

# 데이터마이닝

## 1. 평균 (Mean):

- 데이터 세트의 모든 값의 합을 그 값의 개수로 나눈 값입니다. 데이터 세트 내 모든 값을 대표하는 중심 경향성의 척도입니다.
- 예: 데이터가 1, 2, 3, 4, 5 인 경우, 평균은  $(1+2+3+4+5)/5 = 3$  입니다.

## 2. 중앙값 (Median):

- 데이터를 크기 순으로 나열했을 때 중앙에 위치하는 값입니다. 데이터의 개수가 홀수면 중앙의 값이, 짝수면 중앙 두 값의 평균이 중앙값이 됩니다.
- 예: 1, 2, 3, 4, 5 의 중앙값은 3, 1, 2, 3, 4 의 중앙값은  $(2+3)/2 = 2.5$  입니다.

## 3. 중앙점 (Medoid):

- 클러스터링에서 사용되는 개념으로, 클러스터 내의 모든 점들과의 거리가 가장 작은 데이터 포인트를 의미합니다. 중앙값과 비슷하지만 고차원 데이터에 사용됩니다.
- 예: 다차원 공간에서 점들 사이의 거리를 최소화하는 점이 중앙점입니다.

## 4. 중심점 (Centroid):

- 주로 지리학이나 클러스터링에서 사용되며, 주어진 점들의 평균 위치를 나타냅니다. 각 차원의 좌표 평균을 구하여 계산합니다.
- 예: 점 (1,1), (2,2), (3,3) 의 중심점은 (2,2) 입니다.

## 5. 최빈값 (Mode):

- 데이터 세트에서 가장 빈번하게 나타나는 값입니다. 데이터의 분포를 파악할 때 유용하며, 한 개 이상의 최빈값이 있을 수 있습니다.
  - 예: 1, 2, 2, 3, 4 에서 최빈값은 2 입니다.
-

# Dot production (내적)

내적 (Dot Product) 은 두 벡터 간의 연산으로, 두 벡터의 크기와 사이의 각도를 이용하여 계산됩니다. 수학적으로는 두 벡터의 대응하는 성분들을 곱한 후 그 결과를 모두 더하는 연산입니다. 내적은 물리학, 컴퓨터 그래픽, 기계 학습 등 다양한 분야에서 응용됩니다.

## 내적의 계산

두 벡터  $\mathbf{a} = (a_1, a_2, \dots, a_n)$  과  $\mathbf{b} = (b_1, b_2, \dots, b_n)$  의 내적은 다음과 같이 정의됩니다:

$$\mathbf{a} \cdot \mathbf{b} = a_1 \times b_1 + a_2 \times b_2 + \dots + a_n \times b_n$$

이 값은 물리적으로는 두 벡터 사이의 " 투영 " 된 성분의 곱의 합으로 해석할 수 있습니다.

## 내적의 기하학적 의미

내적은 두 벡터의 각도와 관련이 있습니다. 내적을 이용하여 두 벡터 사이의 각도  $\theta$  를 다음 공식을 통해 구할 수 있습니다:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos(\theta)$$

여기서  $\|\mathbf{a}\|$  와  $\|\mathbf{b}\|$  는 각각 벡터  $\mathbf{a}$  와  $\mathbf{b}$  의 크기 (놈) 입니다.

## 내적의 응용

- 물리학: 힘과 변위의 내적을 통해 일을 계산합니다.
- 컴퓨터 그래픽: 빛의 방향과 면의 방향 사이의 내적을 통해 해당 면이 받는 빛의 양을 계산합니다.
- 기계 학습: 벡터 간의 유사성을 측정하거나, SVM(Support Vector Machine) 같은 알고리즘에서 결정 경계를 찾는 데 사용됩니다.

내적은 이처럼 다양한 분야에서 중요한 수학적 도구로 활용됩니다.

---

# Gradient Descent (경사 하강법)

경사 하강법은 주어진 함수의 지역 최소값을 찾기 위해 사용됩니다. 이 방법은 비용 함수 (Cost Function) 또는 손실 함수 (Loss Function) 의 최소값을 찾는 데 주로 사용되며, 기계 학습과 딥러닝에서 파라미터를 최적화하는 데 활용됩니다.

## 작동 원리

- 임의의 시작점에서 함수의 그라디언트 (기울기) 를 계산합니다.
- 그라디언트의 반대 방향으로 일정 크기 (학습률) 만큼 이동합니다.
- 새 위치에서 다시 그라디언트를 계산하고 반복합니다.
- 그라디언트가 충분히 작아질 때까지 이동을 반복합니다.

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \cdot \nabla J(\theta_{\text{old}})$$

- $\theta$ : 최적화하고자 하는 파라미터입니다.
- $\alpha$ : 학습률 (learning rate) 로, 스텝의 크기를 결정합니다. 즉, 한 번에 얼마나 크게 파라미터를 업데이트할지를 결정하는 값입니다.
- $\nabla J(\theta)$ : 비용 함수  $J$  의 그라디언트입니다. 이 그라디언트는  $\theta$  에서의 함수의 기울기를 나타내며, 함수의 최소값을 찾기 위해 이동해야 할 방향과 크기를 제공합니다.

경사 하강법은 이 수식을 반복적으로 적용하면서  $\theta$  를 업데이트하여, 비용 함수  $J(\theta)$  의 값을 점점 줄여나가 최소값에 도달하도록 합니다. 이 과정은 그라디언트  $\nabla J(\theta)$  가 충분히 작아질 때까지, 즉 변화가 미미할 때까지 계속됩니다.

---

기호  $\nabla$  (나블라) 는 벡터 미적분학에서 주로 사용되며, '그라디언트 (gradient)', '발산 (divergence)', '회전 (curl)' 과 같은 벡터 연산을 나타내는데 사용됩니다. 이 기호의 용도는 주어진 문맥에 따라 달라질 수 있습니다.

## 그라디언트 (Gradient)

- $\nabla f$  형태로 사용될 때, 스칼라 함수  $f$  의 그라디언트를 나타냅니다.
- 그라디언트는 함수의 각 변수에 대한 편미분을 성분으로 갖는 벡터입니다.
- $f$  가  $x, y, z$  의 함수인 경우,  $\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$  입니다.
- 이 벡터는  $f$  가 가장 빠르게 증가하는 방향과 크기를 나타냅니다.

## 발산 (Divergence)

- $\nabla \cdot \mathbf{F}$  형태로 사용될 때, 벡터장  $\mathbf{F}$  의 발산을 나타냅니다.
- 발산은 벡터장 내에서 벡터의 '원천' 혹은 '싱크' 의 정도를 나타내는 스칼라 양입니다.
- $\mathbf{F} = (P, Q, R)$  인 경우,  $\nabla \cdot \mathbf{F} = \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}$  입니다.

## 회전 (Curl)

- $\nabla \times \mathbf{F}$  형태로 사용될 때, 벡터장  $\mathbf{F}$  의 회전을 나타냅니다.
- 회전은 벡터장의 회전하는 경향을 나타내는 벡터입니다.
- $\mathbf{F} = (P, Q, R)$  인 경우,  $\nabla \times \mathbf{F} = \left( \frac{\partial R}{\partial y} - \frac{\partial Q}{\partial z}, \frac{\partial P}{\partial z} - \frac{\partial R}{\partial x}, \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right)$  입니다.

각각의 연산은 벡터 미적분학에서 중요한 역할을 하며, 물리학, 공학, 컴퓨터 그래픽스 등 다양한 분야에서 활용됩니다.

---

데이터를 분류하는 데 사용되는 네 가지 주요 척도인 **명목척도 (Nominal)**, **서열척도 (Ordinal)**, **간격척도 (Interval)**, **비율척도 (Ratio)**에 대해 설명드리겠습니다. 이러한 척도들은 데이터의 특성과 적절한 분석 방법을 결정하는 데 중요합니다.

## 1. 명목척도 (Nominal Scale)

- **특징:** 데이터를 상호 배타적인 범주로 분류합니다. 숫자는 단순히 라벨로 사용되며, 숫자 사이에는 수학적 의미가 없습니다.
- **예시:** 성별 (남자, 여자), 혈액형 (A, B, AB, O), 차량 번호판 등.
- **연산:** 데이터의 동일성과 차이를 비교할 수 있습니다. 평균이나 표준편차를 계산할 수 없습니다.

## 2. 서열척도 (Ordinal Scale)

- **특징:** 범주형 데이터를 순서대로 배열할 수 있습니다. 각 범주 사이의 '간격'은 일정하지 않을 수 있습니다.
- **예시:** 교육 수준 (초등학교, 중학교, 고등학교, 대학교), 만족도 조사 (매우 불만족, 불만족, 보통, 만족, 매우 만족) 등.
- **연산:** 데이터를 순서대로 정렬할 수 있지만, 범주 간의 실제 거리를 측정할 수는 없습니다.

## 3. 간격척도 (Interval Scale)

- **특징:** 숫자로 표현되며, 각 값 사이의 간격이 동일합니다. 절대적인 '영점'은 존재하지 않습니다.
- **예시:** 온도 (섭씨, 화씨), 연도 등.
- **연산:** 더하기와 빼기 연산이 가능하지만, 곱하기나 나누기는 의미가 없습니다. 평균과 표준편차를 계산할 수 있습니다.

## 4. 비율척도 (Ratio Scale)

- **특징:** 간격척도의 모든 특성을 가지며, 추가적으로 절대적인 '영점'을 가집니다. 이로 인해 모든 수학적 연산이 가능합니다.
- **예시:** 무게, 길이, 시간, 속도 등.
- **연산:** 더하기, 빼기, 곱하기, 나누기 모두 가능합니다. 평균, 표준편차, 비율 계산 등이 가능합니다.

이러한 척도들은 데이터를 측정하고 분석하는 방법을 이해하는 데 중요하며, 각각의 척도는 적절한 통계적 기법과 데이터 처리 방법을 결정하는 데 영향을 미칩니다.

---

주어진 함수  $f(x_1, x_2)$  는 두 변수  $x_1$  과  $x_2$  에 대한 식으로, 다음과 같이 표현됩니다:

$$f(x_1, x_2) = (2x_1 - 1)^2 + (3x_2 - 1)^3$$

이 식에서:

- $(2x_1 - 1)^2$  는  $x_1$  에 대한 식이며,  $x_1$  의 값에 따라 그 제곱 형태로 값을 변화시킵니다.
- $(3x_2 - 1)^3$  는  $x_2$  에 대한 식이며,  $x_2$  의 값에 따라 그 세제곱 형태로 값을 변화시킵니다.

함수는  $x_1$  과  $x_2$  의 조합에 따라 서로 다른 출력 값을 생성하며, 이를 통해 2 차원 공간에서의 함수 표면을 형성합니다.

$\nabla f(1, 1)$  벡터의 모든 원소의 합

함수  $f(x_1, x_2) = (2x_1 - 1)^2 + (3x_2 - 1)^3$  의 그라디언트  $\nabla f$  를 계산하고, 특정 점  $(1, 1)$  에서의 그라디언트 값을 구한 후, 그 원소들의 합을 계산하겠습니다.

먼저, 각 변수에 대한 편미분을 계산해야 합니다:

1.  $x_1$  에 대한 편미분:

$$\frac{\partial}{\partial x_1} ((2x_1 - 1)^2) = 2 \times 2(2x_1 - 1) = 4(2x_1 - 1)$$

2.  $x_2$  에 대한 편미분:

$$\frac{\partial}{\partial x_2} ((3x_2 - 1)^3) = 3 \times (3x_2 - 1)^2 \times 3 = 9(3x_2 - 1)^2$$

이제 이 편미분들을 각각 계산한 값으로  $(1, 1)$  에 대입하겠습니다. 이렇게 하여 얻은 각 값의 합을 계산합니다.

이제 Python 을 사용하여 실제 계산을 수행하겠습니다.

점  $(1, 1)$  에서 함수  $f(x_1, x_2)$  의 그라디언트 벡터의 원소들은 다음과 같습니다:

- $x_1$  에 대한 편미분 값은 4 입니다.
  - $x_2$  에 대한 편미분 값은 36 입니다.
-

**Interquartile Range (IQR, 사분위 범위)**는 통계에서 데이터의 분포를 측정하는 척도 중 하나로, 중앙값을 중심으로 데이터의 중간 50% 범위를 나타냅니다. IQR 은 제 3 사분위수 (Q3) 에서 제 1 사분위수 (Q1) 를 빼서 계산합니다. 즉,  $IQR = Q3 - Q1$  입니다. 이 값은 데이터의 중간 범위에 얼마나 많은 변동성이 있는지를 보여줍니다.

## IQR 의 계산 방법

1. 데이터 정렬: 데이터를 크기 순으로 정렬합니다.
2. 사분위수 계산:
  - 제 1 사분위수 (Q1): 데이터 하위 25% 의 값에 해당합니다.
  - 제 3 사분위수 (Q3): 데이터 상위 25% 의 값에 해당합니다.
3. IQR 계산: Q3 에서 Q1 을 빼서 계산합니다.

## IQR 의 중요성

- 이상치 탐지: IQR 은 이상치 (outliers) 를 탐지하는 데 사용됩니다. 일반적으로,  $Q1 - 1.5 IQR$  만 또는  $Q3 + 1.5 IQR$  초과하는 값들을 이상치로 간주합니다.
- 데이터 분포 이해: IQR 은 데이터 집합의 중간 범위 내 분포의 집중도와 퍼짐 정도를 파악하는 데 도움을 줍니다. 데이터의 중심 주위 얼마나 넓게 퍼져 있는지를 보여주며, 극단값의 영향을 덜 받습니다.

IQR 은 특히 왜곡된 (skewed) 데이터나 이상치가 있는 경우 중심 경향성과 퍼짐을 이해하는 데 유용한 도구입니다.

먼저, 주어진 데이터 세트  $[500, 3, 2, 4, -100]$  의 IQR(사분위 범위) 를 계산하기 위해서는 데이터를 크기 순으로 정렬하고, 각 사분위수를 계산해야 합니다.

1. 데이터 정렬:

정렬된 데이터 :  $-100, 2, 3, 4, 500$

2. 사분위수 계산:

- 제 1 사분위수 (Q1): 데이터의 하위 25% 위치입니다. 데이터가 5 개이므로, 하위 25% 는 첫 번째 데이터 뒤에 위치합니다. 중간값은 3 이고, 그 하위 중간값을 계산하면 2 가 됩니다.
- 제 3 사분위수 (Q3): 데이터의 상위 25% 위치입니다. 상위 중간값을 계산하면 4 가 됩니다.

3. IQR 계산:

$$IQR = Q3 - Q1 = 4 - 2 = 2$$

따라서, 이 데이터 세트의 IQR 은 2 입니다. 이 값은 데이터 중간 50% 의 범위가 2 만큼 퍼져 있다는 것을 의미하며, 이 데이터의 분포를 이해하는 데 도움이 됩니다. 데이터의 극단값들 (-100, 500) 은 IQR 계산에 직접적인 영향을 주지 않음을 확인할 수 있습니다.

---

```
print((10)%(3))
print((10)%(-3))
print((-10)%(3))
print((-10)%(-3))
```

나머지 연산 (모듈로 연산) 에서  $(-10) \bmod (-3)$  의 결과가  $-1$  인 이유를 설명하겠습니다.

모듈로 연산은 주어진 숫자를 나눗셈 후의 나머지를 반환하는 연산입니다. 이 연산은 주로 양수에 대해 이야기되지만, 음수에도 적용될 수 있습니다.

일반적으로  $a \bmod b$  는 다음과 같은 수식으로 계산할 수 있습니다:

$$a \bmod b = a - b \times \left\lfloor \frac{a}{b} \right\rfloor$$

여기서  $\lfloor x \rfloor$  는  $x$  를 넘지 않는 가장 큰 정수, 즉  $x$  의 바닥 함수입니다.

## 계산 과정

1.  $\frac{-10}{-3} = 3.333\dots$  이며, 여기서  $\lfloor 3.333\dots \rfloor = 3$  입니다.
2. 따라서,  $(-10) - (-3) \times 3 = -10 - (-9) = -10 + 9 = -1$

따라서  $(-10) \bmod (-3)$  의 결과는  $-1$  이 됩니다.

이 결과가 나오는 이유는 모듈로 연산의 정의에 따라, 나머지는 항상 나누는 수 (모듈로) 의 부호를 따르지 않고, 피제수 ( $a$ , 여기서는  $-10$ ) 의 부호를 따르기 때문입니다. 이러한 계산은 음수를 다룰 때 주의를 요하며, 나눗셈의 결과를 어떻게 처리할지에 대한 정의에 따라 달라질 수 있습니다.

---



```
import numpy as np
print(np.array([[[[1],[1]]]]).shape)

arr=np.array(np.arange(12).reshape(3,4))

print(arr[:2,2:])
# [[ 2 3] [10 11]]

L = np.array([[1,2]])
print(L + L.T)
# [[2 3] [3 4]]
```

```
import numpy as np
print(np.array([[[[1],[1]]]]).shape)

arr=np.array(np.arange(12).reshape(3,4))
print(arr[:2,2:])
```

```
import numpy as np

f = lambda x,y: np.dot(x,y)/np.sqrt((np.dot(x,x) * np.dot(y,y)))
x = [1,2,4,2,3,1,1,10]
y = [3,6,12,6,9,3,3,30]

print(np.dot(x,x))
print(np.dot(y,y))
print(np.sqrt((np.dot(x,x) * np.dot(y,y))))
print(np.dot(x,y))
print(f(x,y))
```

주어진 Python 람다 함수 `f = lambda x, y: np.dot(x, y) / np.sqrt((np.dot(x, x) * np.dot(y, y)))` 는 두 벡터  $x$  와  $y$  사이의 코사인 유사도 (Cosine Similarity) 를 계산하는 수식입니다. 이 함수는 두 벡터의 내적과 각 벡터의 크기를 사용하여 계산됩니다.

## 수식의 구성 요소 설명

1. `np.dot(x, y)` : 두 벡터  $x$  와  $y$  의 내적 (dot product) 을 계산합니다. 내적은  $x$  와  $y$  의 각 성분의 곱의 합으로, 벡터의 방향성이 얼마나 유사한지를 수치로 나타냅니다.
2. `np.dot(x, x)` 와 `np.dot(y, y)` :
  - `np.dot(x, x)` 는 벡터  $x$  의 길이 (또는 크기) 의 제곱을 계산합니다.
  - `np.dot(y, y)` 는 벡터  $y$  의 길이의 제곱을 계산합니다.
3. `np.sqrt((np.dot(x, x) * np.dot(y, y)))` :
  - `np.dot(x, x) * np.dot(y, y)` 는 두 벡터의 길이의 제곱을 곱한 값입니다.
  - `np.sqrt()` 는 이 값을 제곱근으로 취해줍니다, 즉 두 벡터 길이의 곱입니다.

## 코사인 유사도 계산

코사인 유사도는 두 벡터 사이의 각도의 코사인 값으로, 두 벡터의 방향성이 얼마나 유사한지를 나타내는 척도입니다. 코사인 유사도는 다음 공식으로 정의됩니다:

$$\begin{aligned}\text{cosine similarity} &= \frac{\text{dot product of } x \text{ and } y}{\text{product of the magnitudes of } x \text{ and } y} \\ &= \frac{\text{np.dot}(x, y)}{\sqrt{\text{np.dot}(x, x) \times \text{np.dot}(y, y)}}\end{aligned}$$

따라서 이 함수는 두 벡터  $x$  와  $y$  사이의 코사인 유사도를 계산하여, 두 벡터가 얼마나 비슷한 방향을 가리키고 있는지 수치적으로 나타내줍니다. 이 값은  $-1$  에서  $1$  사이의 값을 가집니다:

- $1$  은 두 벡터가 완전히 같은 방향을 가리키는 경우,
  - $0$  은 두 벡터가 직교 (서로에게 수직) 하는 경우,
  - $-1$  은 두 벡터가 정반대 방향을 가리키는 경우를 의미합니다.
-

```
g = lambda x, y: np.dot(x - np.mean(x), y - np.mean(y)) /
np.linalg.norm(x - np.mean(x)) / np.linalg.norm(y - np.mean(y))

print(g(np.array([1,2,3,4,5,6]), np.array([6,5,4,3,2,1])))
```

주어진 Python 람다 함수 `g = lambda x, y: np.dot(x - np.mean(x), y - np.mean(y)) / np.linalg.norm(x - np.mean(x)) / np.linalg.norm(y - np.mean(y))` 는 두 벡터  $x$  와  $y$  사이의 피어슨 상관계수 (Pearson correlation coefficient) 를 계산하는 수식입니다. 이 수식은 두 벡터의 표준화된 값들의 내적을 사용하여 그들 사이의 선형 상관성을 측정합니다.

## 수식의 각 부분 설명

### 1. $x - \text{np.mean}(x)$ 와 $y - \text{np.mean}(y)$ :

- `np.mean(x)` 와 `np.mean(y)` 는 각각 벡터  $x$  와  $y$  의 평균값입니다.
- $x - \text{np.mean}(x)$  와  $y - \text{np.mean}(y)$  는 각 벡터에서 해당 벡터의 평균값을 뺀 것으로, 벡터의 중심을 원점으로 이동시킨 "중심화된 벡터" 입니다.

### 2. `np.dot(x - np.mean(x), y - np.mean(y))` :

- 이는 중심화된 두 벡터의 내적으로, 두 벡터 사이의 공분산을 계산합니다.

### 3. `np.linalg.norm(x - np.mean(x))` 와 `np.linalg.norm(y - np.mean(y))` :

- `np.linalg.norm()` 는 벡터의 유클리드 길이 (또는 놈) 를 계산합니다.
- 이는 각각 중심화된 벡터  $x$  와  $y$  의 표준 편차와 관련이 있습니다.

## 피어슨 상관계수 계산

피어슨 상관계수는 두 변수 사이의 선형 관계의 강도와 방향을 측정합니다. 계산 공식은 다음과 같습니다:

$$\begin{aligned} \text{Pearson correlation coefficient} &= \frac{\text{covariance of } x \text{ and } y}{\text{standard deviation of } x \times \text{standard deviation of } y} \\ &= \frac{\text{np.dot}(x - \text{np.mean}(x), y - \text{np.mean}(y))}{\text{np.linalg.norm}(x - \text{np.mean}(x)) \times \text{np.linalg.norm}(y - \text{np.mean}(y))} \end{aligned}$$

이 값은 -1 에서 1 사이를 가지며:

- 1 은 완벽한 양의 선형 관계,
- 0 은 선형 관계가 전혀 없음,
- -1 은 완벽한 음의 선형 관계를 의미합니다.

위의 함수는 두 데이터 세트가 얼마나 잘 일치하는지를 수치적으로 나타내주는 데 사용됩니다.

```
p = np.array([1,2,3])
u = p[:, np.newaxis] ; print(u) # [[1] [2] [3]]
v = p.T[np.newaxis, :] ; print(v) # [[1 2 3]]
print(np.sum(np.abs(u - v)))
```

---

```
from functools import reduce

d = [[1,2,3,4], [1,2,4], [1,2], [1,5], [1,4]]
is1 = set(reduce(lambda x, y: x + y, d)) # {1, 2, 3, 4, 5}
il = list(is1) # [1, 2, 3, 4, 5]

is2 = [(i, j) for i in il for j in is1 if i < j]
# [(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5),
(4, 5)]

t = [sum([set(i) <= set(b) for b in d]) for i in is2]
# [3, 1, 3, 1, 1, 2, 0, 1, 0, 0]

s = map(lambda x: x >= 0.6 * len(d), t)

r = [is2[index] for index, selected in enumerate(s) if selected]
# [(1, 2), (1, 4)]

print(np.sum(np.array(r))) # 8
```

---

# Gradient Descent 알고리즘에서 Overshooting 이나 Local minimum 에 빠지는 문제를 해결하기 위해서 어떻게 해야 하는지 설명하시오

overshooting - learning rate 를 조율하여 발산하지 않도록 하거나, learning rate decay 기법을 이용하여 학습률을 점진적으로 줄여나가도록 해야 합니다.

local minimum - stochastic, mini batch gradient descent 기법을 활용했을 때 발생하는 무작위성을 이용하여 local minimum 에 빠지지 않도록 도움을 줄 수 있습니다. 다만, 무조건 global minimum 에 도달하는 것을 보장하지는 않습니다.

Gradient Descent 알고리즘에서 겪을 수 있는 일반적인 문제들 중 하나는 Overshooting(과도하게 지나치는 현상) 과 Local Minimum(지역 최소값) 에 빠지는 현상입니다. 이러한 문제들을 해결하기 위한 몇 가지 전략을 소개하겠습니다.

## 1. 학습률 조절 (Learning Rate Adjustment)

- **Overshooting**은 주로 학습률 (learning rate) 이 너무 클 때 발생합니다. 학습률이 너무 크면 파라미터 업데이트 시 너무 큰 단계를 밟아 최소값을 지나치게 되고, 이는 알고리즘의 발산으로 이어질 수 있습니다.
- **해결책:** 학습률을 점진적으로 감소시키는 학습률 감소 기법 (learning rate decay) 을 사용하거나, 학습 과정에서 학습률을 동적으로 조정할 수 있는 기법 (예: Adam, RMSprop 등) 을 사용합니다.

## 2. 모멘텀 사용 (Using Momentum)

- 모멘텀 기법은 이전 스텝의 업데이트를 현재 스텝에 일정 비율로 반영하여, 지역 최소값에서 빠져나오는 데 도움을 줄 수 있습니다.
- **해결책:** 모멘텀은 이전 그라디언트 업데이트를 고려하여 파라미터를 업데이트함으로써 알고리즘이 골짜기의 작은 요동에 갇히는 것을 방지하고, 보다 안정적으로 글로벌 최소값에 접근할 수 있도록 돕습니다.

## 3. 두 번째 순서 최적화 방법 사용 (Using Second-Order Optimization Methods)

- 두 번째 순서 최적화 방법은 헤시안 행렬 (Hessian Matrix) 을 이용하여 더 정확한 스텝 크기와 방향을 결정합니다.
- **해결책:** 뉴턴 방법과 같은 두 번째 순서 방법은 그라디언트만 사용할 때보다 더 정교한 스텝 크기 조정을 가능하게 하여, 지역 최소값 문제를 완화할 수 있습니다. 그러나 계산 비용이 높다는 단점이 있습니다.

## 4. 다양한 초기값 사용 (Using Different Initial Values)

- 지역 최소값에 빠지는 문제는 초기값에 따라 다르게 나타날 수 있습니다.

- **해결책:** 여러 다른 초기값에서 경사 하강법을 여러 번 실행하여, 각기 다른 지점에서 시작된 탐색이 서로 다른 최소값에 도달하는지 관찰합니다. 이를 통해 더 좋은 글로벌 최소값에 접근할 수 있습니다.

## 5. 스토캐스틱 및 미니 배치 경사 하강법 (Stochastic & Mini-batch Gradient Descent)

- 스토캐스틱 경사 하강법 (SGD) 은 매 업데이트에서 전체 데이터셋 대신 하나의 샘플을, 미니 배치 경사 하강법은 소량의 샘플을 사용합니다. 이는 노이즈가 있는 그라디언트를 생성하며, 이 노이즈가 알고리즘을 지역 최소값에서 벗어나게 할 수 있습니다.
- **해결책:** 노이즈가 있는 업데이트는 알고리즘이 지역 최소값에서

쉽게 벗어날 수 있도록 도와줍니다. 또한, 미니 배치 방법은 계산 효율성도 높입니다.

이러한 전략들을 조합하거나 적절히 선택하여 사용하면 Gradient Descent 알고리즘의 성능을 크게 향상시킬 수 있으며, Overshooting 이나 Local Minimum 문제를 효과적으로 완화할 수 있습니다.

---

# 경사 하강법 종류

## 1. Batch Gradient Descent

배치 경사 하강법은 가장 전통적인 형태의 경사 하강법으로, 파라미터 업데이트를 위해 전체 데이터 세트를 사용합니다. 이 방법은 전체 트레이닝 데이터 셋을 사용하여 한 번의 반복 (iteration) 동안 파라미터를 업데이트합니다. 각 반복에서 전체 데이터셋에 대해 그라디언트를 계산하므로, 계산 비용이 높고 시간이 오래 걸릴 수 있지만, 더 안정적인 그라디언트 추정과 부드러운 수렴을 제공합니다.

### 예시

선형 회귀 모델에서 비용 함수  $J(\theta)$  를 최소화하기 위해 사용:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

여기서  $m$  은 트레이닝 샘플 수입니다. 반복마다 전체 데이터셋에 대한 그라디언트를 계산하여 파라미터  $\theta$  를 업데이트:

$$\theta = \theta - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

여기서  $\alpha$  는 학습률입니다.

## 2. Stochastic Gradient Descent

스토캐스틱 경사 하강법은 매 업데이트에서 데이터셋의 하나의 샘플만을 무작위로 선택하여 그라디언트를 계산하고 파라미터를 업데이트합니다. 이 방식은 빠르고 효율적이며, 큰 데이터셋에 적합합니다. 그러나 그라디언트의 추정이 불안정할 수 있어, 수렴 과정이 불안정할 수 있습니다.

### 예시

선형 회귀에서 각 반복에서 하나의 트레이닝 샘플  $(x^{(i)}, y^{(i)})$  을 무작위로 선택하고, 그라디언트를 계산하여  $\theta$  를 업데이트:

$$\theta = \theta - \alpha (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

## 3. Mini-batch Gradient Descent

미니 배치 경사 하강법은 배치와 스토캐스틱 경사 하강법의 장점을 절충한 방법입니다. 각 반복에서 전체 데이터셋의 소수의 샘플, 즉 미니 배치를 사용하여 그라디언트를 계산합니다. 이 방법은 계산 효율성을 높이면서도 상대적으로 안정적인 수렴을 기대할 수 있습니다.

### 예시

선형 회귀에서 각 반복에서 50 개의 데이터 샘플을 선택하여 그라디언트를 계산하고  $\theta$  를 업데이트:



$$\theta = \theta - \alpha \frac{1}{50} \sum_{i=1}^{50} (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

각 방법의 선택은 트레이닝 데이터의 크기, 요구되는 수렴 속도, 계산 자원 등 여러 요소를 고려하여 결정됩니다.

---

# Association rule(연관규칙) mining 은 history data로부터 causation(인과관계)를 추론하는 기술이다. 어떻게 인과관계를 추출하는지 설명하시오

Association rule mining (연관 규칙 마이닝) 은 history data로부터 유용한 패턴, 연관성, 구조를 발견하는 데이터 마이닝의 한 방법입니다. 하지만, 연관 규칙 마이닝이 인과관계 (causation)를 추론하는 기술이라는 설명은 정확하지 않습니다. 실제로 연관 규칙 마이닝은 인과관계보다는 상관관계 (correlation)나 아이템 간의 연관성을 발견하는 데 초점을 맞춥니다. 이 방법은 특정 아이템 셋이 함께 발생할 가능성이 높은 패턴을 찾아내지만, 이러한 패턴이 왜 발생하는지, 즉 인과 관계를 설명하지는 않습니다.

## 연관 규칙 마이닝의 기본 개념

연관 규칙 마이닝은 대표적으로 시장 바구니 분석 (market basket analysis)에서 자주 사용됩니다. 이 분석은 고객이 특정 상품을 구매할 때 다른 상품도 함께 구매하는 경향이 있는지를 파악하는 데 사용됩니다.

### 규칙의 구성

연관 규칙은 일반적으로 "If  $X$ , then  $Y$ " 형태로 표현됩니다. 여기서  $X$ 와  $Y$ 는 상품 집합이며, 이 규칙은  $X$ 아이템 집합을 구매했을 때  $Y$ 아이템 집합을 구매할 확률이 높음을 의미합니다.

### 주요 메트릭

- 지지도 (Support):** 전체 거래 중  $X$ 와  $Y$ 가 함께 발생하는 거래의 비율입니다.
- 신뢰도 (Confidence):**  $X$ 가 포함된 거래 중  $Y$ 도 포함하는 거래의 비율입니다.
- 향상도 (Lift):**  $Y$ 의 구매가  $X$ 에 의존하는 정도를 나타내며,  $Y$ 의 구매가  $X$ 의 존재와 독립적일 경우에 비해 얼마나 증가하는지를 측정합니다.

## 인과관계와의 차이

연관 규칙 마이닝에서 추출된 규칙은 단순히 통계적인 관계를 나타내며,  $X$ 의 발생이  $Y$ 의 발생을 유발한다는 인과 관계를 의미하지 않습니다. 예를 들어, "맥주와 기저귀"가 함께 구매되는 유명한 예시는 기저귀를 사러 온 젊은 부모가 맥주도 함께 구매하는 경향이 있다는 상관관계를 보여줍니다. 하지만 이것이 맥주 구매가 기저귀 구매를 유발하거나 그 반대의 인과 관계가 있다고 말할 수는 없습니다.

## 결론

연관 규칙 마이닝은 데이터 내 숨겨진 패턴과 관계를 찾아내는 유용한 도구입니다. 하지만, 이러한 관계가 실제 인과관계인지는 추가적인 실험적 또는 종단적 연구를 통해 확인해야 합니다. 연관 규칙은 상관관계를 나타내며, 이러한 상관관계가 항상 인과관계를 의미하지는 않습니다.

여기 주어진 수학 및 통계 용어들에 대한 간단한 정의와 설명을 제공하겠습니다.

## 1. 역행렬 (Inverse)

- 행렬  $A$  의 역행렬은  $A^{-1}$  로 표현되며, 원래 행렬과 곱했을 때 항등 행렬  $I$  가 되는 행렬입니다. 즉,  $A \times A^{-1} = I$ . 모든 행렬이 역행렬을 가지는 것은 아니며, 역행렬이 존재하기 위해서는 행렬이 정방 행렬이어야 하고, 행렬의 행렬식 (determinant) 이 0 이 아니어야 합니다.

## 2. 행렬식 (Determinant)

- 행렬식은 정방행렬에 대해 정의되며, 행렬이 선형 변환을 수행했을 때 공간이 확장하거나 수축하는 정도를 나타내는 값입니다. 행렬식이 0 이면 행렬은 역행렬을 가지지 않으며, 이는 행렬이 특이행렬 (singular matrix) 임을 의미합니다.

## 3. 고유값 (Eigenvalue)

- 행렬  $A$  의 고유값은  $Av = \lambda v$  형식의 방정식을 만족하는  $\lambda$  값입니다. 여기서  $v$  는 0 이 아닌 벡터이며, 고유벡터 (eigenvector) 라고 합니다. 고유값과 고유벡터는 행렬이 벡터에 작용하는 효과를 분석할 때 중요한 역할을 합니다.

## 4. 회귀 (Regression)

- 회귀 분석은 변수들 사이의 관계를 모델링하고, 한 변수 (종속 변수) 를 다른 하나 또는 여러 개의 변수 (독립 변수) 로부터 예측하거나 추정하는 통계 기법입니다. 회귀는 선형 (linear) 또는 비선형 (non-linear) 일 수 있으며, 데이터의 패턴을 이해하고 미래 값을 예측하는 데 사용됩니다.

## 5. 공분산 (Covariance)

- 공분산은 두 변수  $X$  와  $Y$  가 함께 변하는 경향을 수치적으로 나타내는 통계적 척도입니다. 공분산이 양수면 한

변수가 증가할 때 다른 변수도 증가하는 경향을 보이고, 음수면 한 변수가 증가할 때 다른 변수는 감소하는 경향을 보입니다. 공분산이 0 에 가까우면 두 변수 사이에 선형 관계가 없다고 해석할 수 있습니다.

## 6. 상관계수 (Correlation)

- 상관계수는 두 변수 간의 선형 관계의 강도와 방향을 나타내는 척도입니다. 가장 널리 사용되는 피어슨 상관계수는  $-1$  에서  $1$  사이의 값을 가지며,  $1$  은 완벽한 양의 선형 관계,  $-1$  은 완벽한 음의 선형 관계,  $0$  은 변수 간에 선형 관계가 없음을 의미합니다. 상관계수는 공분산을 두 변수의 표준편차의 곱으로 나눈 것입니다.

## 7. GIGO (Garbage In, Garbage Out)

- GIGO 는 컴퓨터 과학과 데이터 과학에서 사용되는 용어로, 입력된 데이터의 질이 낮으면 출력 결과도 나쁠 것이라는 개념을 설명합니다. 즉, 분석이나 결과의 품질은 사용된 데이터의 품질에 직접적으

로 의존한다는 것을 의미합니다.

## 8. 내적 (Dot Product)

- 벡터의 내적 (또는 스칼라 곱) 은 두 벡터 간의 연산으로, 하나의 스칼라 값을 결과로 반환합니다. 두 벡터  $\mathbf{a}$  와  $\mathbf{b}$  의 내적은  $\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + \dots + a_nb_n$  으로 계산되며, 이는 벡터의 각 성분을 곱한 후 결과를 모두 더한 것입니다. 내적은 두 벡터의 방향성과 관련이 있으며, 특히 두 벡터의 각도나 두 벡터 간의 직교성을 판단할 때 유용합니다.

이러한 개념들은 각각의 분야에서 매우 중요하며, 데이터를 분석하고 모델을 구축하는 데 기초가 됩니다. 데이터 과학, 통계학, 수학, 물리학 등 다양한 분야에서 폭넓게 활용됩니다.

---

$f(x)$  를 미분한 형태를 표현할 때 사용하는 기호는  $f(x)$  가 스칼라 함수인지, 벡터 함수인지에 따라 달라집니다. 각 경우에 대해 적절한 표현 방식을 설명하겠습니다.

## 스칼라 함수의 경우

만약  $f(x)$  가 단일 변수 스칼라 함수일 경우 (예:  $f(x) = x^2$ ), 미분 결과는 일반적으로  $f'(x)$  또는  $\frac{df}{dx}$  로 표현합니다. 여기서 나블라 기호 ( $\nabla$ ) 는 사용되지 않습니다.

## 다변수 스칼라 함수의 경우

만약  $f(x)$  가 다변수 스칼라 함수일 경우 (예:  $f(x, y) = x^2 + y^2$ ), 함수의 각 변수에 대한 편미분을 모아 벡터 형태로 표현한 것이 그라디언트입니다. 이 때는 나블라 기호를 사용하여 그라디언트를 다음과 같이 표현할 수 있습니다:

$$\nabla f(x, y) = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

이 경우,  $\nabla f(x)$  는 다변수 스칼라 함수의 그라디언트를 적절히 표현한 것입니다. 그라디언트는 각 입력 변수에 대한 함수의 변화율을 벡터로 나타낸 것이며, 함수의 최대 증가 방향을 나타냅니다.

## 벡터 함수의 경우

$f(x)$  가 벡터 함수인 경우 (예:  $\mathbf{f}(x) = (x^2, x + 1)$ ), 함수의 미분 결과는 **야코비안 행렬 (Jacobian matrix)** 입니다. 야코비안은 함수의 각 성분의 편미분을 행렬 형태로 나열한 것으로, 나블라 기호는 일반적으로 이 경우에는 사용되지 않습니다.

## 결론

따라서  $f(x)$  를 미분한 형태를  $\nabla f(x)$  라고 표현하는 것은  $f(x)$  가 다변수 스칼라 함수일 때 그라디언트를 의미하는 경우에만 적합합니다. 단일 변수 함수나 벡터 함수의 경우에는 다른 표현 방식이 사용됩니다.

---

## $x + 2 \cdot y + 2 \cdot z = 6$ 이 평면에서 원점과 평면상의 점과 거리의 최소값을 알려줘

원점에서 주어진 평면까지의 거리는 평면의 방정식의 계수를 사용하여 계산할 수 있습니다. 이 거리는 평면의 방정식이

$$ax + by + cz = d$$

형태일 때 다음 공식으로 주어집니다:

$$\text{Distance} = \frac{|ax_0 + by_0 + cz_0 - d|}{\sqrt{a^2 + b^2 + c^2}}$$

여기서  $(x_0, y_0, z_0)$  는 점의 좌표입니다. 원점에서의 거리를 구하려면  $(x_0, y_0, z_0)$  을  $(0, 0, 0)$  로 설정합니다. 그렇게 되면 식은 다음과 같이 간단해집니다:

$$\text{Distance} = \frac{|d|}{\sqrt{a^2 + b^2 + c^2}}$$

주어진 평면의 방정식은

$$x + 2y + 2z = 6$$

입니다, 따라서  $a = 1, b = 2, c = 2, d = 6$  입니다.

이제 이 값을 대입해서 원점에서 평면까지의 최소 거리를 계산하겠습니다.

원점에서 주어진 평면  $x + 2y + 2z = 6$  까지의 최소 거리는 2 입니다.

---

```
import numpy as np

T = np.array([[2,2],[-1,10],[0,1],[-3,-3]])
print(np.abs(T))
print(np.sum(np.abs(T), axis=1))
print(np.argsort(np.sum(np.abs(T), axis=1)))
print(np.argsort(np.sum(np.abs(T), axis=1))[:,::-1])

# [[2,2],[-1,10],[0,1],[-3,-3]]
# [[2,2],[1,10],[0,1],[3,3]]
# [4,11,1,6]
# [2,0,3,1]
# [1,3,0,2]

print(np.argsort([1,2,3,4]))
print(np.argsort([4,11,1,6]))
```

---

```
x = np.random.rand(10000000)
print(np.mean(x))

# 0.5
```

```
y = np.arange(6).reshape(2,3)
b = y % 3 == 1
y[b] = -1
print(y[1])

# [0,1,2,3,4,5]
# y = [[0,1,2],[3,4,5]]
# y%3 = [[0,1,2],[0,1,2]]
# b = [[F,T,F],[F,T,F]]
# y = [3,-1,5]
```

```
v = np.logspace(0,10,11)
w = np.logspace(0,10,11)
X, Y = np.meshgrid(v, w)
print(v)
print(X[-1,-1])
print(X[-1,-1]*Y[-1,-1])
print(np.log10(X[-1,-1]*Y[-1,-1]))

print(np.meshgrid([1,2,3],[10,20]))
```

1e10\*1e10 = 1e20

```
def f(d, c, k):
    t = np.linalg.norm(d - c, axis=1, ord=1); print('t =', t)
    print('np.partition(t, k-1) =', np.partition(t, k-1))
    print('np.partition(t, k-1)[:k] =', np.partition(t, k-1)[:k])
    r = np.max(np.partition(t, k-1)[:k]); print('r =', r)
    print('np.argwhere(t <= r) =', np.argwhere(t <= r))
    i = np.argwhere(t <= r).flatten(); print('i =', i)
    return i

d = np.array([[2.,0.],[0.,2.],[-2.,0.],[0.,-2.],[1.,1.],[1.,-1.],
              [-1.,1.],[-1.,-1.]])
c = np.array([0.,0.])
print(f(d, c, 1))

...
t = [2. 2. 2. 2. 2. 2. 2. 2.]
```



```
np.partition(t, k-1) = [2. 2. 2. 2. 2. 2. 2. 2.]
np.partition(t, k-1)[:k] = [2.]
r = 2.0
np.argwhere(t <= r) = [[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]]
i = [0 1 2 3 4 5 6 7]
...
```

```
C = np.array([[2,0],[0,42.1]])
x = np.array([11234, -10000001])
for _ in range(100000):
    x = C.dot(x) / np.linalg.norm(x)

print(x / np.linalg.norm(x))
```

```
data = [3,1,3]
print(reduce(lambda acc,cur:acc+cur*cur, data,0))
```

---

```
x = np.arange(45).reshape(5,9)
y = x-(np.mean(x, axis=1)[:,np.newaxis])
print(y)
```

```
x = np.random.rand(1000001)
print(np.std((x-np.mean(x))/np.std(x)))
```

이 코드는 NumPy 를 사용하여 특정한 통계적 변환을 수행한 결과의 표준편차를 계산하고 출력합니다. 코드를 단계별로 분석해 보겠습니다.

### 1. 배열 생성:

```
x = np.random.rand(1000001)
```

여기서 `np.random.rand(1000001)` 는 0 과 1 사이의 균등 분포를 따르는 난수 1,000,001 개를 생성하여 배열 `x` 에 저장합니다.

### 2. 표준화 과정:

```
(x - np.mean(x)) / np.std(x)
```

이 부분에서 배열 `x` 의 모든 요소에서 `x` 의 평균 (`np.mean(x)`) 을 빼고, 그 결과를 `x` 의 표준편차 (`np.std(x)`) 로 나눕니다. 이는 `x` 의 데이터를 표준화하는 과정으로, 결과 데이터는 평균이 0 이고 표준편차가 1 인 정규분포 형태로 변환됩니다.

### 3. 표준편차 계산:

```
print(np.std((x-np.mean(x))/np.std(x)))
```

표준화된 데이터의 표준편차를 계산합니다. 이론적으로, 표준화 과정을 거친 데이터의 표준편차는 1 에 매우 근접하거나 1 이 되어야 합니다. 이는 표준화된 데이터가 단위 표준편차를 갖기 때문입니다.

결론적으로, 이 코드는 1000001 개의 난수를 생성하고, 이를 표준화하여 그 결과의 표준편차를 계산하는 것을 목표로 합니다. 출력 결과는 1 에 가깝거나 1 이 되어야 합니다, 이는 데이터 표준화 과정이 올바르게 수행되었음을 확인하는 방법입니다.

---

```
def corr(x,y):
    xm = x - np.mean(x)
    ym = y - np.mean(y)
    return (xm.dot(ym))/(np.linalg.norm(xm)*np.linalg.norm(ym))

x = np.arange(10)
y = np.arange(10)
print(corr(x,y))
```

코사인 유사도 (Cosine Similarity) 는 두 벡터 간의 코사인 각도를 계산하여, 그들의 유사성을 평가하는 방법입니다. 이 수치는 두 벡터의 방향이 얼마나 유사한지를 나타내며, 벡터의 크기는 고려하지 않습니다.

## 코사인 유사도의 수식은 다음과 같습니다

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\| \|B\|}$$

여기서:

- $A \cdot B$  는 두 벡터  $A$  와  $B$  의 점곱 (내적) 입니다.
- $\|A\|$  와  $\|B\|$  는 각각 벡터  $A$  와  $B$  의 노름 (크기) 입니다, 보통 유클리드 노름 (2- 노름) 을 사용합니다.

## 점곱 (내적) 과 노름 (크기) 의 계산

- 벡터의 내적은 다음과 같이 계산됩니다:

$$A \cdot B = \sum_{i=1}^n A_i B_i$$

여기서  $A_i$  와  $B_i$  는 각각 벡터  $A$  와  $B$  의  $i$  번째 요소입니다.

- 벡터의 노름 (크기) 는 다음과 같이 계산됩니다:

$$\|A\| = \sqrt{\sum_{i=1}^n A_i^2}$$

$$\|B\| = \sqrt{\sum_{i=1}^n B_i^2}$$

## 예제

두 벡터  $A = (1, 2, 3)$  와  $B = (4, 5, 6)$  에 대한 코사인 유사도를 계산하면:

1. 내적:  $1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 4 + 10 + 18 = 32$
2.  $A$  의 크기:  $\sqrt{1^2 + 2^2 + 3^2} = \sqrt{1 + 4 + 9} = \sqrt{14}$

3.  $B$ 의 크기:  $\sqrt{4^2 + 5^2 + 6^2} = \sqrt{16 + 25 + 36} = \sqrt{77}$

4. 코사인 유사도:  $\frac{32}{\sqrt{14} \times \sqrt{77}}$

코사인 유사도는 -1 에서 1 사이의 값을 가집니다. 값이 1 에 가까우면 두 벡터의 방향이 매우 유사하고, -1 에 가까우면 방향이 정반대임을 나타냅니다. 값이 0 이면 두 벡터는 서로 독립적, 즉 직교함을 의미합니다.

---