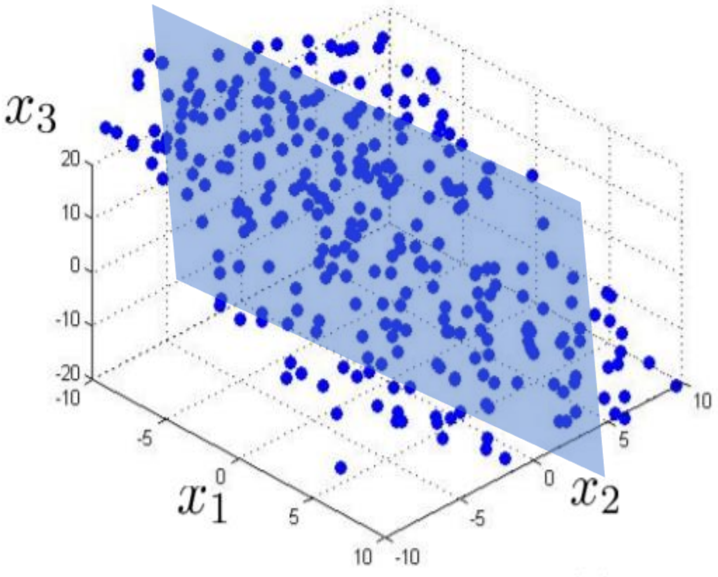
비선형 사영 (Nonlinear Projection)

- 비선형 맵핑을 통해 저차원의 특징을 추출하는 방식

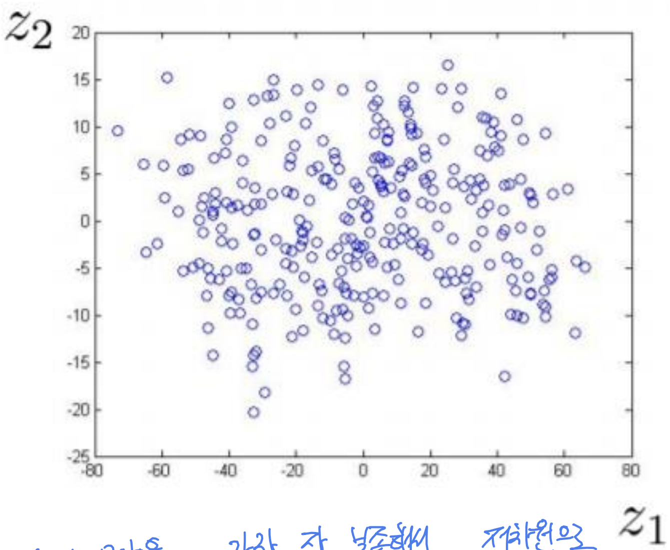


주성분 분석 (Principal Component Analysis)

정사영 방식으로 데이터의 분산을 가장 크게 표현할 수 있는 차원을 찾는 방식



3D

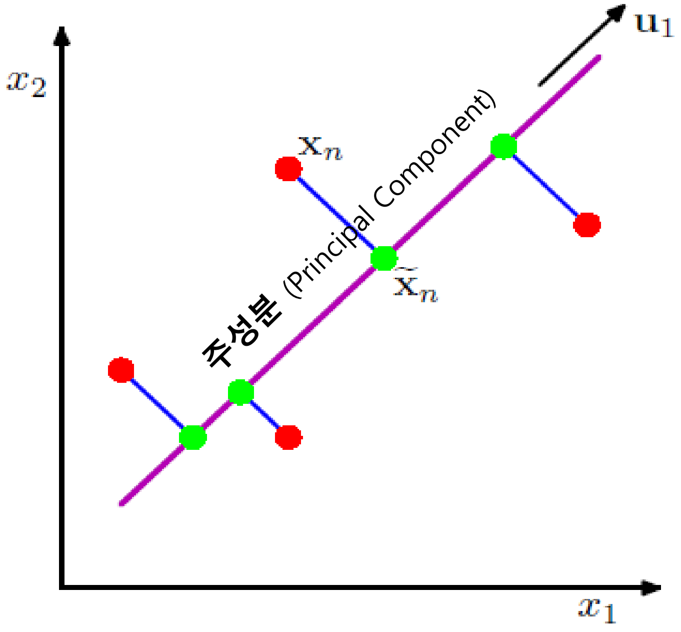


원래 모양을 가장 잘 보존해서 저차원으로
투영할려면 골고루 퍼진 상태인 차원을 찾아야한다.

2D

주성분 분석 (Principal Component Analysis)

데이터의 주성분은 공분산 행렬의 가장 큰 고윳값에 해당하는 고유 벡터이다.



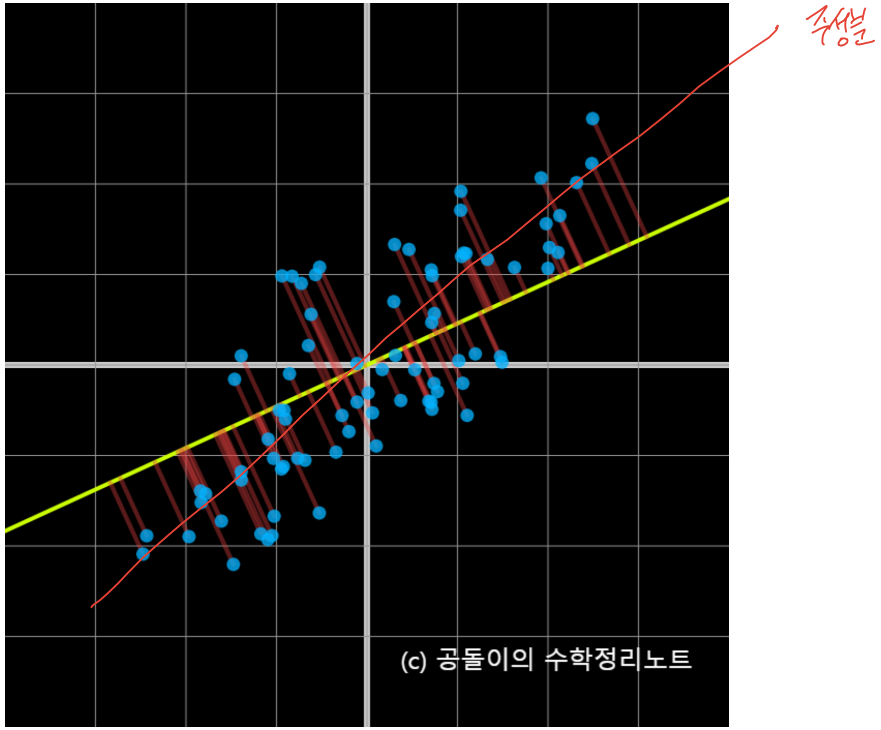
주성분을 찾는 두 가지 관점

1. 데이터의 분산을 가장 크게 만드는 차원을 찾자.
2. 데이터와의 오차를 최소화하는 차원을 찾자.

두 관점 중 어떤 관점에서 출발하더라도
데이터의 **공분산 행렬의 고유 벡터**가 주성분이 됨

주성분 분석 (Principal Component Analysis)

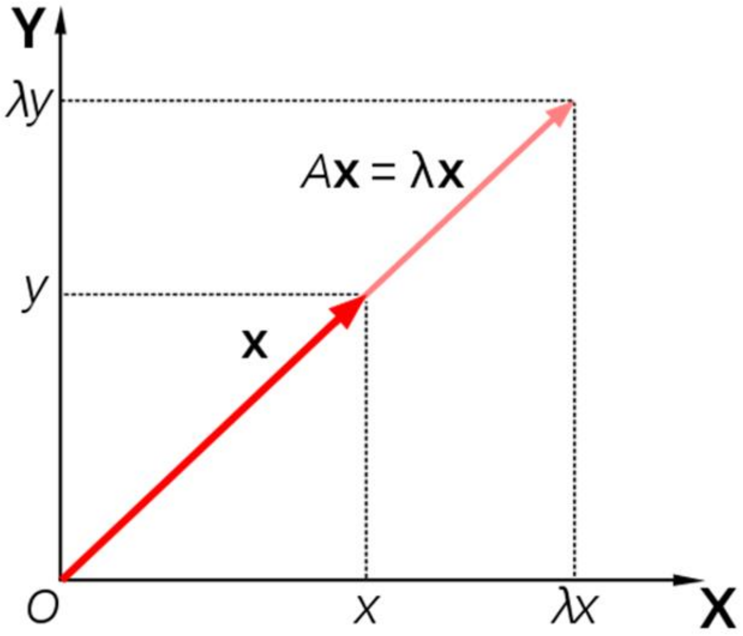
정사영 방식으로 데이터의 분산을 가장 크게 표현할 수 있는 차원을 찾는 방식



https://angeloyeo.github.io/2019/07/27/PCA.html

고윳값과 고유벡터

데이터를 선형변환 후 크기는 바뀌지만 방향이 바뀌지 않는 벡터가 고유벡터

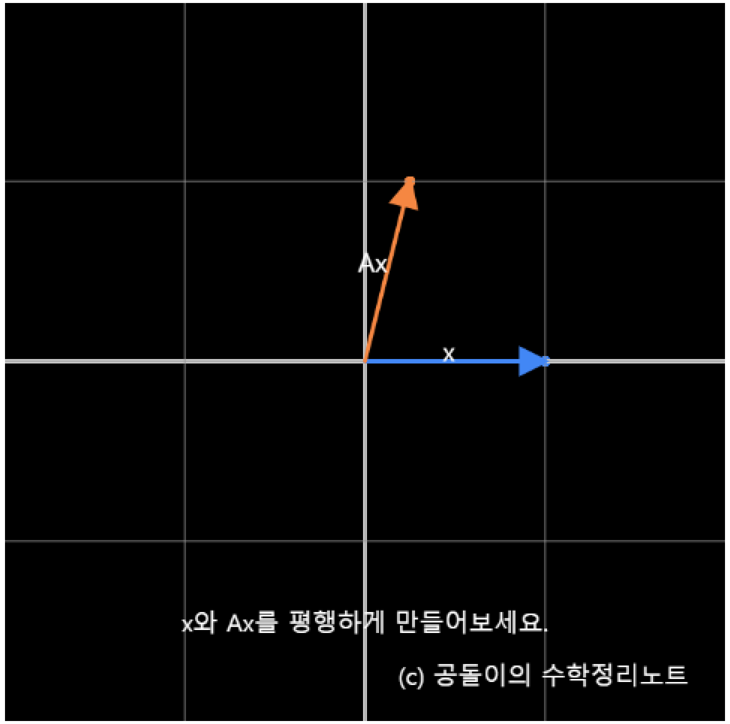


$$Ax = \lambda x$$

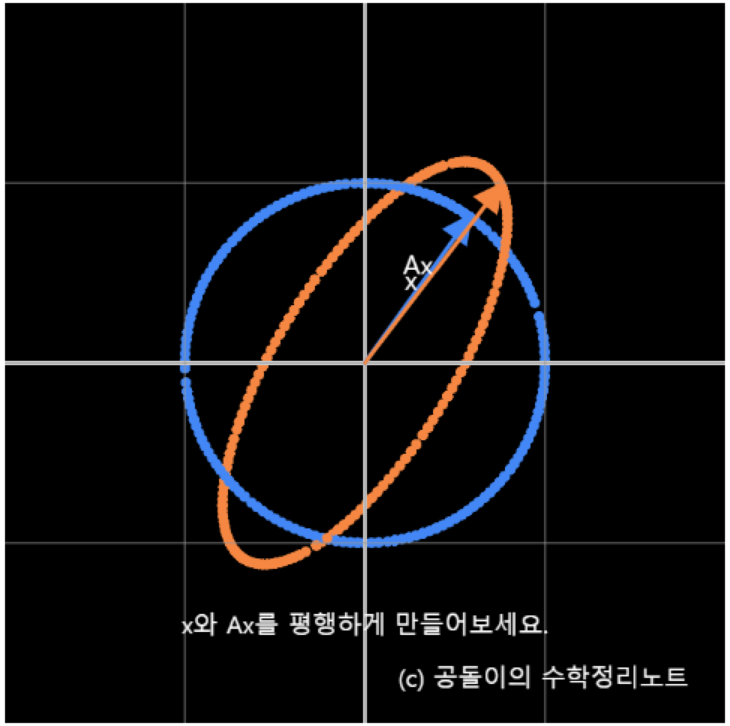
선형변환 고윳값 고유벡터

고윳값과 고유벡터

x 벡터를 회전했을 때 x 벡터와 Ax 벡터가 일치하는 방향이 고유벡터이다.



- 파란색 벡터 : x, 주황색 벡터 : Ax



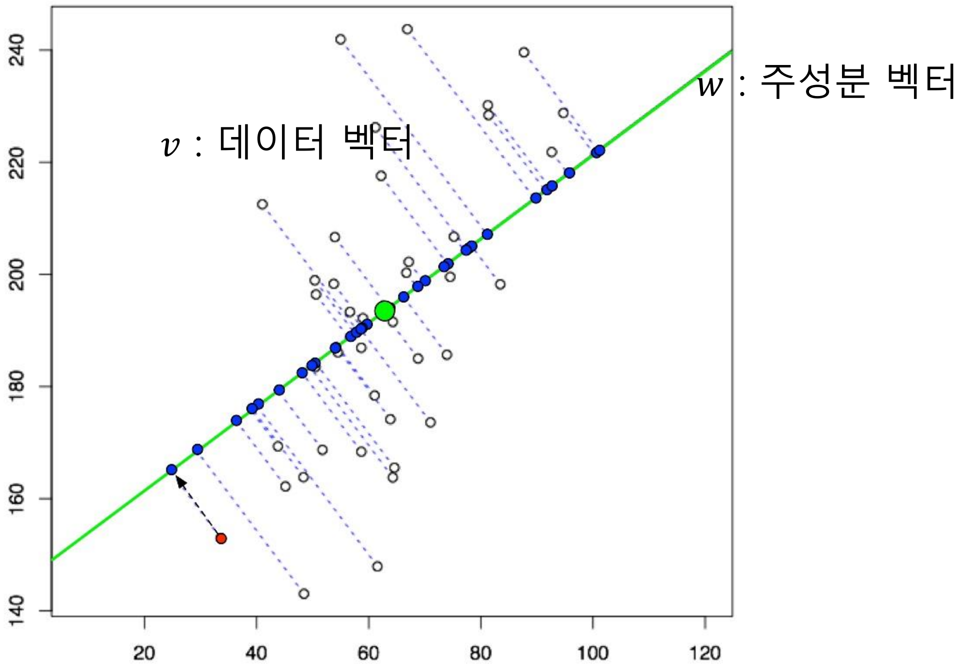
- 두 벡터가 평행이 되는 방향이 2개 존재

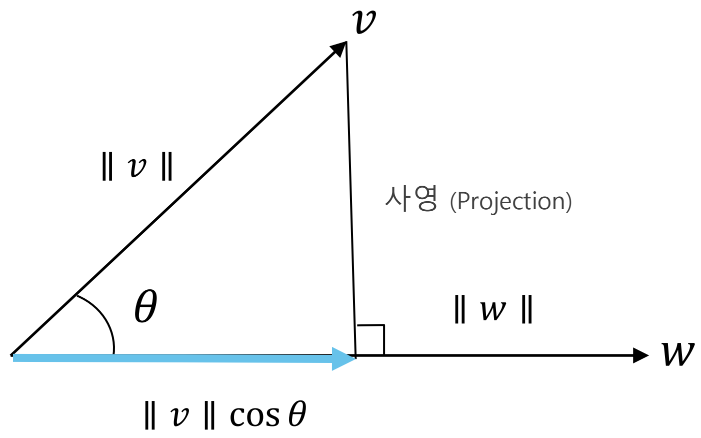
https://angeloyeo.github.io/2019/07/17/eigen_vector.html

2. 주성분 분석 (PCA)

주성분 분석 알고리즘

데이터를 사영했을 때 분산이 최대화되는 방향 벡터를 찾아보자!





사영 길이 $\dfrac{v \cdot w}{\| w \|} = \| v \| \cos\theta$ 방향 : $\dfrac{w}{\| w \|}$

$\|w\|=1$ 이라는 가정하에

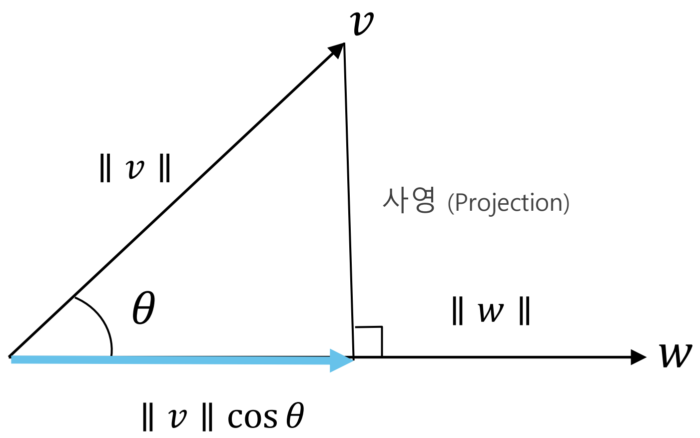
내적

w가 단위 벡터이면 사영 길이는 내적 $v \cdot w$이 됨

단위 벡터 $\boldsymbol{w}$방향으로 v를 사영해서 분산을 측정하자!

(v를 원점으로 이동한 후 $\boldsymbol{w}$와 내적하고, 이 값의 제곱의 합이 분산이 됨)

참고 내적 (dot product)



내적 : $v \cdot w = \| v \| \| w \| \cos\theta$

$= \underbrace{\| v \| \cos\theta}_{\text{사영}} \| w \|$

- 내적은 사영에 비례하기 때문에 사영 대신 내적을 사용하기도 함

내적 (Dot product)

- 두 벡터 v와 w가 이루는 각도가 θ일 때

$v \cdot w = \| v \| \| w \| \cos\theta$ 를 내적이라고 함

$$v \cdot w = \sum_{i=0}^{n} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_n w_n$$

사영 (Projection)

$$\text{사영} = \frac{v \cdot w}{\| w \|} = \| v \| \cos\theta$$

주성분 분석 알고리즘

1 **평균을 원점으로 이동** $\boldsymbol{v}_k = \boldsymbol{v}_k - \overline{\boldsymbol{v}}$ $\quad \overline{\boldsymbol{v}} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{v}_k$

관측 데이터 $\mathcal{D} = \{(\boldsymbol{v}_k) : k = 1 \ldots N\}$

2 **분산 $\hat{\sigma}^2$을 최대화 하는 방향 성분 w를 찾기**

목적 함수

$$\hat{\boldsymbol{\sigma}}^2 = \sum_{k=1}^{N} (\boldsymbol{v}_k \cdot \widehat{\boldsymbol{w}})^2$$

directional variance

w : 주성분 벡터 $\quad \widehat{w} = \frac{w}{\|w\|}$ 단위 벡터

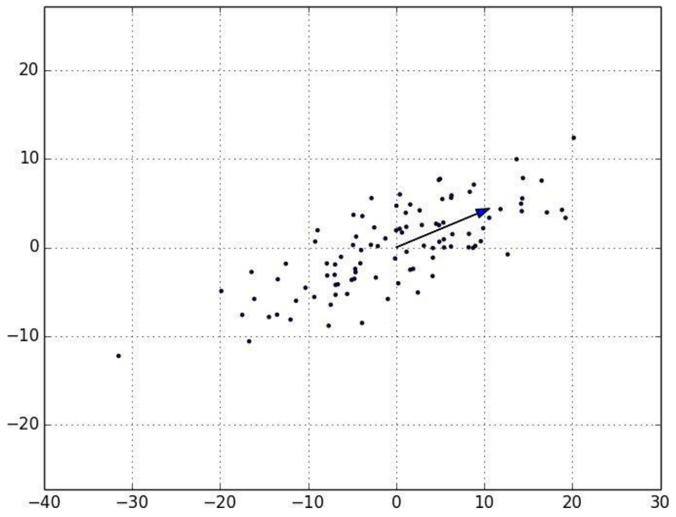
최적화할때에는 $\frac{1}{N}$은 있어도되고 없어도됨.

PCA 최적화 문제 :

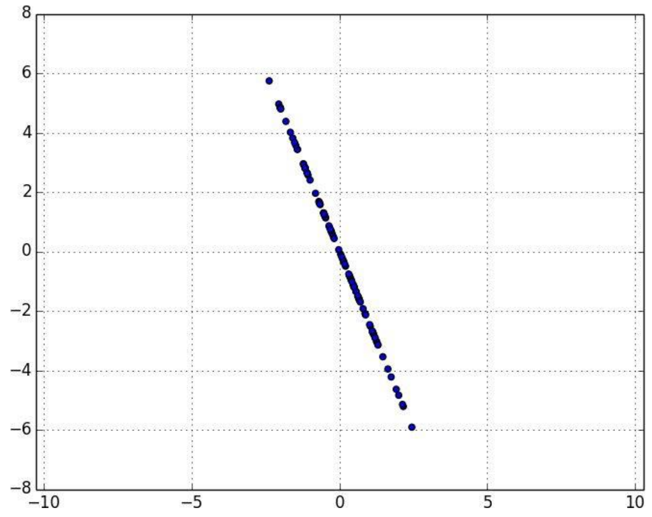
$$\max_{w} \sum_{k=1}^{N} (\boldsymbol{v}_k \cdot \widehat{\boldsymbol{w}})^2$$

주성분 분석 알고리즘

3 **주성분을 크기 순서대로 순차적으로 구해 나감**



주성분 제거



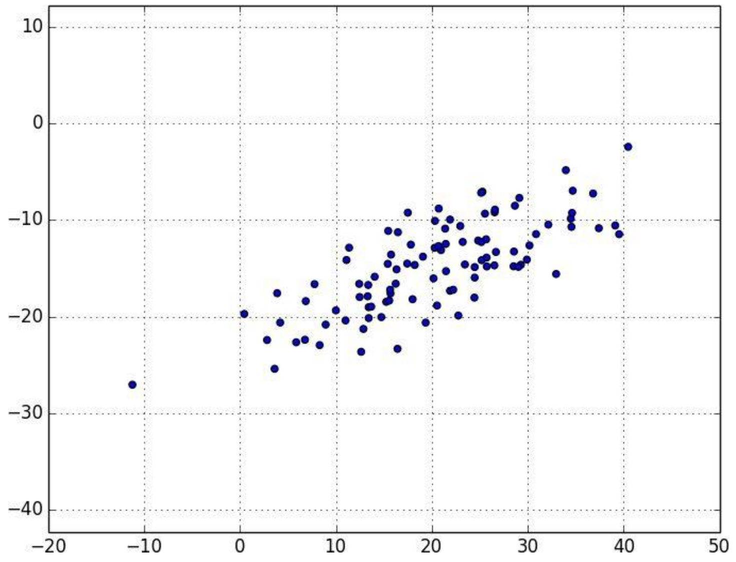
- 해당 주성분 방향으로 데이터를 사영한 후 뺀다.

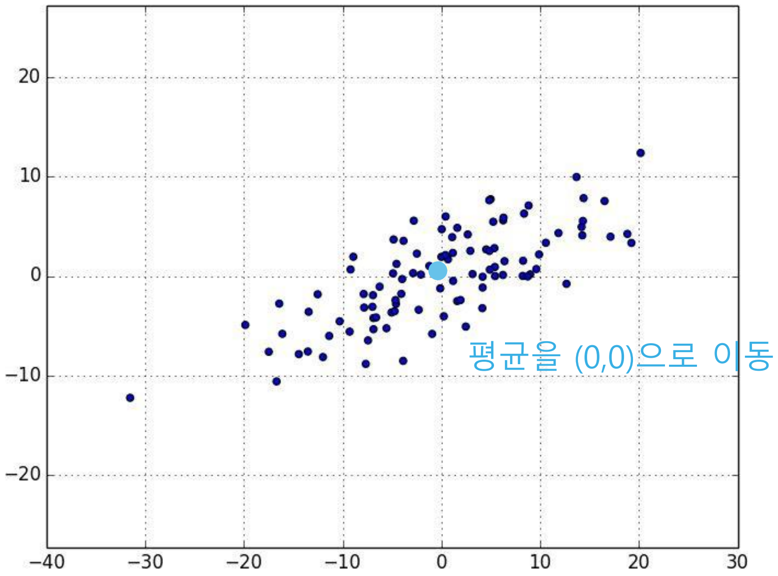
4 **차원 축소 실행**

- 구한 주성분 방향으로 데이터를 사영한다.

1단계 원점으로 이동

평균을 원점으로 이동





모든 데이터에서 평균을 빼기

```
from scratch.linear_algebra import subtract

def de_mean(data: List[Vector]) -> List[Vector]:
    """Recenters the data to have mean 0 in every dimension"""
    mean = vector_mean(data)
    return [subtract(vector, mean) for vector in data]
```

2단계 목적 함수

w를 단위 벡터 $\hat{w}$로 변환 (길이가 1인 벡터는 방향 성분만 남음)

```
from scratch.linear_algebra import magnitude

def direction(w: Vector) -> Vector:
    mag = magnitude(w)
    return [w_i / mag for w_i in w]
```

단위 벡터

$$\hat{w} = \frac{w}{\|w\|}$$

단위 벡터 $\hat{w}$ 방향으로 투영했을 때의 분산

```
from scratch.linear_algebra import dot

def directional_variance(data: List[Vector], w: Vector) -> float:
    """
    Returns the variance of x in the direction of w
    """
    w_dir = direction(w)
    return sum(dot(v, w_dir) ** 2 for v in data)
```

directional variance

$$\hat{\boldsymbol{\sigma}}^2 = \sum_{k=1}^{N} (\boldsymbol{v}_k \cdot \hat{\boldsymbol{w}})^2$$

- **단위 벡터 $\hat{w}$ 방향으로 투영했을 때 길이는 $\frac{v \cdot w}{\|w\|}$ 이며 $\| w \| = 1$ 이므로 내적이 길이가 됨**
- **데이터 별 투영 길이의 제곱의 합**

3단계 경사 상승법 (Gradient Ascent)

목적 함수

$$\hat{\boldsymbol{\sigma}}^2 = \sum_{k=1}^{N} (\boldsymbol{v}_k \cdot \widehat{\boldsymbol{w}})^2$$

$v_{k1} \cdot \hat{w}_1 + v_{k2} \cdot \hat{w}_2 \cdot + \cdots + v_{ki} \cdot \hat{w}_i + \cdots + v_{kD} \cdot \hat{w}_D$

그래디언트

목적함수

$$\frac{\partial \hat{\boldsymbol{\sigma}}^2}{\partial \boldsymbol{w}} = \left[\frac{\partial \hat{\boldsymbol{\sigma}}^2}{\partial w_1}, \frac{\partial \hat{\boldsymbol{\sigma}}^2}{\partial w_2}, \ldots, \frac{\partial \hat{\boldsymbol{\sigma}}^2}{\partial w_i}, \ldots, \frac{\partial \hat{\boldsymbol{\sigma}}^2}{\partial w_D}\right]$$

각성분에대한 벡터

$$\frac{\partial \hat{\boldsymbol{\sigma}}^2}{\partial w_i} = \sum_{k=1}^{N} 2(\boldsymbol{v}_k \cdot \widehat{\boldsymbol{w}}) v_{ki}$$

$$i = 1, 2, \ldots, D$$

단위 벡터 $\hat{w}$ 방향으로 투영했을 때의 분산의 그래디언트

```python
def directional_variance_gradient(data: List[Vector], w: Vector) -> Vector:
    """
    The gradient of directional variance with respect to w
    """
    w_dir = direction(w)
    return [sum(2 * dot(v, w_dir) * v[i] for v in data)
            for i in range(len(w))]
```

3단계 경사 상승법 (Gradient Ascent)

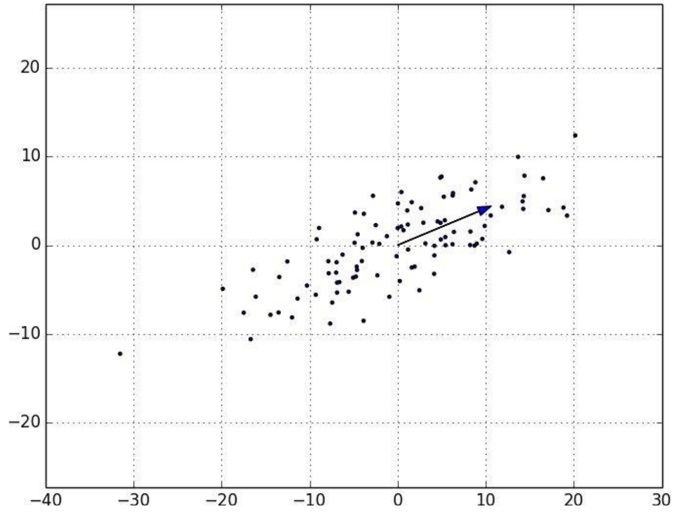
목적 함수를 최대화 하기 위해 경사 상승법(Gradient Ascent)를 수행

주성분 찾기

```
from scratch.gradient_descent import gradient_step

def first_principal_component(data: List[Vector],
                              n: int = 100,
                              step_size: float = 0.1) -> Vector:
    # Start with a random guess
    guess = [1.0 for _ in data[0]]

    with tqdm.trange(n) as t:
        for _ in t:
            dv = directional_variance(data, guess)
            gradient = directional_variance_gradient(data, guess)
            guess = gradient_step(guess, gradient, step_size)
            t.set_description(f"dv: {dv:.3f}")

    return direction(guess)
```

W

목적



$\hat{\omega} = \frac{w}{\|w\|}$

stepsize

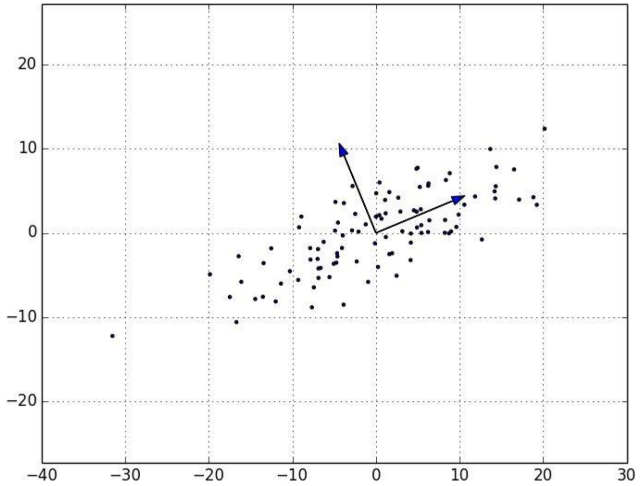
gradien.

$\theta^+ = \theta + \alpha \frac{\partial J}{\partial \theta}$

- **guess** : 찾고자 하는 주성분 벡터 ***w***
- **directional_variance** : 목적 함수 (유틸리티 함수)
- step_size가 양수이므로 **Gradient Ascent**가 실행됨
- 단위 벡터 $\hat{w}$로 만들어서 반환

4단계 주성분 구하기

주성분을 크기 순서대로 순차적으로 구해 나감



Num_components 개수만큼 주성분 찾기

```
def pca(data: List[Vector], num_components: int) -> List[Vector]:
    components: List[Vector] = []
    for _ in range(num_components):
        component = first_principal_component(data)
        components.append(component)
        data = remove_projection(data, component)

    return components
```

→ 데이터셋에서 주성분 빼는 함수.

- 찾은 주성분 component는 단위 벡터

4단계 주성분 구하기

주성분을 제거하기 위해 주성분 방향으로 데이터를 투영한 후 뺌

벡터 w에 투영

```
from scratch.linear_algebra import scalar_multiply

def project(v: Vector, w: Vector) -> Vector:
    """return the projection of v onto the direction w"""
    projection_length = dot(v, w)
    return scalar_multiply(projection_length, w)
```

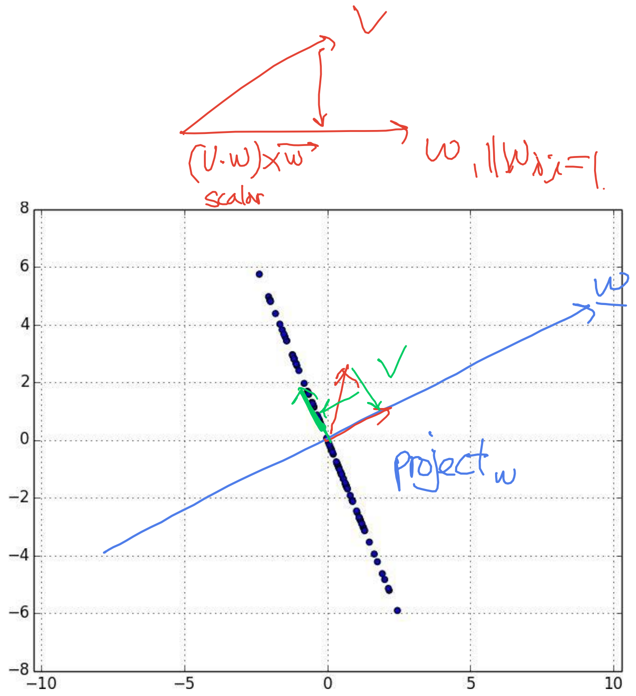
길이에 방향성분 곱해서 벡터로 만들어줌

- w가 단위 벡터이므로 내적을 했을 때는 사영의 길이가 나오고
  w를 곱해주면 사영된 벡터가 구해 짐

데이터에서 주성분 제거

```
from scratch.linear_algebra import subtract

def remove_projection_from_vector(v: Vector, w: Vector) -> Vector:
    """projects v onto w and subtracts the result from v"""
    return subtract(v, project(v, w))

def remove_projection(data: List[Vector], w: Vector) -> List[Vector]:
    return [remove_projection_from_vector(v, w) for v in data]
```

- w는 단위 벡터



2D → 1D

파란색선을 빼버리면 위에 점들 처럼 직선의 점들의 모양이 된다.

5단계 차원 축소

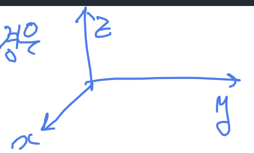
벡터를 각 성분에 대해 투영

```python
def transform_vector(v: Vector, components: List[Vector]) -> Vector:
    return [dot(v, w) for w in components]
```

방향 차원에 대한 basis

$\{x_p, y_p, z_p\}$

ex) 3차원 일 경우



전체 데이터에 대해 저차원 공간으로 변환

```python
def transform(data: List[Vector], components: List[Vector]) -> List[Vector]:
    return [transform_vector(v, components) for v in data]
```

Thank you!