

로지스틱 회귀 (Logistic Regression)

학습 목표

- 로지스틱 회귀 모델의 개념과 구현을 알아본다.

주요 내용

1. 척도 조절
2. 선형 회귀와 분류 문제
3. 로지스틱 회귀
4. 최대 우도 추정
5. 경사 하강법으로 학습



1. 척도 조절

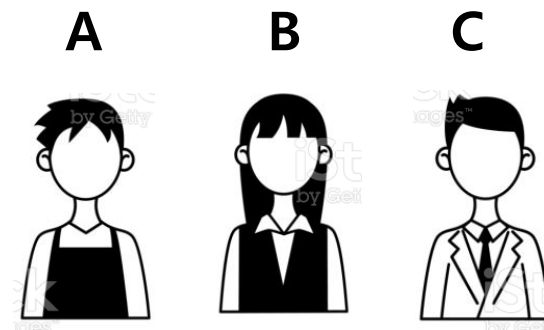
- 10장. 데이터 다루기 (Working with Data)



데이터의 척도 (Scale)

사람들의 키와 몸무게 데이터가 있을 때 체형을 군집화 한다고 해보자.

Person	Height (inches)	Height (centimeters)	Weight
A	63 inches	160 cm	150 pounds
B	67 inches	170.2 cm	160 pounds
C	70 inches	177.8 cm	171 pounds

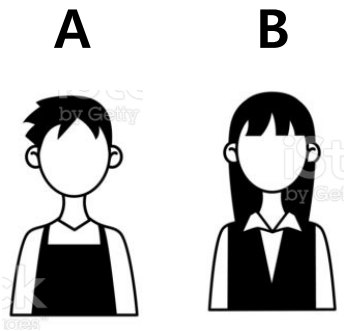


키를 inch와 cm 두 단위로 측정한 값이 있다. (1 inch = 2.54 cm)

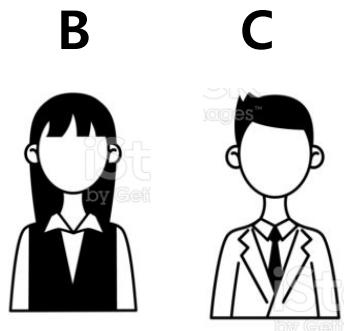
군집화를 하기 위해 데이터 간에 거리(distance)를 측정해보자.

데이터의 척도 (Scale)

어떤 척도를 사용하는지에 따라 거리가 달라진다.



A는 B와 가깝다.



B는 C와 가깝다.

키를 inch 단위로 측정했을 때

```
from scratch.linear_algebra import distance

a_to_b = distance([63, 150], [67, 160])      # 10.77
a_to_c = distance([63, 150], [70, 171])      # 22.14
b_to_c = distance([67, 160], [70, 171])      # 11.40
```

키를 cm 단위로 측정했을 때

```
a_to_b = distance([160, 150], [170.2, 160])  # 14.28
a_to_c = distance([160, 150], [177.8, 171])  # 27.53
b_to_c = distance([170.2, 160], [177.8, 171]) # 13.37
```

이런 현상이 왜 일어날까?

데이터의 척도 (Scale)

키를 inch 단위로 측정했을 때

Person	Height (inches)	Height (centimeters)	Weight
A	63 inches	160 cm	150 pounds
B	67 inches	170.2 cm	160 pounds
C	70 inches	177.8 cm	171 pounds

- 키보다 몸무게의 척도가 크다.
- 몸무게가 차이가 적은 A와의 거리가 가깝다.

키를 cm 단위로 측정했을 때

Person	Height (inches)	Height (centimeters)	Weight
A	63 inches	160 cm	150 pounds
B	67 inches	170.2 cm	160 pounds
C	70 inches	177.8 cm	171 pounds

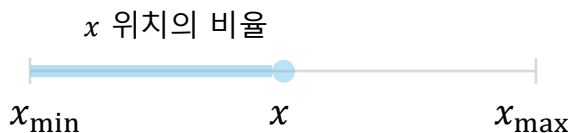
- 키와 몸무게의 척도가 비슷하다.
- 키의 차이가 적은 C와의 거리가 가깝다.

데이터의 척도가 크면 결과에 대한 영향이 크다!

데이터 표준화

따라서, 데이터의 척도를 같은 단위가 되도록 표준화해야 한다.

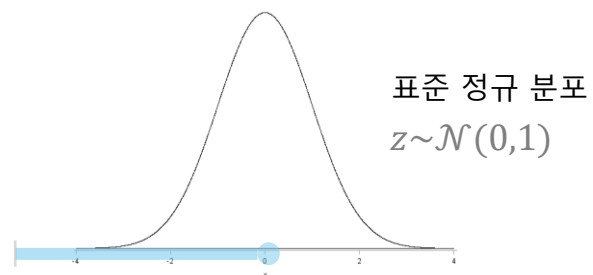
1. 최대 최소 정규화 (Min-max scaling)



$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- 데이터 구간의 값을 $[0,1]$ 구간으로 정규화
- 값의 최대 최소가 정해져 있을 때 유용

2. 정규분포 표준화 (standardization)



$$z = \frac{x - \mu}{\sigma}$$

μ : 평균 σ : 표준 편차

- 데이터의 68.3%가 $[-1,1]$ 구간으로,
95.5%가 $[-2,2]$ 구간으로,
99.7%가 $[-3,3]$ 구간으로 정규화
- 값의 구간이 $[-\infty, \infty]$ 일 때 유용

데이터 표준화

평균과 표준편차 구하기

```
from typing import Tuple

from scratch.linear_algebra import vector_mean
from scratch.statistics import standard_deviation

def scale(data: List[Vector]) -> Tuple[Vector, Vector]:
    """returns the means and standard deviations for each position"""
    dim = len(data[0])

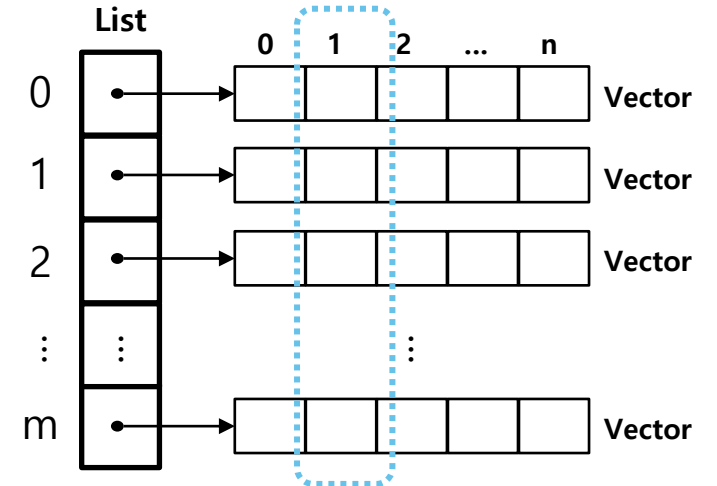
    means = vector_mean(data)
    stdevs = [standard_deviation([vector[i] for vector in data])
              for i in range(dim)]

    return means, stdevs
```

- 데이터의 차원 별로 평균과 표준 편차를 구한다.

3차원 데이터에 대해 테스트

```
vectors = [[-3, -1, 1], [-1, 0, 1], [1, 1, 1]]
means, stdevs = scale(vectors)
assert means == [-1, 0, 1]
assert stdevs == [2, 1, 0]
```

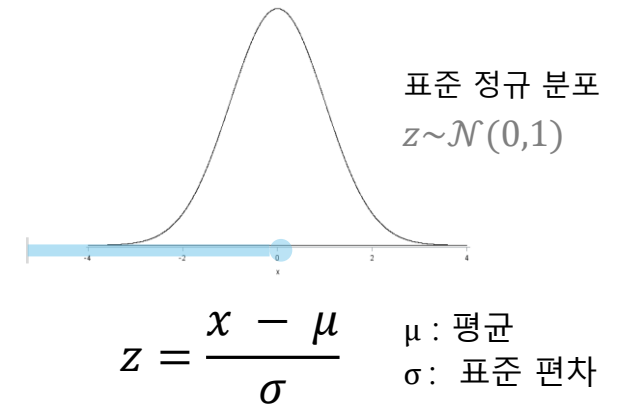


각 열 단위로 평균, 표준편차

데이터 표준화

표준 정규 분포로 표준화

```
def rescale(data: List[Vector]) -> List[Vector]:  
    """  
    Rescales the input data so that each position has  
    mean 0 and standard deviation 1. (Leaves a position  
    as is if its standard deviation is 0.)  
    """  
    dim = len(data[0])  
    means, stdevs = scale(data)  
  
    # Make a copy of each vector  
    rescaled = [v[:] for v in data]  
  
    for v in rescaled:  
        for i in range(dim):  
            if stdevs[i] > 0:  
                v[i] = (v[i] - means[i]) / stdevs[i]  
  
    return rescaled
```



표준화 했을 때 평균과 표준편차 확인

```
means, stdevs = scale(rescale(vectors))  
assert means == [0, 0, 1]  
assert stdevs == [1, 1, 0]
```


tqdm

함수나 반복문에서 작업의 진행율을 막대 그래프로 보여주는 툴

```
import tqdm
```

1 %pip install tqdm 설치 후 tqdm import

```
for i in tqdm.tqdm(range(100)):
```

2 Iterable을 tqdm으로 감싸면 bar가 나타남

오래 걸리는 연산



완료까지 남은 시간(Time to Completion)을 표시하는 진행 상태 바(progress bar)를 보여줌

tqdm이란 이름은 아랍어로 "progress"를 나타내는 단어 *taqaddum* (تَقَدُّم)에서 유래되었고 스페인어로 "I love you so much" 의 약자 *te quiero demasiado*이다.

tqdm

Jupyter Notebook에서 깔끔하게 출력하고 싶을 때 `tqdm.notebook.tqdm` 사용

import `tqdm`

for i in `tqdm.notebook.tqdm`(range(100)):

오래 걸리는 연산

9592 primes: 100%  99997/99997 [00:19<00:00, 5161.89it/s]

tqdm

trange 사용 예시

```
def primes_up_to(n: int) -> List[int]:
    primes = [2]

    with tqdm.trange(3, n) as t:
        for i in t:
            # i is prime if no smaller prime divides it.
            i_is_prime = not any(i % p == 0 for p in primes)
            if i_is_prime:
                primes.append(i)

            t.set_description(f"{len(primes)} primes")

    return primes

my_primes = primes_up_to(100_000)
```

- set_description을 사용하면 추가적인 정보를 보여줄 수 있다.



9592 primes: 100% 99997/99997 [00:19<00:00, 5161.89it/s]

set_description으로 표시한 정보

참고 밑줄(_) 숫자 표현 (Underscores in Numeric Literals)

PEP8에서 제안한 긴 숫자를 쉽게 읽을 수 있도록 _를 사용해서 표기 방법

```
amount = 10_000_000.0
```

10진수 : 천 단위로 그룹핑

```
addr = 0xCAFE_F00D
```

16진수 : word 단위로 그룹핑

```
flags = 0b_0011_1111_0100_1110
```

2진수 : 4 bit 단위로 그룹핑

```
flags = int('0b_1111_0000', 2)
```

2진수 문자열 변환

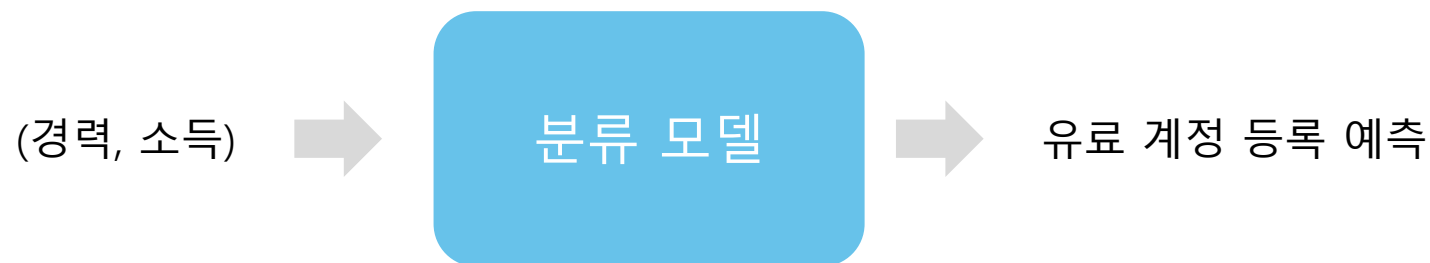
<https://www.python.org/dev/peps/pep-0515/>

2. 선형 회귀와 분류 문제



분류 문제

한 사이트에서 사용자의 경력, 소득 정보를 이용해서 유료 계정 등록을 예측하려고 한다.



선형 회귀 모델로 예측을 할 수 있을까?

$$\text{유료 계정 등록 여부} = \alpha + \beta_1 \text{경력} + \beta_2 \text{소득}$$

데이터셋

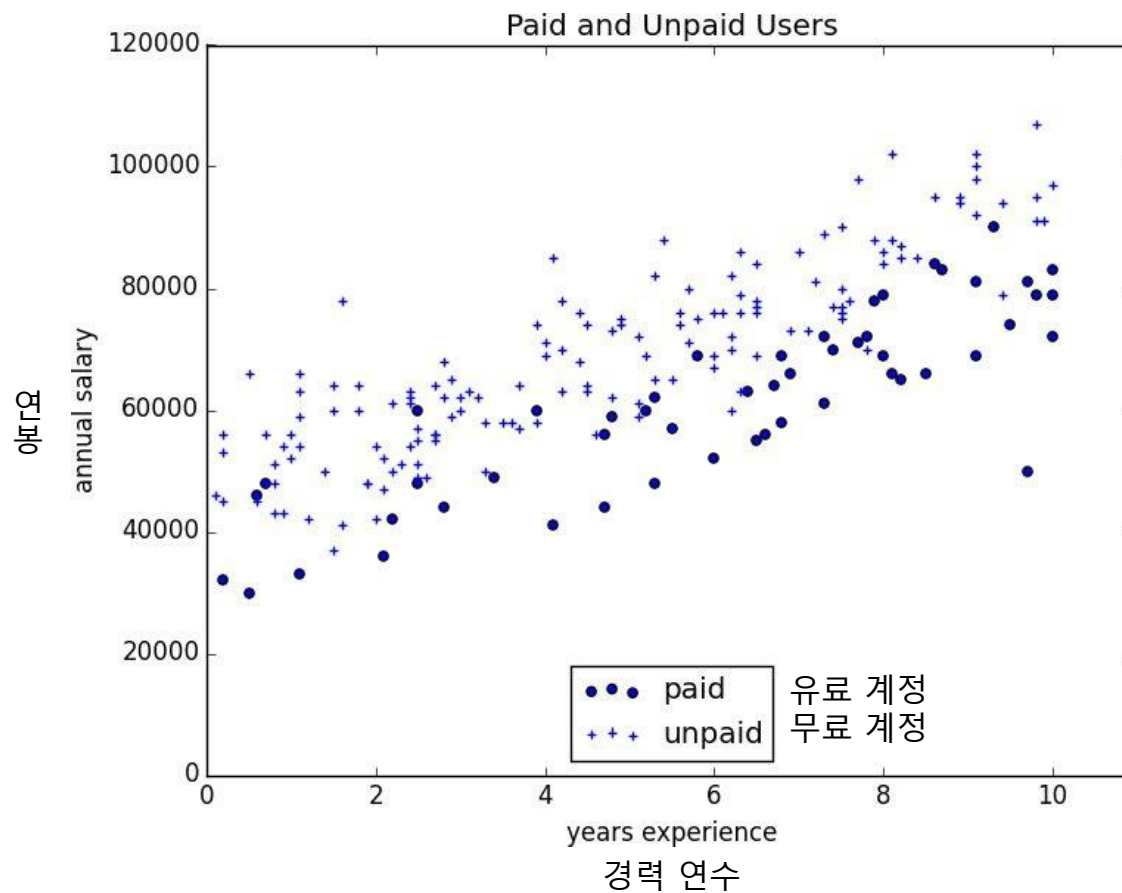
사용자 정보 데이터셋

```
tuples = [(0.7,48000,1),  
          (1.9,48000,0),  
          (2.5,60000,1),  
          (4.2,63000,0),  
          (6,76000,0),  
          (6.5,69000,0),  
          (7.5,76000,0),  
          (8.1,88000,0),  
          (8.7,83000,1),  
          ..., ]
```



- 200명 사용자
- (경력, 소득, 유료 계정 등록 여부)

데이터셋 시각화



다중 선형 회귀

입력과 레이블로 분리

```
data = [list(row) for row in tuples]

xs = [[1.0] + row[:2] for row in data] # [1, experience, salary]
ys = [row[2] for row in data]         # paid_account
```

선형 회귀 분석

```
from matplotlib import pyplot as plt
from scratch.working_with_data import rescale
from scratch.multiple_regression import least_squares_fit, predict
from scratch.gradient_descent import gradient_step

learning_rate = 0.001
rescaled_xs = rescale(xs)
beta = least_squares_fit(rescaled_xs, ys, learning_rate, 1000, 1)
# [0.26, 0.43, -0.43]
predictions = [predict(x_i, beta) for x_i in rescaled_xs]
```

- 입력 데이터를 표준정규분포로 표준화 (rescaled_xs)
- 선형 회귀 분석 및 예측

다중 선형 회귀

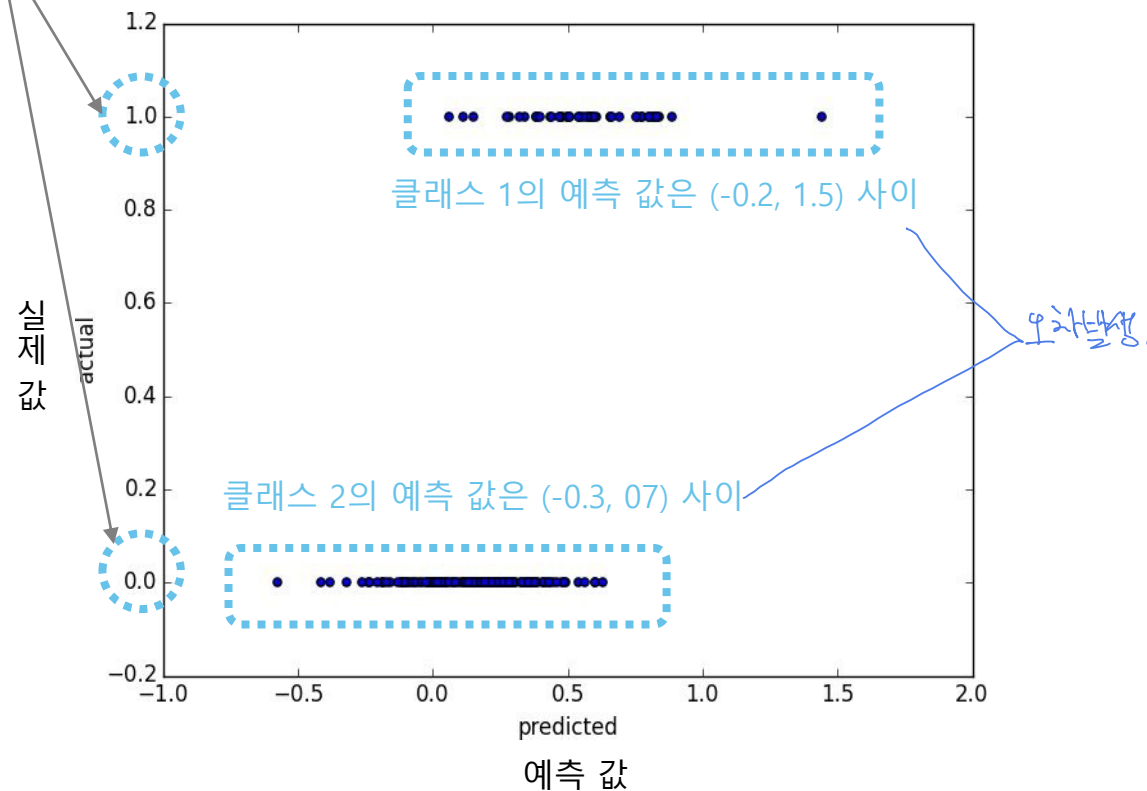
실제 y값은 0과 1로 되어 있는 명목 데이터 *음료 제전 0과 1 만 있음.*

실제 값과 예측 값의 산포도

```
plt.scatter(predictions, ys)
plt.xlabel("predicted")
plt.ylabel("actual")
plt.show()
```

선형 회귀를 분류 문제에 적용할 수 없는 이유

- 예측 값이 $[-\infty, \infty]$ 범위의 실수 값이므로 클래스 (0 or 1)를 정확히 예측하지 못함
- 분류 문제에서 예측 오차가 큰 편향된 모델

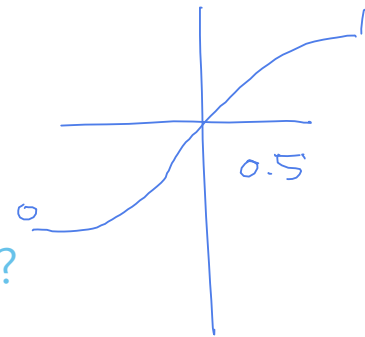


선형 회귀는 분류 문제에 적합하지 않다.

3. 로지스틱 회귀



이진 분류 (Binary Classification)

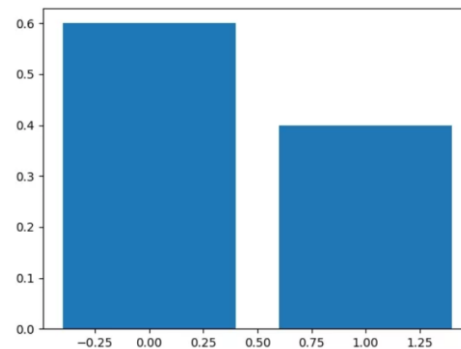


선형 회귀를 이용해서 이진 분류기(Binary Classifier)를 확률 모델로 만들 수 있을까?

이진 분류의 확률 모델은
베르누이 분포를 추정!



베르누이 분포 (Bernoulli Distribution)

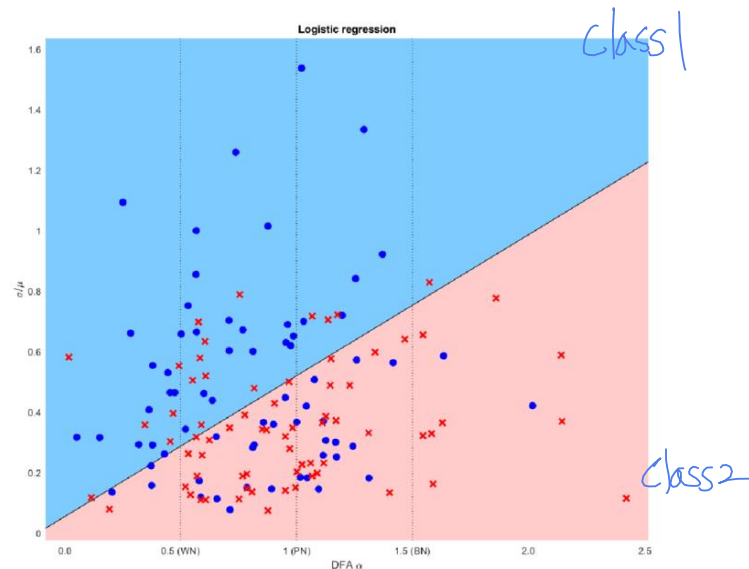


c_1 : 클래스 1 c_2 : 클래스 2

$$p(x; \mu) = \mu^x (1 - \mu)^{1-x}$$
$$x \in \{0, 1\}$$

로지스틱 회귀 (Logistic Regression)

일반화된 선형 모델 (Generalized Linear Model)의 한 종류로
독립 변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는 확률 모델



$$p(y = 1 | x) = f(x; \beta) = \sigma(\beta \cdot x) + \varepsilon$$

변환 함수
↓
클래스1의 조건부 확률
↑
선형 모델

관측 오차 $\varepsilon \sim \mathcal{N}(\varepsilon | 0, \sigma^2)$

정규 분포를 따르며 평균은 0이고 분산이 σ^2

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
 로지스틱 함수 (Logistic Function)

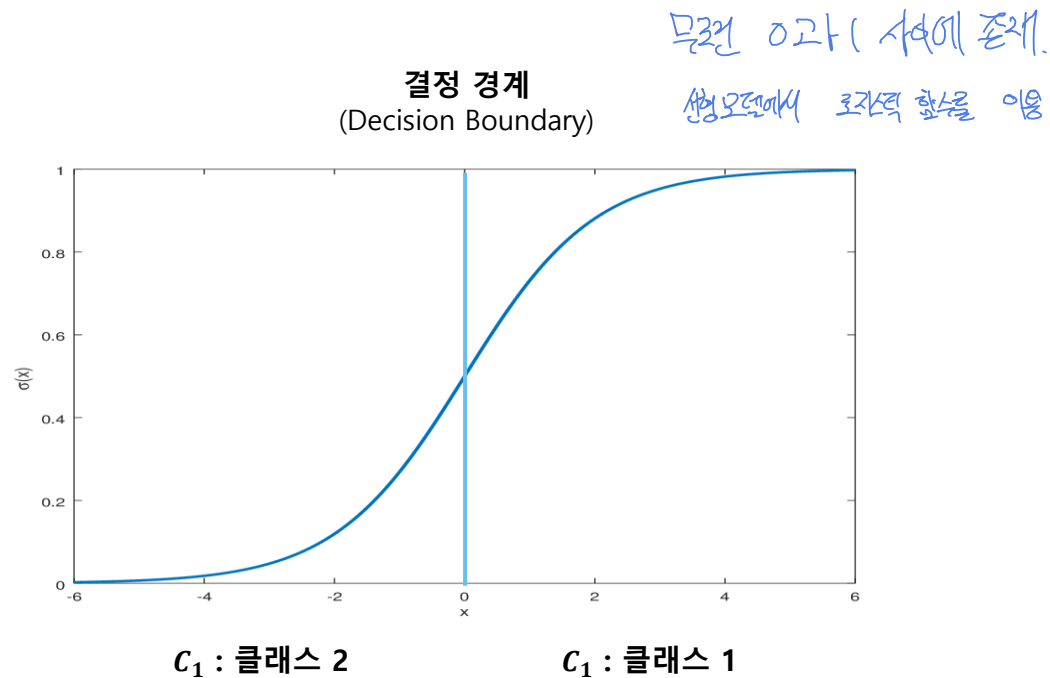
로지스틱 함수 (Logistic Function)

로지스틱 함수를 사용해서 선형 예측 결과를 확률로 변환한다.

로지스틱 함수 (Logistic Function)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

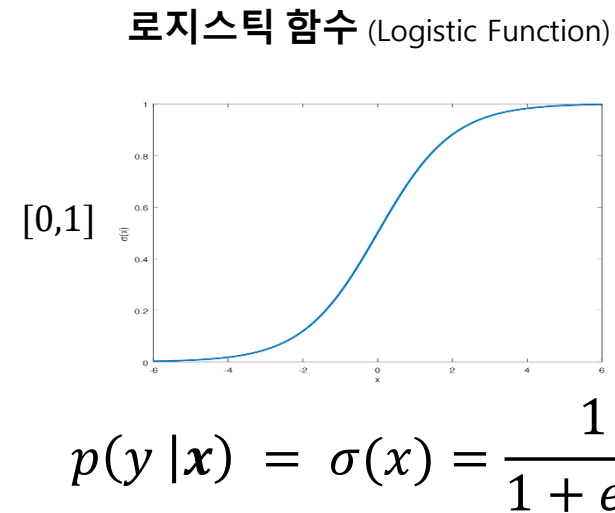
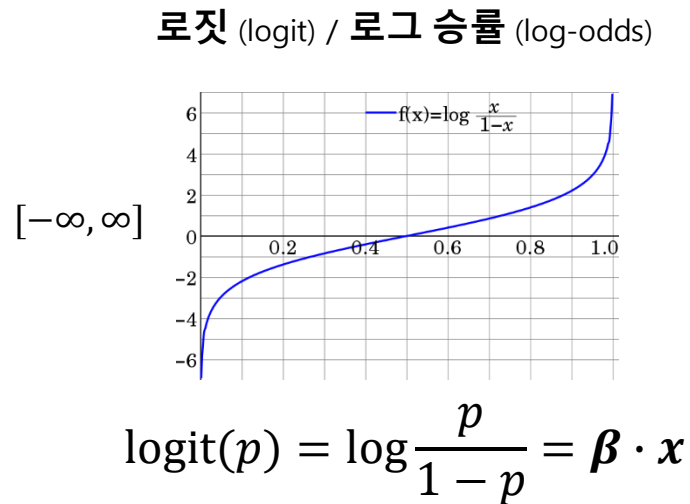
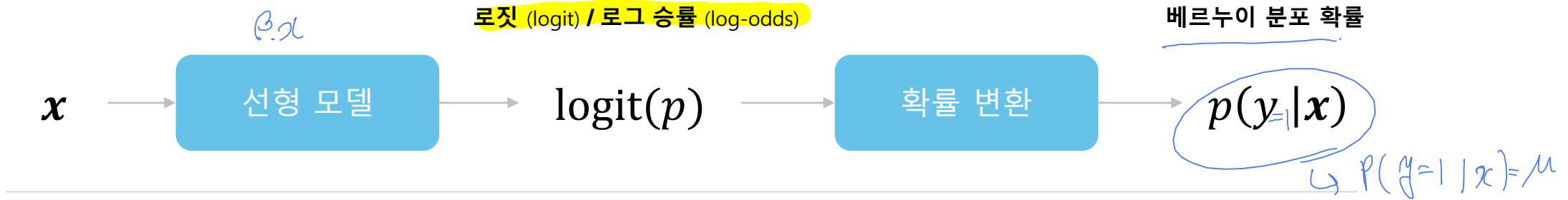
- $[-\infty, \infty]$ 범위의 값을 $[0, 1]$ 범위의 값으로 변환
- 실수 값을 확률로 변환할 때 사용
- 두 개의 클래스에 대한 확률 분포를 표현하는 베르누이 분포를 추정할 때 사용



- 결정 경계를 지정해서 해당 확률 보다 값이 크면 클래스 1로 분류하고 작으면 클래스 2로 분류

로지스틱 회귀 단계

선형 예측 결과는 로그 승률을 나타내는 로짓이 되며 이 값을 다시 확률로 변환한다.



출력값은 베르누이 분포에서 (μ) 에 해당됨.

로지스틱 함수 (Logistic Function)

로지스틱 함수

```
from matplotlib import pyplot as plt

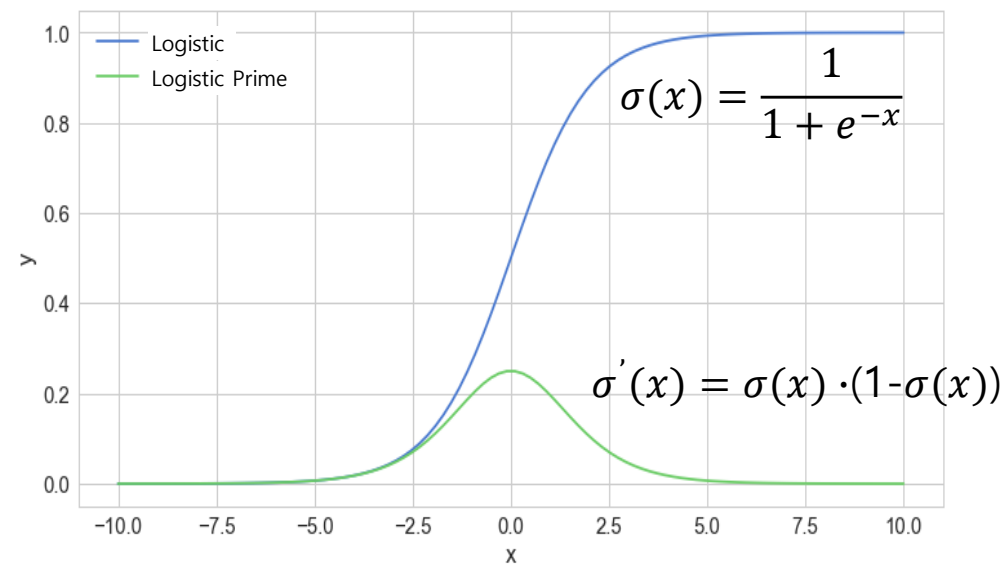
def logistic(x: float) -> float:
    return 1.0 / (1 + math.exp(-x))
```

로지스틱 함수의 미분

```
def logistic_prime(x: float) -> float:
    y = logistic(x)
    return y * (1 - y)
```

$$y = \sigma(x)$$

로지스틱 함수 (Logistic Function)



4. 최대 우도 추정



최적의 모델 (Optimal Model)

최적의 모델을 어떻게 찾을 것인가?

오차 최소화
(Error Minimization)

vs.

최대 가능도 추정
(Maximum Likelihood Estimation)

⇒ 손실함수 최소화

“모델의 예측 오차를 최소화 한다.”

“모델이 예측하는 관측 데이터의 확률을 최대화 한다.”

표현은 다르지만 이 둘의 결과는 같은 말이다.

최적화 기반의 학습 방식

오차를 최소화 하는 문제를 가능도를 최대화하는 문제로 바꾼다면?

손실 계산



최적화

- 모델의 예측 값과 관측 값의 오차를 이용해서 손실 계산
 - 회귀 문제 : 평균 오차 제곱 (Mean Squared Error)
 - 분류 문제 : 크로스 엔트로피 (Cross Entropy)
- 손실이 최소화되도록 파라미터를 변경
 - 파라미터의 최적해를 해석적으로 계산하지 못하는 경우
경사 하강법(Gradient Descent)와 같은 최적화 알고리즘을 사용
 - 최적해를 향해 반복적으로 수렴하도록 최적화 문제를 풀게 됨

최적화 기반의 학습 방식

오차를 최소화 하는 문제를 가능도를 최대화하는 문제로 바꾼다면?



- 모델의 예측 확률 분포에서 관측 데이터의 **가능도**를 계산

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

Handwritten annotations:
- $P(A|B)$: likelihood (likelihood)
- $P(B)$: prior (prior)
- $P(A)$: evidence (evidence)
- $P(B|A)$: posterior (posterior)
- A : 관측 데이터 (observed data)
- B : 가설 (hypothesis)

- 가능도가 최대화 되도록 파라미터를 변경**
- 목적 함수에 마이너스를 붙이면 최대화 문제는 최소화 문제로 풀 수 있다.

최대 가능도 추정 (Maximum Likelihood Estimate)

$$\frac{P(A|\theta)P(\theta)}{P(A)}$$

↑ 가설
↑ (μ, σ)
↑ 참값

가능도 (Likelihood)

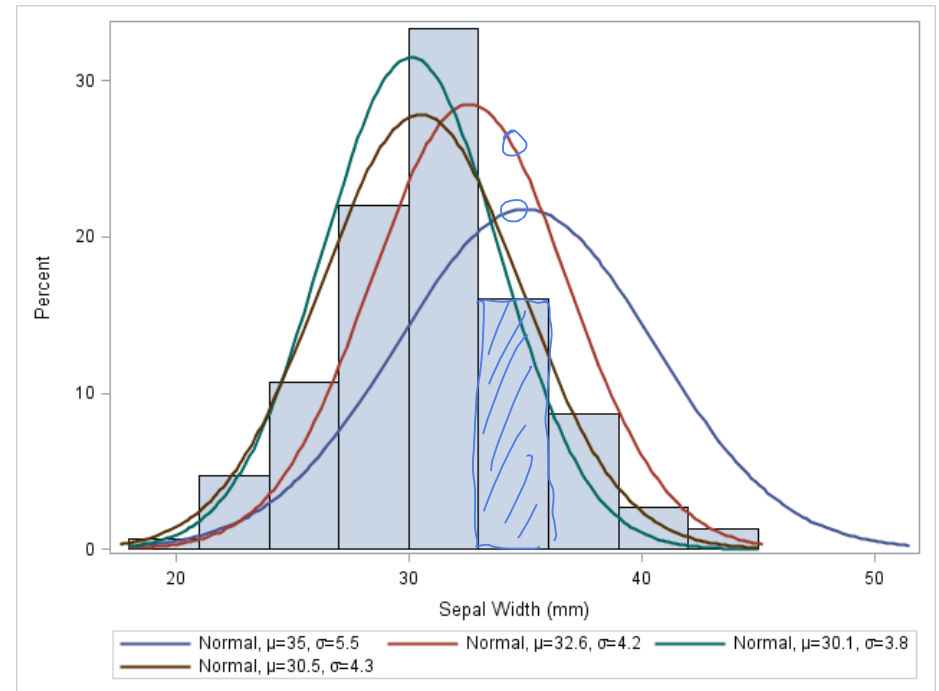
- 파라미터 θ 로 추정된 분포에서 관측 값의 확률 값
- 추정된 분포가 관측 데이터의 분포를 얼마나 잘 나타내고 있는지 또는 일관된 지를 나타내는 척도

$$\mathcal{L}(\theta|x) = p(x|\theta)$$

최대 가능도 추정 (MLE : Maximum Likelihood Estimate)

- 최대 가능도(Maximum Likelihood) 즉, 관측 값의 확률을 최대화 하는 추정 분포의 파라미터 θ 를 찾는 방법

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \mathcal{L}(\theta|x)$$



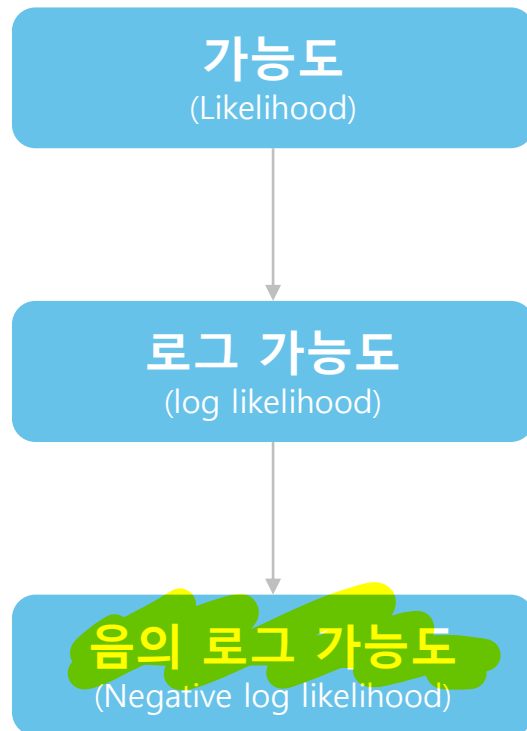
히스토그램과 확률분포와 비슷한 윤곽선이
관측된 값을 근사함.

추정된 확률 분포가 관측 데이터의 분포의 모양을 잘 표현해야 가능도가 최대화 될 수 있다.

최대 가능도 추정 (Maximum Likelihood Estimate)

최대 가능도 추정 문제를 표준 형태의 최적화 문제로 재정의해보자.

가능도를 음의 로그 가능도로 변환해서 최대화 문제를 최소화 문제로 만든다.



- 관측 값의 가능도

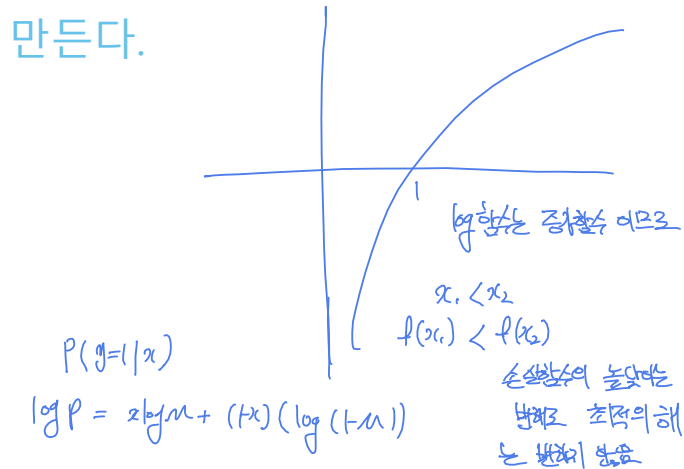
- 모델 예측한 확률 분포를 이용해서 계산

- 확률 분포에 로그 적용

- 지수족(exponential family) 확률 분포에 log를 적용하면 지수 항이 다항식으로 변환되어 수치적으로 다루기가 쉽다.
- 또한, 목적 함수에 log와 같은 증가 함수를 적용해도 최적해는 동일하다.

- 최대화 문제를 최소화 문제로 변경

- 로그 가능도에 마이너스 붙인 음의 로그 가능도를 목적 함수로 정의



최대 가능도 추정 (Maximum Likelihood Estimate)

음의 로그 가능도 (NLL : Negative Log Likelihood)

관측 데이터

$$J(\boldsymbol{\beta}) = -\log P(y | \mathbf{x}; \boldsymbol{\beta}) \Rightarrow \text{가능도} \quad \beta \text{를 추정할 때 확률}$$

가능도 β 로 추정된 모델에서의 관측 데이터의 확률

μ

$$= -\log f(\mathbf{x}; \boldsymbol{\beta})^y (1 - f(\mathbf{x}; \boldsymbol{\beta}))^{1-y}$$

모델이 예측한 μ

$$= -(y \cdot \log f(\mathbf{x}; \boldsymbol{\beta}) + (1 - y) \cdot \log(1 - f(\mathbf{x}; \boldsymbol{\beta})))$$

이진 크로스 엔트로피 (Binary Cross Entropy)

베르누이 분포 (Bernoulli Distribution)

$$p(x; \mu) = \mu^x (1 - \mu)^{1-x} \quad x \in \{0, 1\}$$

- 관측 데이터 y
 - $y = 1$: 클래스1 (ex, 동전의 앞면)
 - $y = 0$: 클래스2 (ex, 동전의 뒷면)
- 모델 $f(\mathbf{x}; \boldsymbol{\beta})$
 - 베르누이 분포의 경우 확률 모델은 클래스1일 확률인 μ 를 출력

μ 예측

손실 함수 크로스 엔트로피

이진 크로스 엔트로피 (Binary Cross Entropy)

$$\min_{\beta} - \sum_{i=1}^N y \cdot \log f(x; \beta) + (1 - y) \cdot \log(1 - f(x; \beta))$$

최대 엔트로피 추정은 확률 분포에
가장 가까운 형태로
1/2이 없는 형태

관측 데이터 $\mathcal{D} = \{(x_i, y_i) : i = 1 \dots N\}$

- $K = 2$ 일 때 크로스 엔트로피를 이진 크로스 엔트로피라고 한다.

크로스 엔트로피 (Cross Entropy)

$$\min_{\theta} - \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K y_{ik} \cdot \log f(x_i; \theta)_k$$

K : Class 개수 $y_k = \begin{cases} 1, k = c \\ 0, k \neq c \end{cases} \quad c \in \{1, 2, \dots, K\}$

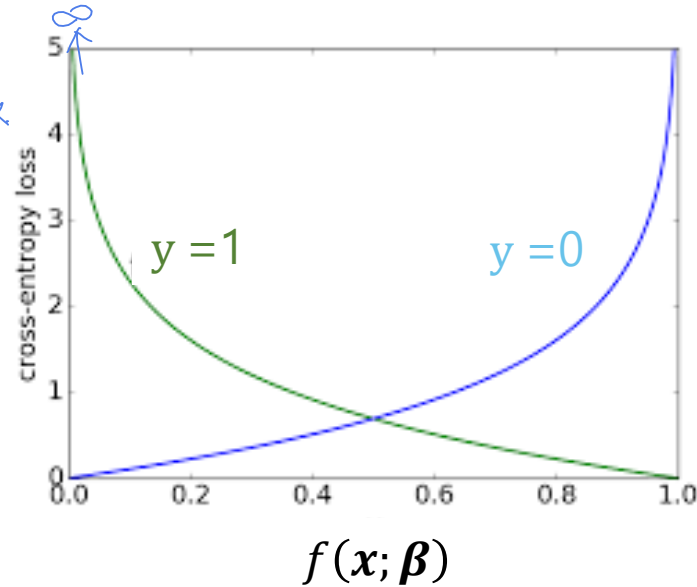
관측 분포
최적화에 사용
모델이 예측한 분포

이진 크로스 엔트로피 (Binary Cross Entropy)

$$J(\boldsymbol{\beta}) = -(y \cdot \log f(\mathbf{x}; \boldsymbol{\beta}) + (1 - y) \cdot \log(1 - f(\mathbf{x}; \boldsymbol{\beta})))$$

1 $y = 1$ 일 때 $(1-y)=0$
 $J(\boldsymbol{\beta}) = -\log(f(\mathbf{x}; \boldsymbol{\beta}))$

Handwritten notes: $(1-y)=0$ (blue), \log (blue), \times (blue)



2 $y = 0$ 일 때 $y=0$
 $J(\boldsymbol{\beta}) = -\log(1 - f(\mathbf{x}; \boldsymbol{\beta}))$

Handwritten notes: $y=0$ (blue), \log (blue), \times (blue)

$y = f(\mathbf{x}; \boldsymbol{\beta})$ 인 경우에는 엔트로피는 0이 됨
(즉, y 와 $f(\mathbf{x}; \boldsymbol{\beta})$ 모두 1이거나 모두 0인 경우)

5. 경사 하강법으로 학습



손실 함수 미분

로지스틱 회귀 식

$$p(y = 1 | x) = f(x; \beta) = \sigma(\beta \cdot x)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

로지스틱 함수의 미분

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

손실 함수 (이진 크로스 엔트로피)

대입

$$J(\beta) = -(y \cdot \log f(x; \beta) + (1 - y) \cdot \log(1 - f(x; \beta)))$$

$$= -(y \cdot \log \sigma(\beta \cdot x) + (1 - y) \cdot \log(1 - \sigma(\beta \cdot x)))$$

β_j 에 대해서 미분 (단, $\sigma(\beta \cdot x)$ 는 σ 로 간단히 표기)

$$\frac{\partial J}{\partial \beta_j} = - \left(y \cdot \frac{1}{\sigma} \cdot \sigma \cdot (1 - \sigma) \cdot x_j - (1 - y) \cdot \frac{1}{(1 - \sigma)} \cdot \sigma \cdot (1 - \sigma) \cdot x_j \right)$$

$$= -(y \cdot (1 - \sigma) \cdot x_j - (1 - y) \cdot \sigma \cdot x_j)$$

$$= -(y - y\sigma - \sigma + y\sigma) x_j$$

$$= -(y - \sigma)x_j$$

손실 함수 (loss)

$$J(\boldsymbol{\beta}) = -(y \cdot \log f(\mathbf{x}; \boldsymbol{\beta}) + (1 - y) \cdot \log(1 - f(\mathbf{x}; \boldsymbol{\beta})))$$

1 $y = 1$ 일 때

$$J(\boldsymbol{\beta}) = -\log(f(\mathbf{x}; \boldsymbol{\beta}))$$

2 $y = 0$ 일 때

$$J(\boldsymbol{\beta}) = -\log(1 - f(\mathbf{x}; \boldsymbol{\beta}))$$

음의 로그 우도 (NLL : Negative Log Likelihood)

```
import math
from scratch.linear_algebra import Vector, dot

def _negative_log_likelihood(x: Vector, y: float, beta: Vector) -> float:
    """The negative log likelihood for one data point"""
    if y == 1:
        return -math.log(logistic(dot(x, beta)))
    else:
        return -math.log(1 - logistic(dot(x, beta)))
```

$$-\log(1 - \sigma(\mathbf{x}, \boldsymbol{\beta}))$$

손실 함수 (loss)

$$\min_{\boldsymbol{\beta}} - \sum_{i=1}^N y \cdot \log f(\mathbf{x}; \boldsymbol{\beta}) + (1 - y) \cdot \log(1 - f(\mathbf{x}; \boldsymbol{\beta}))$$

Sum

전체 데이터셋에 대해 NLL 합산

```
from typing import List

def negative_log_likelihood(xs: List[Vector],
                           ys: List[float],
                           beta: Vector) -> float:
    return sum(_negative_log_likelihood(x, y, beta)
               for x, y in zip(xs, ys))
```


각각해서 Sum

그래디언트 계산 (Gradient)

β_j 에 대한 NLL 편미분

```
from scratch.linear_algebra import vector_sum

def _negative_log_partial_j(x: Vector, y: float, beta: Vector, j: int) -> float:
    """
    The j-th partial derivative for one data point
    here i is the index of the data point
    """
    return -(y - logistic(dot(x, beta))) * x[j]
```

$$\frac{\partial J}{\partial \beta_j} = -(y - \sigma)x_j$$


β 에 대한 그래디언트

β 값들에 대한 Vector 미분

```
def _negative_log_gradient(x: Vector, y: float, beta: Vector) -> Vector:
    """
    The gradient for one data point
    """
    return [_negative_log_partial_j(x, y, beta, j)
            for j in range(len(beta))]
```

$$\begin{aligned} \frac{\partial J}{\partial \beta} &= -(y - \sigma)x \\ &= \left(\frac{\partial J}{\partial \beta_1}, \frac{\partial J}{\partial \beta_2}, \dots, \frac{\partial J}{\partial \beta_n} \right) \end{aligned}$$

그래디언트 계산 (Gradient)

전체 데이터 셋에 대해 그래디언트 합산

```
def negative_log_gradient(xs: List[Vector],  
                          ys: List[float],  
                          beta: Vector) -> Vector:  
    return vector_sum([_negative_log_gradient(x, y, beta)  
                       for x, y in zip(xs, ys)])
```

경사 하강법으로 로지스틱 회귀

데이터셋 분리

```
from scratch.machine_learning import train_test_split
import random
import tqdm

random.seed(0)
x_train, x_test, y_train, y_test = train_test_split(rescaled_xs, ys, 0.33)
```

데이터 스플릿 33%.

초기화

```
learning_rate = 0.01

# pick a random starting point
beta = [random.random() for _ in range(3)]
```

경사 하강법 적용

여러번 β 초기화

```
with tqdm.trange(5000) as t:
    for epoch in t:
        gradient = negative_log_gradient(x_train, y_train, beta)
        beta = gradient_step(beta, gradient, -learning_rate)
        loss = negative_log_likelihood(x_train, y_train, beta)
        t.set_description(f"loss: {loss:.3f} beta: {beta}")
```

5000번 돌음.

훈련하면서 loss가 잘리고 있는지 확인함.

로지스틱 회귀 검증

원래 입력 데이터와 정규화 된 입력 데이터의 NLL 비교

```
from scratch.working_with_data import scale

means, stdevs = scale(xs)
beta_unscaled = [(beta[0]
                  - beta[1] * means[1] / stdevs[1]
                  - beta[2] * means[2] / stdevs[2]),
                  beta[1] / stdevs[1],
                  beta[2] / stdevs[2]]
# [8.9, 1.6, -0.000288]
assert (negative_log_likelihood(xs, ys, beta_unscaled) ==
        negative_log_likelihood(rescaled_xs, ys, beta))
```

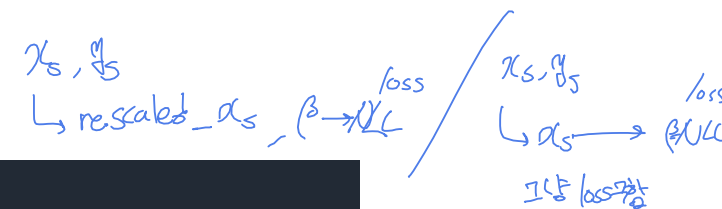
- 데이터셋의 평균과 표준편차를 구함

$$\mathbf{x} = (1, x_1, x_2) \quad \boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2) \quad \boldsymbol{\beta}_{\text{unscaled}} = \left(\beta_0 - \beta_1 \frac{\bar{x}_1}{s_1} - \beta_2 \frac{\bar{x}_2}{s_2}, \frac{\beta_1}{s_1}, \frac{\beta_2}{s_2} \right)$$

데이터를 표준화 했을 때 β 데이터를 표준화 하기 전 β

$\boldsymbol{\beta}_{\text{unscaled}} = [8.9, 1.6, -0.000288]$

- $\beta_1 = 1.6$: 경력이 많을수록 유료 계정에 등록할 가능성이 높음
- $\beta_2 = -0.000288$: 월급이 높을수록 유료 계정에 등록할 가능성이 낮음



데이터 표준화 안했을때
데이터 표준화했을때
차이를 보지 못함.

참고 로지스틱 회귀 모델 적용

데이터를 표준화를 해서 구한 β 를 이용해서
데이터를 표준화 하기 전의 $\beta_{unscaled}$ 를 계산할 수 있다.

$$\begin{aligned} \text{유료 계정 등록 여부} &= \sigma(\beta_0 + \beta_1 \text{경력} + \beta_2 \text{소득}) && x_1 : \text{경력}, x_2 : \text{소득} && \mathbf{x} = (1, x_1, x_2) \\ &= \sigma\left(\beta_0 + \beta_1 \frac{x_1 - \bar{x}_1}{s_1} + \beta_2 \frac{x_2 - \bar{x}_2}{s_2}\right) && \text{입력 데이터 표준화} && \boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2) \\ &= \sigma\left(\left(\beta_0 - \beta_1 \frac{\bar{x}_1}{s_1} - \beta_2 \frac{\bar{x}_2}{s_2}\right) + \frac{\beta_1}{s_1} x_1 + \frac{\beta_2}{s_2} x_2\right) && \text{데이터를 표준화 했을 때 } \beta && \text{상수항과 } x_1 \text{과 } x_2 \text{ 항으로 정리} \\ & && \text{따라서, } \boldsymbol{\beta}_{unscaled} = \left(\beta_0 - \beta_1 \frac{\bar{x}_1}{s_1} - \beta_2 \frac{\bar{x}_2}{s_2}, \frac{\beta_1}{s_1}, \frac{\beta_2}{s_2}\right) \text{ 이다.} && \end{aligned}$$

데이터를 표준화 하기 전 β

모델 성능

정밀도와 재현율 계산

```
true_positives = false_positives = true_negatives = false_negatives = 0

for x_i, y_i in zip(x_test, y_test):
    prediction = logistic(dot(beta, x_i))

    if y_i == 1 and prediction >= 0.5: # TP: paid and we predict paid       $\hat{y}=1$  0.5↑
        true_positives += 1
    elif y_i == 1: # FN: paid and we predict unpaid       $\hat{y}=1$  0.5↓
        false_negatives += 1
    elif prediction >= 0.5: # FP: unpaid and we predict paid       $\hat{y}_i=0$  0.5↑
        false_positives += 1
    else: # TN: unpaid and we predict unpaid       $\hat{y}_i=0$  0.5↓
        true_negatives += 1

precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)

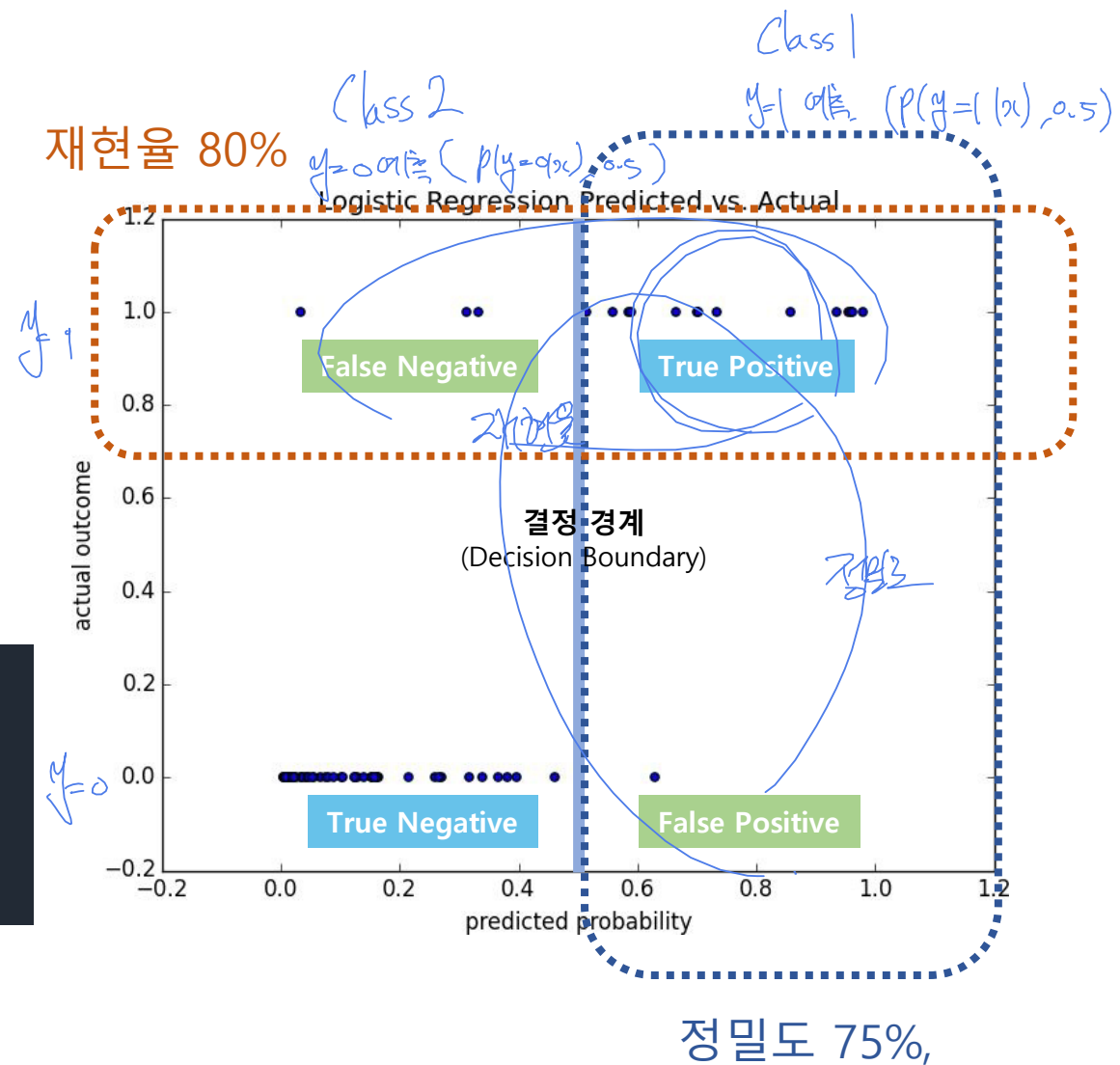
assert precision == 0.75
assert recall == 0.8
```

정밀도 75%, 재현율 80%

모델 성능

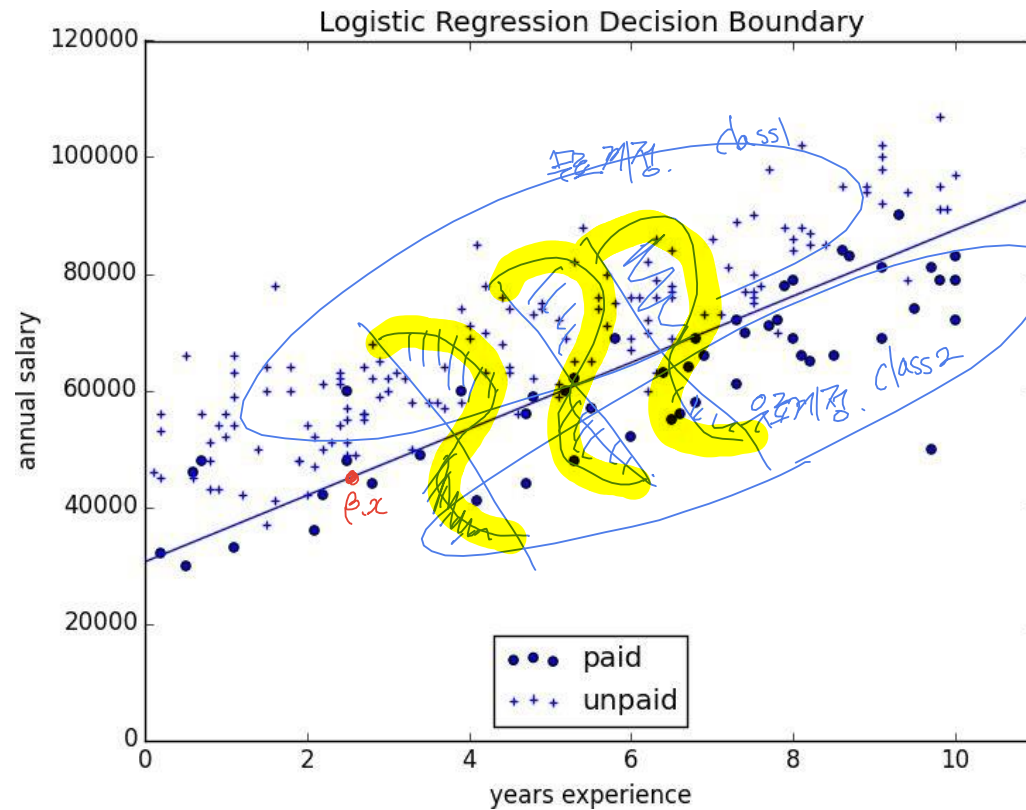
실제값과 예측값 시각화

```
predictions = [logistic(dot(beta, x_i)) for x_i in x_test]
plt.scatter(predictions, y_test, marker='+')
plt.xlabel("predicted probability")
plt.ylabel("actual outcome")
plt.title("Logistic Regression Predicted vs. Actual")
plt.show()
```



결정 경계 (Decision Boundary)

로지스틱 회귀 분석 결과 결정 경계는 $\beta \cdot x = 0$ 로 정의되는 초평면이다.

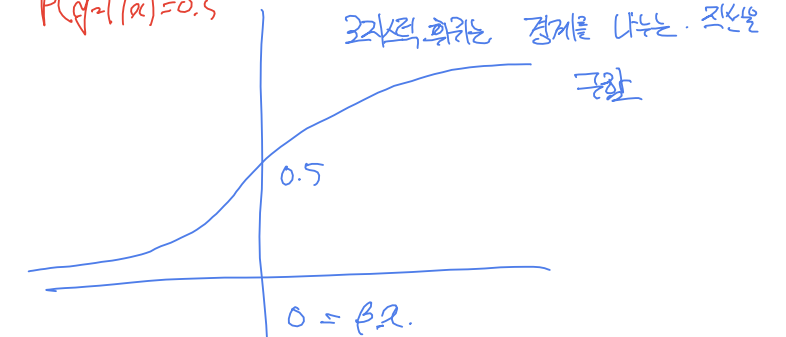


결정 경계
(Decision Boundary)

$$\beta \cdot x = 0$$

$$P(y=1|x) = 0.5$$

$$P(y=1|x) = P(x; \beta) = \sigma(\beta \cdot x)$$



Thank you!

