AI 라이브러리 활용

5장 참 거짓 판단 장치: 로지스틱 회귀

이 찬 우

학 습 내 용

1 | 로지스틱 회귀의 정의

2 | 시그모이드 함수

3 | 오차 공식

4 | 로그 함수

5 | 코딩으로 확인하는 로지스틱 회귀

6 | 로지스틱 회귀에서 퍼셉트론으로

참 거짓 판단 장치: 로지스틱 회귀

- 전달받은 정보를 놓고 참과 거짓 중에 하나를 판단해 다음 단계로 넘기는 장치들이 딥러닝 내부에서 쉬지 않고 작동함
- 딥러닝을 수행한다는 것은 겉으로 드러나지 않는 '미니 판단 장치'들을 이용해서 복잡한 연산을 해낸 끝에 최적의 예측 값을 내놓는 작업

참 거짓 판단 장치: 로지스틱 회귀

- 참인지 거짓인지를 구분하는 로지스틱 회귀의 원리를 이용해 '참, 거짓 미니 판단 장치'를 만들어 주어진 입력 값의 특징을 추출함
- 이를 저장해서 '모델(model)'을 만듦
- 누군가 비슷한 질문을 하면 지금까지 만들어 놓은 이 모델을 꺼내어 답을 함
  - ✓ 이것이 바로 딥러닝의 동작 원리임

1 | 로지스틱 회귀의 정의

- 직선으로 해결하기에는 적절하지 않은 경우도 있음
- 점수가 아니라 오직 합격과 불합격만 발표되는 시험이 있다고 하자

공부한 시간	2	4	6	8	10	12	14
합격 여부	불합격	불합격	불합격	합격	합격	합격	합격

표 5-1 공부한 시간에 따른 합격 여부

1 | 로지스틱 회귀의 정의

- 합격을 1, 불합격을 0이라 하고, 이를 좌표 평면에 표현하면 그림 5-1과 같음

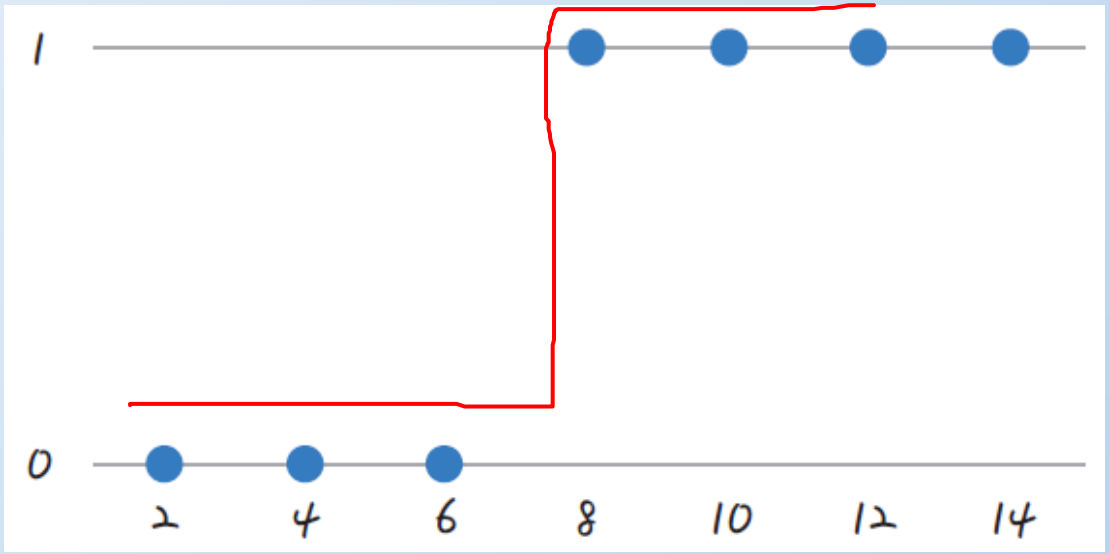
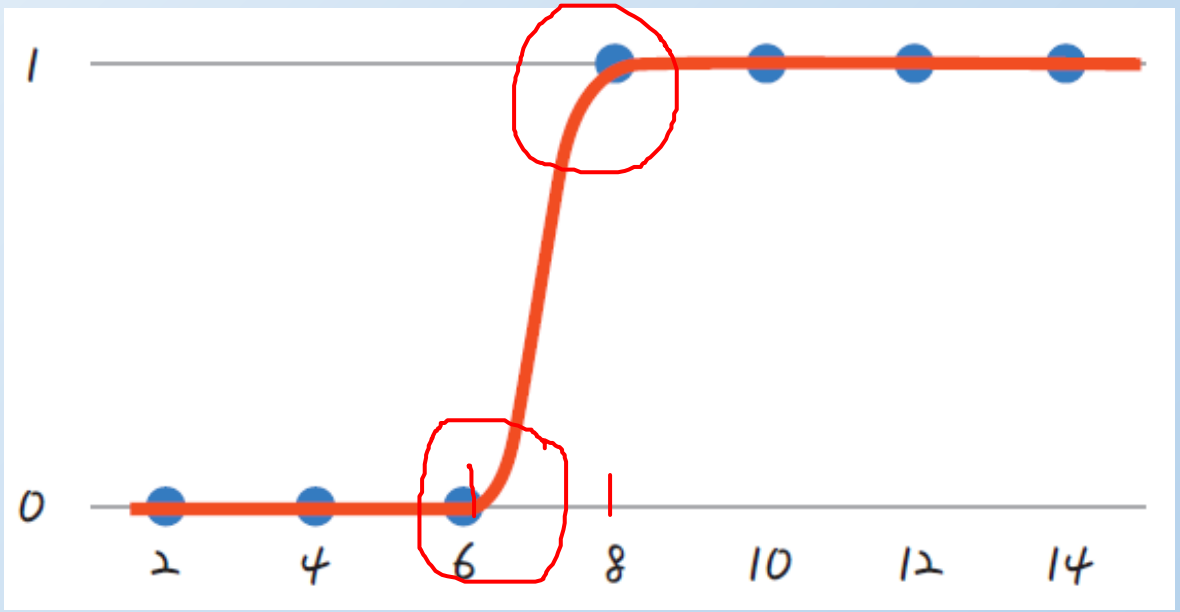


그림 5-1 합격과 불합격만 있을 때의 좌표 표현

1 | 로지스틱 회귀의 정의

- 이 점들은 1과 0 사이의 값이 없으므로 직선으로 그리기가 어려움
- 점들의 특성을 정확하게 담아내려면 직선이 아니라 다음과 같이 S자 형태여야 함



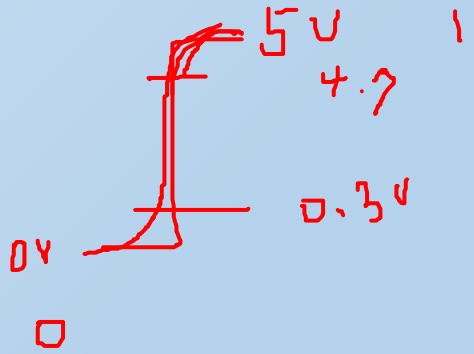


그림 5-2 각 점의 특성을 담은 선을 그었을 때

1 | 로지스틱 회귀의 정의

- 로지스틱 회귀 :

  선형 회귀와 마찬가지로 적절한 선을 그려가는 과정

- 다만 직선이 아니라, 참(1)과 거짓(0) 사이를 구분하는 S자 형태의 선을 그어 주는작업

2 | 시그모이드 함수

- 시그모이드 함수(sigmoid function) :

  S자 형태로 그래프가 그려지는 함수

- 시그모이드 함수를 이용해 로지스틱 회귀를 풀어나가는 공식은 다음과 같음

$$y = \frac{1}{1 + e^{-(ax+b)}}$$

2 | 시그모이드 함수

- a 는 그래프의 경사도를 결정함

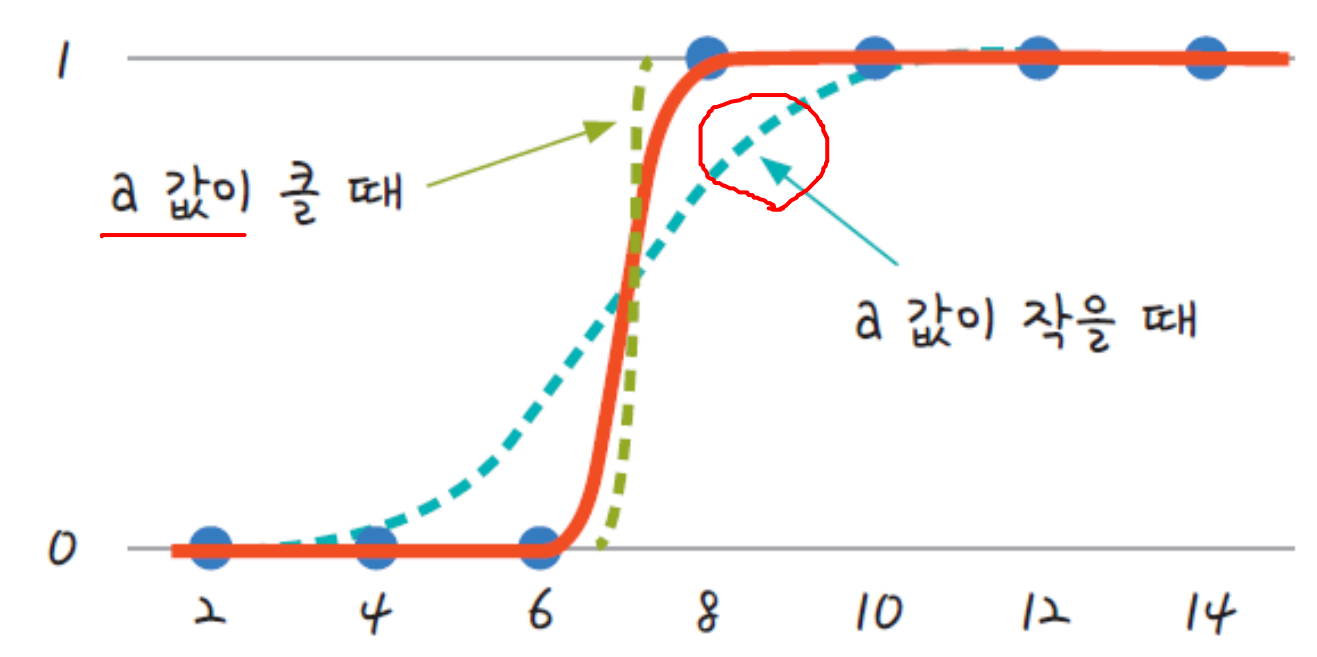


그림 5-3 a 값이 클 때와 작을 때의 그래프 변화

2 | 시그모이드 함수

- b 그래프의 좌우 이동을 의미함

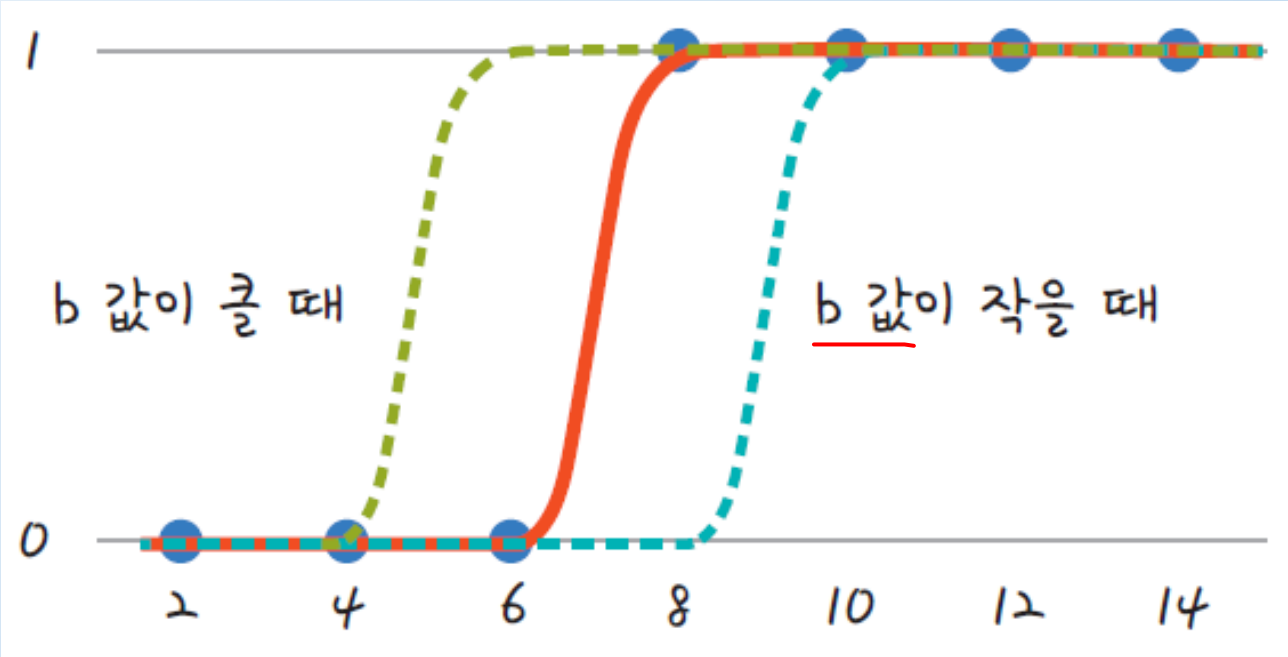


그림 5-4 b 값이 클 때와 작을 때의 그래프 변화

2 | 시그모이드 함수

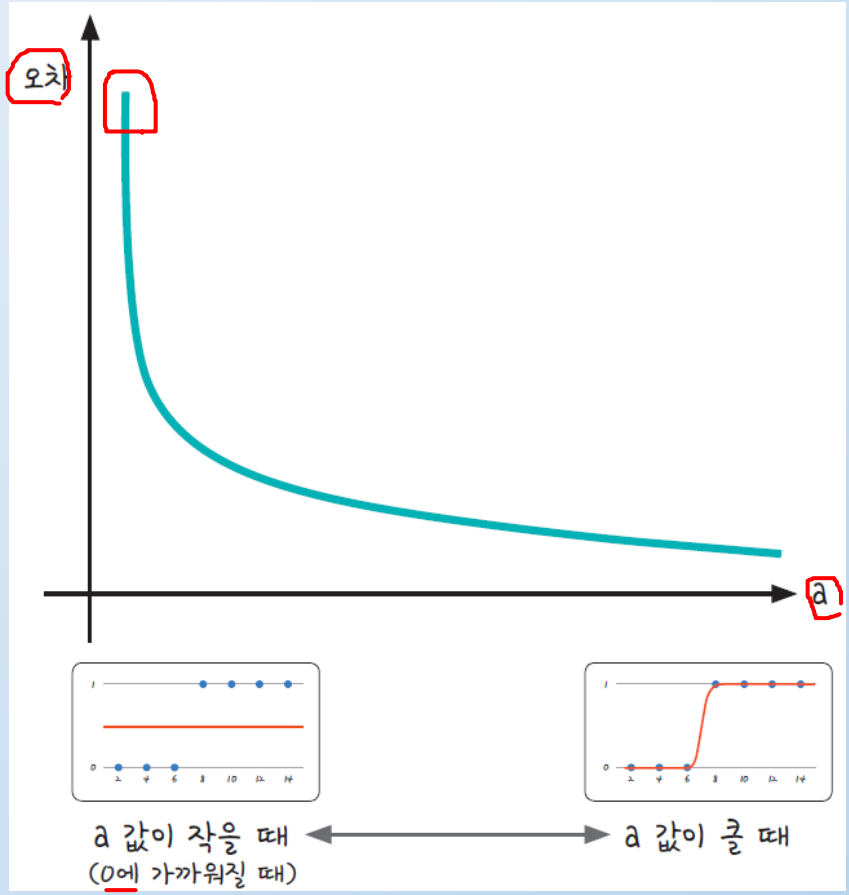


그림 5-5 a와 오차와의 관계: a가 작아질수록 오차는 무한대로 커지지만, a가 커진다고 해서 오차가 없어지지는 않는다.

2 | 시그모이드 함수

- b값에 따른 오차의 그래프는 그림 5-6과 같음

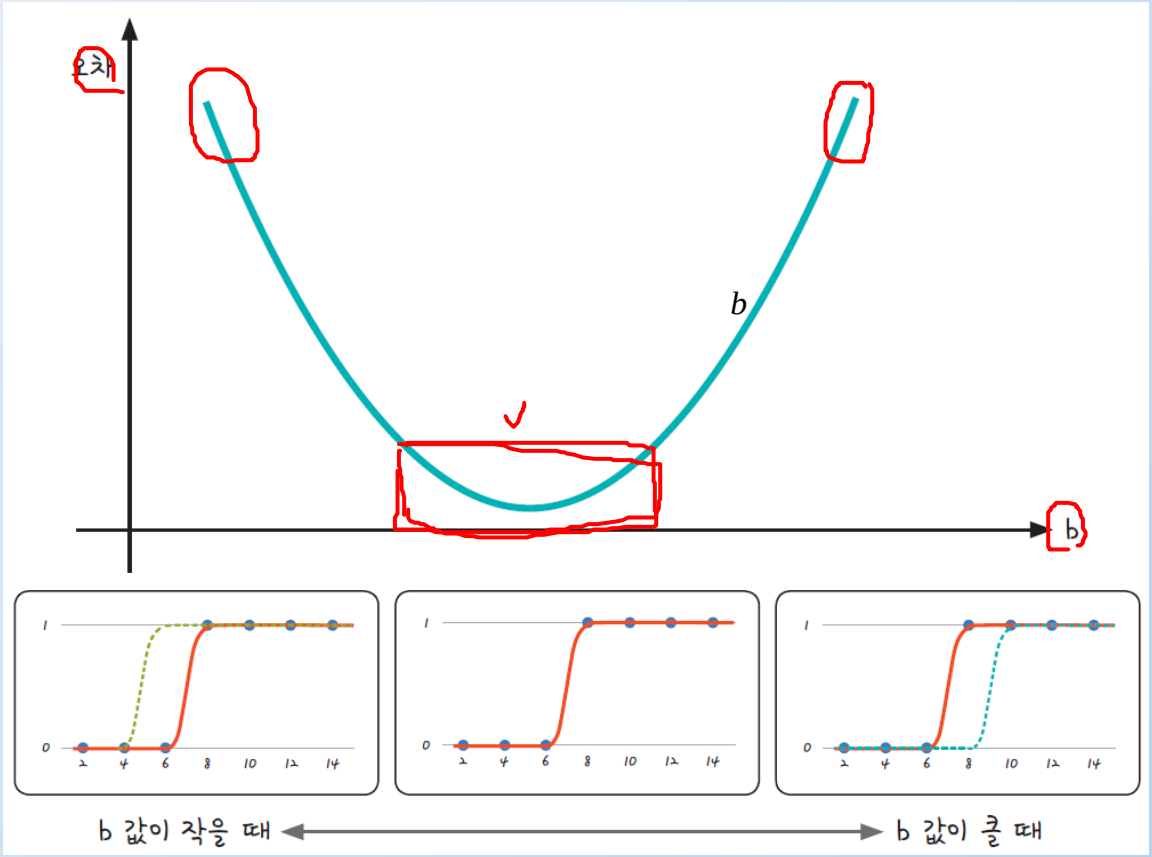


그림 5-6 b와 오차와의 관계: b 값이 너무 작아지거나 커지면 오차도 이에 따라 커진다.

3 | 오차 공식

- 경사 하강법은 먼저 오차를 구한 다음 오차가 작은 쪽으로 이동시키는 방법
- 예측 값과 실제 값의 차이, 즉 오차를 구하는 공식이 필요함

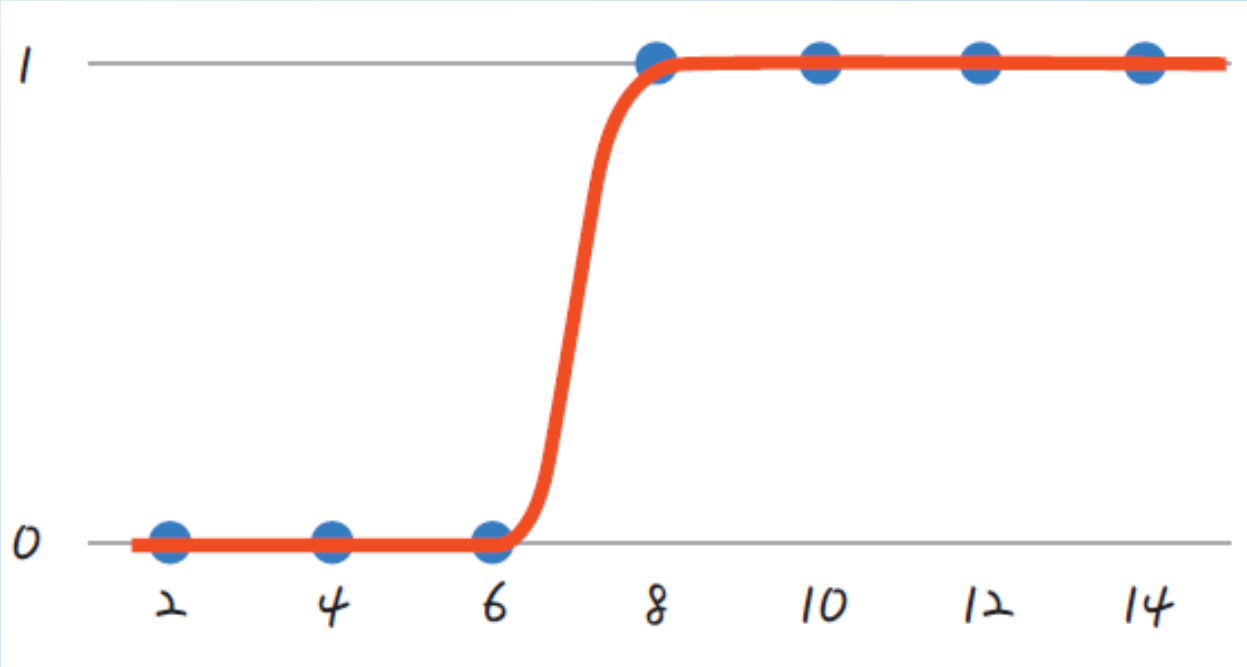


그림 5-7 시그모이드 함수 그래프

3 | 오차 공식

- 시그모이드 함수의 특징은 y 값이 0과 1 사이라는 것임
- 실제 값이 1일 때 예측 값이 0에 가까워지면 오차가 커짐
- 반대로, 실제 값이 0일 때 예측 값이 1에 가까워지는 경우에도 오차는 커짐
- 이를 공식으로 만들 수 있게 해 주는 함수가 바로 로그 함수임

4 | 로그 함수

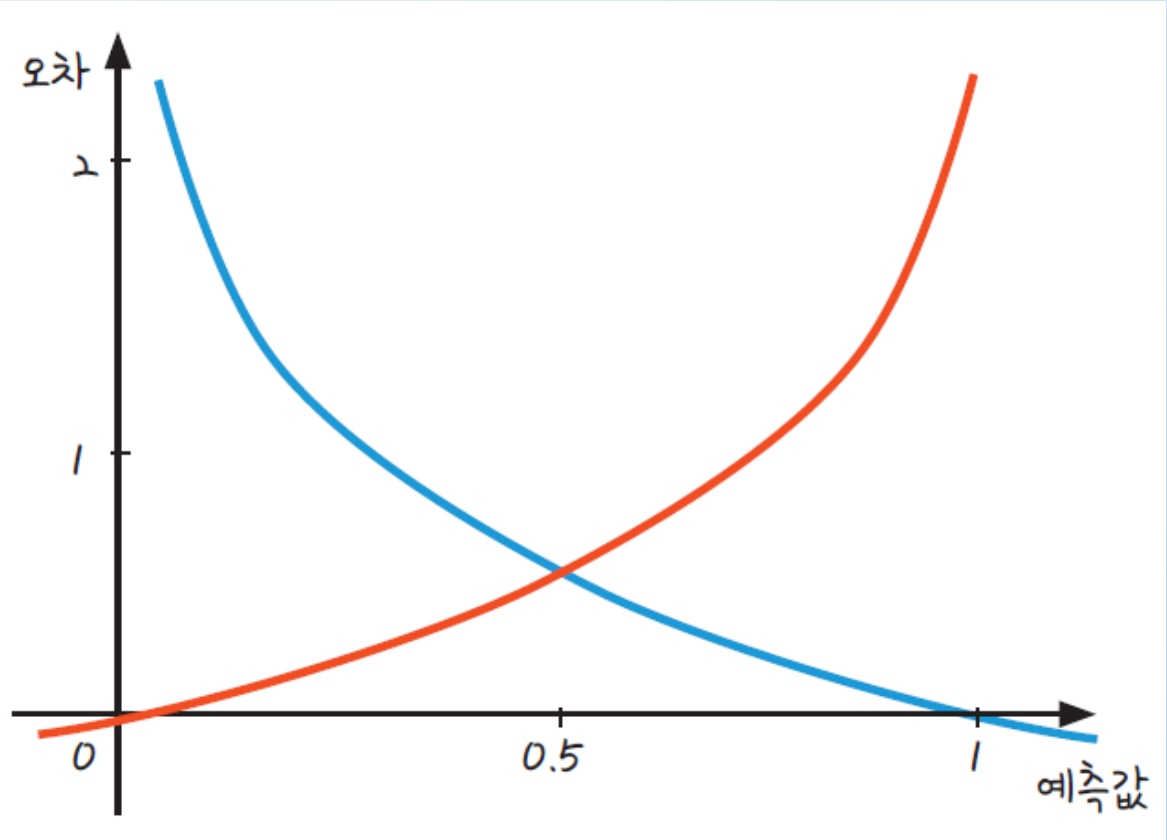


그림 5-8 실제 값이 1일 때(파란색)와 0일 때(빨간색) 로그 함수 그래프

4 | 로그 함수

- 파란색 선은 실제 값이 1일 때 사용할 수 있는 그래프임
- 예측 값이 1일 때 오차가 0이고, 반대로 예측 값이 0에 가까울수록 오차는 커짐
- 빨간색 선은 반대로 실제 값이 0일 때 사용할 수 있는 함수임
- 예측 값이 0일때 오차가 없고, 1에 가까워질수록 오차가 매우 커짐

4 | 로그 함수

- 파란색과 빨간색 그래프의 식은 각각 $-\log h$ 와 $-\log(1-h)$
- 실제 값이 1일 때는 $-\log h$ 그래프를 쓰고, 0일 때는 $-\log(1-h)$ 그래프를 써야 함

$$-\{\underbrace{y\_data \log h}_{\text{A}} + \underbrace{(1-y\_data)\log(1-h)}_{\text{B}}\}$$

- 실제 값을 y_data 라 할 때, 이 값이 1이면 B 부분이 없어짐
- 반대로 0이면 A부분이 없어짐
- 실제 값에 따라 빨간색 그래프와 파란색 그래프를 각각 사용할 수 있게 됨

5 | 코딩으로 확인하는 로지스틱 회귀

- 로지스틱 회귀를 위해서는
  - 시그모이드 함수를 사용한다는 것
  - 0부터 1사이의 값을 가지는 특성 때문에 로그 함수를 함께 써야 한다는 것
- 이제 경사 하강법을 이용해 a와 b의 최적 값을 찾는 과정을 해 볼 차례임

```
data = [[2, 0], [4, 0], [6, 0], [8, 1], [10, 1], [12, 1], [14, 1]]


x_data = [i[0] for i in data]    # 공부한 시간 데이터
y_data = [i[1] for i in data]    # 합격 여부
```

5 | 코딩으로 확인하는 로지스틱 회귀

```
import matplotlib.pyplot as plt


plt.scatter(x_data, y_data)
plt.xlim(0, 15)
plt.ylim(-.1, 1.1)
```

5 | 코딩으로 확인하는 로지스틱 회귀

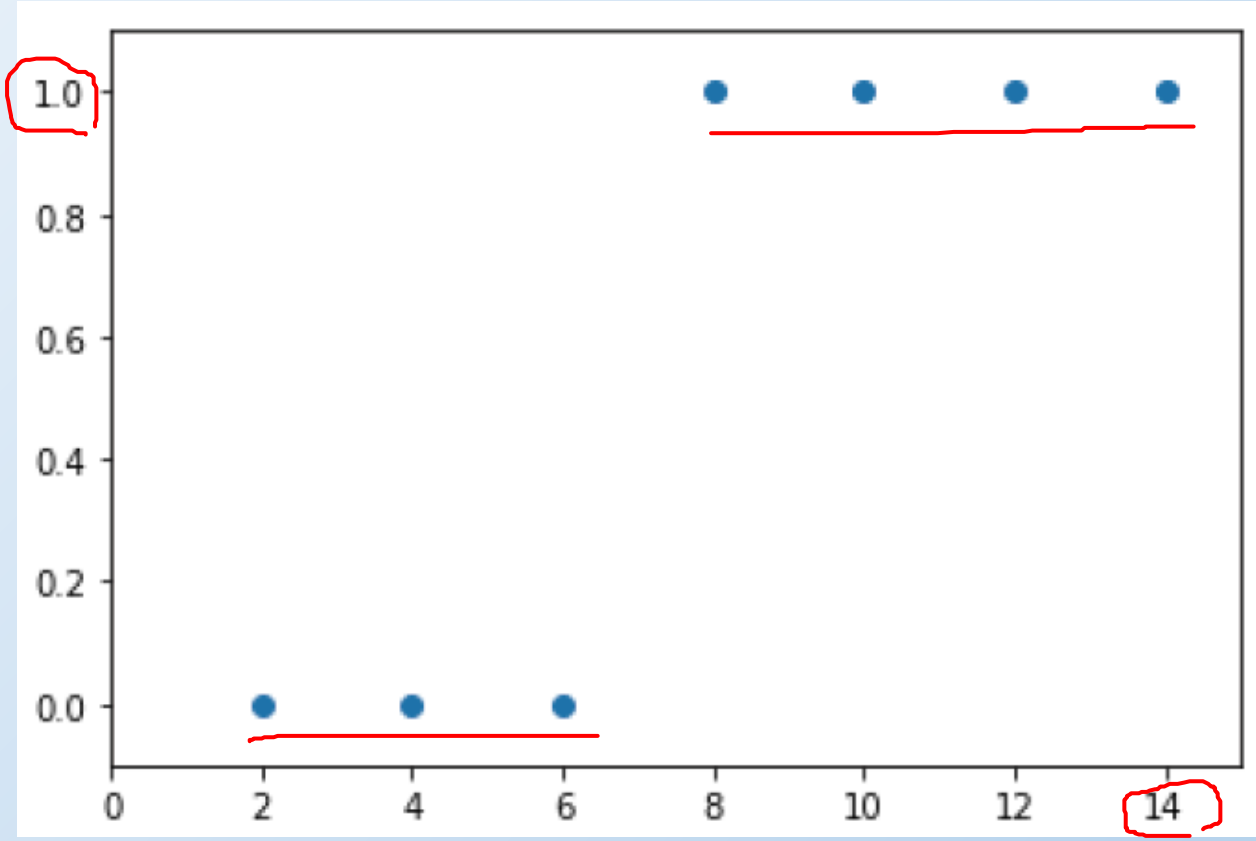


그림 5-9 공부 시간에 따른 합격 여부를 그래프로 나타낸 모습

5 | 코딩으로 확인하는 로지스틱 회귀

- 기울기와 a 와 절편 b 의 값을 초기화하고 학습률을 정함
- 학습률은 임의로 0.05로 정함

```
a = 0
b = 0


lr = 0.05  # 학습률
```

```
def sigmoid(x):                          # sigmoid라는 이름의 함수 정의
    return 1 / (1 + np.e ** (-x)) # 시그모이드 식의 형태 그대로 파이썬으로 옮김
```

5 | 코딩으로 확인하는 로지스틱 회귀

- 경사 하강법을 시행할 순서임
- a와 b로 편미분한 값(a_diff, b_diff)에 학습률(lr)을 곱하여 각각 업데이트 하는 방법은

  앞서 배운 선형 회귀와 같음
- 다만 오차를 구하는 함수가 다르므로 a_diff과 b_diff를 결정하는 식에 변화가 생김
- 로지스틱 회귀의 오차 함수는 조금 전에 배운 것을 이용해 다음과 같은 식으로 표시할 수 있음

$$-\frac{1}{n}\sum y \log h + (1-y)\log(1-h))$$

5 | 코딩으로 확인하는 로지스틱 회귀

- 이 식을 편미분한 후 파이썬 코드로 옮겨 놓은 결과가 다음과 같다는 것은 기억하기 바람

```
for i in range(2001):
    for x_data, y_data in data:
        # a에 관한 편미분. 앞서 정의한 sigmoid 함수 사용
        a_diff = x_data*(sigmoid(a*x_data + b) - y_data)
        # b에 관한 편미분
        b_diff = sigmoid(a*x_data + b) - y_data
        # a를 업데이트 하기 위해 a - diff에 학습률 lr을 곱한 값을 a에서 뺌
        a = a - lr * a_diff
        # b를 업데이트 하기 위해 b - diff에 학습률 lr을 곱한 값을 b에서 뺌
        b = b - lr * b_diff
```

5 | 코딩으로 확인하는 로지스틱 회귀

코드 5-1 코딩으로 확인하는 로지스틱 회귀

- 예제 소스: deeplearnig_class/05_Logistic_regression.ipynb

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# 공부 시간 X와 합격 여부 Y의 리스트 만들기
data = [[2, 0], [4, 0], [6, 0], [8, 1], [10, 1], [12, 1], [14, 1]]


x_data = [i[0] for i in data]
y_data = [i[1] for i in data]
```

5 | 코딩으로 확인하는 로지스틱 회귀

```
# 그래프로 나타내기
plt.scatter(x_data, y_data)
plt.xlim(0, 15)
plt.ylim(-.1, 1.1)


# 기울기 a와 절편 b의 값 초기화
a = 0
b = 0


#학습률
lr = 0.05
```

5 | 코딩으로 확인하는 로지스틱 회귀

```
# 시그모이드 함수 정의
def sigmoid(x):
    return 1 / (1 + np.e ** (-x))

# 경사 하강법 실행
# 1,000번 반복될 때마다 각 x_data 값에 대한 현재의 a 값, b 값 출력
for i in range(2001):
    for x_data, y_data in data:
        a_diff = x_data*(sigmoid(a * x_data + b) - y_data)
        b_diff = sigmoid(a * x_data + b) - y_data
        a = a - lr * a_diff
        b = b - lr * b_diff
        if i % 1000 == 0:
            print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))
```

5 | 코딩으로 확인하는 로지스틱 회귀

```
    # 앞서 구한 기울기와 절편을 이용해 그래프 그리기
    plt.scatter(x, y)
    plt.xlim(0, 15)
    plt.ylim(-.1, 1.1)
    x_range = (np.arange(0, 15, 0.1))   # 그래프로 나타낼 x 값의 범위 정하기
    plt.plot(np.arange(0, 15, 0.1), np.array([sigmoid(a * x + b)
for x in x_range]))
    plt.show( )
```

5 | 코딩으로 확인하는 로지스틱 회귀

실행 결과

```
epoch=0, 기울기=-0.0500, 절편=-0.0250
epoch=0, 기울기=-0.1388, 절편=-0.0472
epoch=0, 기울기=-0.2268, 절편=-0.0619
epoch=0, 기울기=0.1201, 절편=-0.0185
epoch=0, 기울기=0.2374, 절편=-0.0068
epoch=0, 기울기=0.2705, 절편=-0.0040
epoch=0, 기울기=0.2860, 절편=-0.0029
epoch=1000, 기울기=1.4978, 절편=-9.9401
epoch=1000, 기울기=1.4940, 절편=-9.9411
epoch=1000, 기울기=1.4120, 절편=-9.9547
epoch=1000, 기울기=1.4949, 절편=-9.9444
epoch=1000, 기울기=1.4982, 절편=-9.9440
```

5 | 코딩으로 확인하는 로지스틱 회귀

```
epoch=1000, 기울기=1.4984, 절편=-9.9440
epoch=1000, 기울기=1.4985, 절편=-9.9440
epoch=2000, 기울기=1.9065, 절편=-12.9489
epoch=2000, 기울기=1.9055, 절편=-12.9491
epoch=2000, 기울기=1.8515, 절편=-12.9581
epoch=2000, 기울기=1.9057, 절편=-12.9514
epoch=2000, 기울기=1.9068, 절편=-12.9513
epoch=2000, 기울기=1.9068, 절편=-12.9513
epoch=2000, 기울기=1.9068, 절편=-12.9513
```

5 | 코딩으로 확인하는 로지스틱 회귀

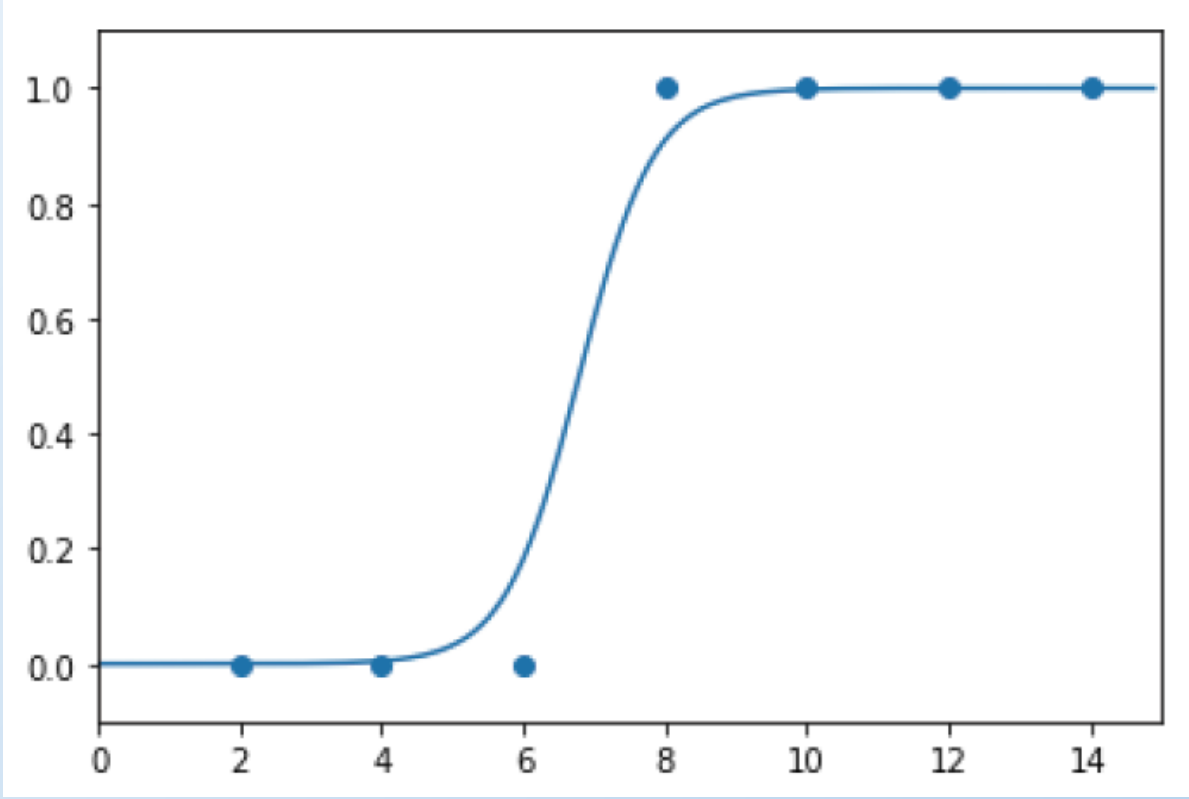


그림 5-10 시그모이드 형태로 출력된 그래프

5 | 코딩으로 확인하는 로지스틱 회귀

- 시그모이드 형태의 함수가 잘 만들어지도록 a 와 b의 값이 수렴된 것을 알 수 있음
- 만약 여기에 입력 값이 추가되어 세 개 이상의 입력 값을 다룬다면 시그모이드 함수가 아니라 소프트맥스(softmax)라는 함수를 써야 함

6 | 로지스틱 회귀에서 퍼셉트론으로

- 입력 값을 통해 출력 값을 구하는 함수 는 다음과 같이 표현할 수 있음 y

$$y = a_1 x_1 + a_2 x_2 + b$$

- 입력 값 :

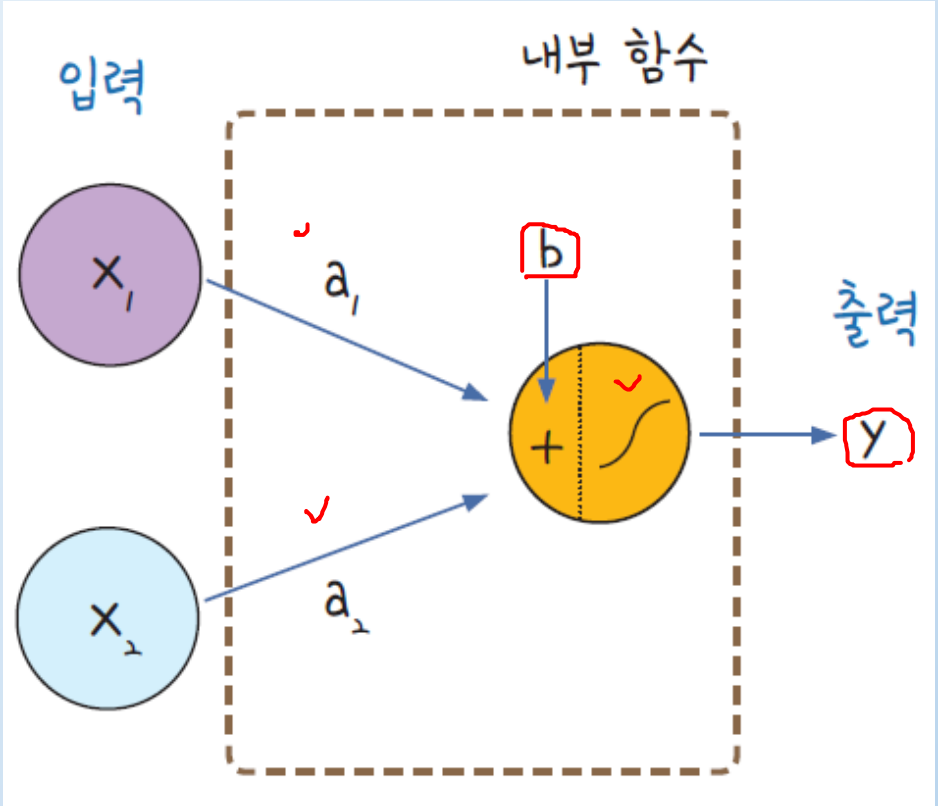
  우리가 가진 값은 x_1 과 x_2

- 출력 값 :

  계산으로 얻는 값 y

- 즉, 출력 값을 구하려면 값 값 그리고 값이 필요함 a_1 a_2 b

6 | 로지스틱 회귀에서 퍼셉트론으로



$$y = a_1 x_1 + a_2 x_2 + b$$

그림 5-11 로지스틱 회귀를 퍼셉트론 방식으로 표현한 예

정 리학 습

1 | 로지스틱 회귀의 정의

2 | 시그모이드 함수

3 | 오차 공식

4 | 로그 함수

5 | 코딩으로 확인하는 로지스틱 회귀

6 | 로지스틱 회귀에서 퍼셉트론으로

다음 수업

파이썬의 함수 부분을 학습