혼자 공부하는 머신러닝 + 딥러닝

CH04. 다양한 분류 알고리즘

인공지능공학과 12223557 여예진

목차

1

K-NN 분류기의 확률 예측

K-NN 분류기

= 특정 입력 데이터가 주어졌을 때 입력 데이터와 가까운 k개의 데이터를 이용하여 예측하는 모형

데이터

$$(\tilde{x}_i, y_i), \quad i = 1, \ldots, n$$

p차원(연속형) 입력 변수

$$\tilde{x}_i = (x_{1i}, x_{2i}, \ldots, x_{pi})^t \in \mathbb{R}^p$$

출력 변수

$$y_i \in \{1, 2, \ldots, J\}$$

J : 카테고리 개수

K-NN 분류기의 예측

$$\hat{y}(\tilde{x}) = \underset{j \in \{1,2,\ldots,J\}}{\operatorname{argmax}} \frac{1}{k} \sum_{i:\tilde{x}_i \in N_k(\tilde{x})} I(y_i = j)$$

훈련데이터 준비하기
판다스 사용

```
# 훈련데이터 준비하기 - 판다스 사용

import pandas as pd
# read_csv() = csv파일 -> 데이터프레임으로 변환
fish = pd.read_csv('https://bit.ly/fish_csv_data')
# head() = 데이터프레임 객체를 위에서부터 n열 반환 (기본값:n=5)
fish.head()
```

	Species	Weight	Length	Diagonal	Height	Width
0	Bream	242.0	25.4	30.0	11.5200	4.0200
1	Bream	290.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	26.5	31.1	12.3778	4.6961
3	Bream	363.0	29.0	33.5	12.7300	4.4555
4	Bream	430.0	29.0	34.0	12.4440	5.1340

```
# unique() = (데이터프레임에서) 데이터에 고유값들이 어떤 종류가 있는지 알고 싶을 때 사용
print(pd.unique(fish['Species']))
```

```
['Bream' 'Roach' 'Whitefish' 'Parkki' 'Perch' 'Pike' 'Smelt']
```

훈련데이터 준비하기

입력데이터 + 타깃데이터 만들기

```
# 입력데이터 만들기 - 데이터프레임에서 원하는 열을 리스트로 나열하여 열 선택하기
fish_input = fish[['Weight', 'Length', 'Diagonal', 'Height', 'Width']].to_numpy()
# 입력데이터에 5개의 특성 잘 저장되었는지 확인하기
fish_input[:5]
```

```
array([[242.    ,  25.4   ,  30.    ,  11.52  ,   4.02  ],
       [290.    ,  26.3   ,  31.2   ,  12.48  ,   4.3056],
       [340.    ,  26.5   ,  31.1   ,  12.3778,   4.6961],
       [363.    ,  29.    ,  33.5   ,  12.73  ,   4.4555],
       [430.    ,  29.    ,  34.    ,  12.444 ,   5.134 ]])
```

```
# 타깃데이터 만들기
fish_target = fish['Species'].to_numpy()
```

클래스 7개
-> 다중 분류 : 타깃 데이터에 2개 이상의 클래스가 포함된 문제

데이터를 훈련세트, 테스트세트로 나누기

train_test_split()

```
# 데이터를 훈련세트, 테스트세트로 나누기 - train_test_split()
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(fish_input, fish_target, random_state=42)
```

훈련세트, 테스트세트 표준화 전처리하기

StandardScaler 클래스

```
# 훈련세트, 테스트세트 표준화 전처리하기 - StandardScaler클래스
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

K-NN 분류기의 확률 예측

predict_proba()

```
# 모델 훈련시키고 점수 확인하기
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(train_scaled, train_target)
print(kn.score(train_scaled, train_target))
print(kn.score(test_scaled, test_target))
```

```
0.8907563025210085
0.85
```

In 사이킷런
- 문자열로 된 타깃값을 그대로 사용 가능
- 알파벳 순으로 자동 정렬

```
# KNeighborsClassifier에서 정렬된 타깃값 출력하기 - classes_ 속성
print(kn.classes_)
```

```
['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']
```

```
# 테스트세트의 5개 샘플의 타깃값 예측하기
print(kn.predict(test_scaled[:5]))
```

```
['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']
```

K-NN 분류기의 확률 예측

predict_proba()

```
# 5개 샘플 예측에 대한 클래스별 확률값 출력하기 - predict_proba()
import numpy as np
proba = kn.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=4))
```

predict_proba() : 클래스별 확률값 반환

```
[[0.     0.     1.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     1.     0.    ]
 [0.     0.     0.     1.     0.     0.     0.    ]
 [0.     0.     0.6667 0.     0.3333 0.     0.    ]
 [0.     0.     0.6667 0.     0.3333 0.     0.    ]]
```

이웃한 샘플의 클래스 비율이므로 항상 정해진 확률만 출력한다.

```
# 네 번째 샘플의 최근접 이웃의 클래스 확인하기
distances, indexes = kn.kneighbors(test_scaled[3:4])
print(train_target[indexes])
```

```
[['Roach' 'Perch' 'Perch']]
```

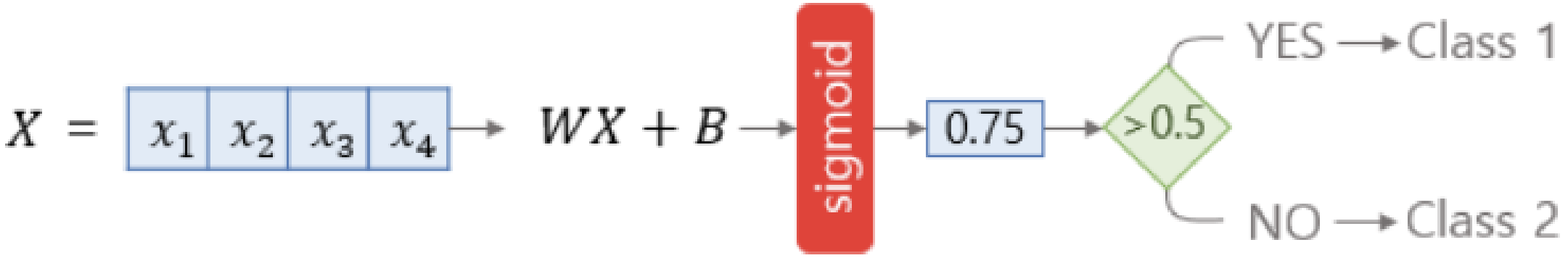
2

로지스틱 회귀의 이진 분류

로지스틱 회귀

= 이진 분류를 풀기 위한 분류 알고리즘

- 주로 **이진 분류** 문제를 해결하는데 사용된다.
- **시그모이드 함수** : 선형 방정식의 출력을 0과 1 사이의 값으로 압축한다.



가설 : $H(X) = sigmoid(WX + B)$

훈련데이터 준비하기

불리언 인덱싱 이용

```
# 도미(bream), 빙어(smelt의 행만 골라내기 - 불리언 인덱싱 이용
bream_smelt_indexes = (train_target == 'Bream') | (train_target == 'Smelt')
train_bream_smelt = train_scaled[bream_smelt_indexes]
target_bream_smelt = train_target[bream_smelt_indexes]
```

로지스틱 회귀 모델 훈련하기

LogisticRegression 클래스

```
# 위 데이터로 로지스틱 회귀 모델 훈련하기 - LogisticRegression클래스

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_bream_smelt, target_bream_smelt)
```

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

샘플 예측 및 클래스별 확률값 출력하기

predict() / predict_porba() 사용

```
# train_bream_smelt에 있는 처음 5개 샘플 예측하기
print(lr.predict(train_bream_smelt[:5]))
```

```
['Bream' 'Smelt' 'Bream' 'Bream' 'Bream']
```

```
# 5개 샘플 예측에 대한 클래스별 확률값 출력하기 - predict_proba()
print(lr.predict_proba(train_bream_smelt[:5]))
```

```
[[0.99759855 0.00240145]
 [0.02735183 0.97264817]
 [0.99486072 0.00513928]
 [0.98584202 0.01415798]
 [0.99767269 0.00232731]]
```

```
# 정렬된 타깃값 출력하기 - classes_속성
print(lr.classes_)
```

```
['Bream' 'Smelt']
```

```
# 로지스틱 회귀가 학습한 계수 확인하기
print(lr.coef_, lr.intercept_)
```

```
[-0.4037798  -0.57620209 -0.66280298 -1.01290277 -0.73168947]] [-2.16155132]
```

Z값 계산 및 확률 변환하기

decision_function() / expit() 사용

```
# LogisticRegression모델로 z값 계산하기 - decision_function() = 양성 클래스에 대한 z값 반환

# train_bream_smelt의 처음 5개 샘플의 z값 출력하기
decisions = lr.decision_function(train_bream_smelt[:5])
print(decisions)
```

```
-6.02927744  3.57123907 -5.26568906 -4.24321775 -6.0607117 ]
```

```
# decisions배열의 z값을 확률로 변환하기 - expit() = scipy라이브러리의 시그모이드 함수
from scipy.special import expit
print(expit(decisions))
```

시그모이드 함수

```
0.00240145 0.97264817 0.00513928 0.01415798 0.00232731]
```

```
# 5개 샘플 예측에 대한 클래스별 확률값 출력하기 - predict_proba()
print(lr.predict_proba(train_bream_smelt[:5]))
```

```
[0.9975985  0.00240145]
[0.0273518  0.97264817]
[0.9948607  0.00513928]
[0.9858420  0.01415798]
[0.9976726  0.00232731]]
```

Decision_function() : 양성 클래스에 대한 z값 계산

3

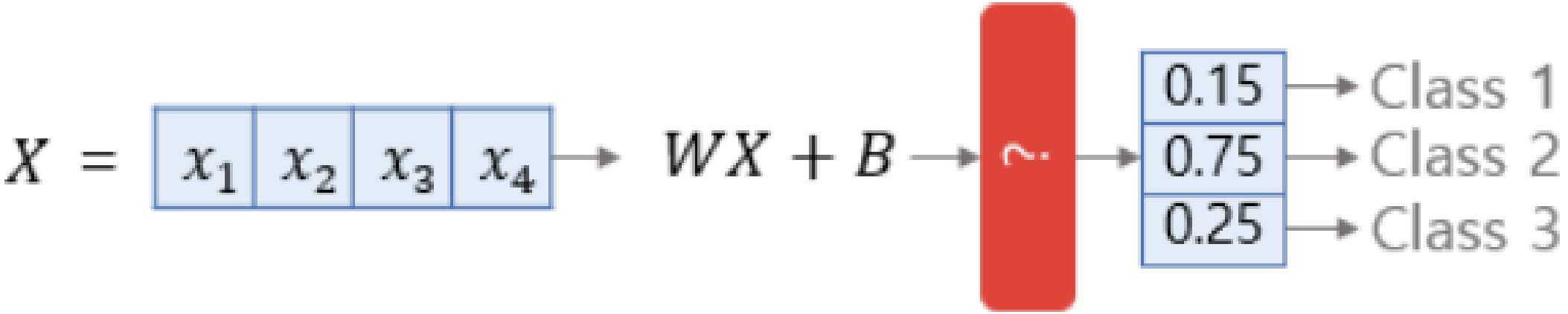
소프트맥스 회귀의 다중 분류

소프트맥스 회귀

= 다중 분류를 풀기 위한 분류 알고리즘

소프트맥스 회귀 : 로지스틱 회귀에서 소프트맥스 함수를 사용하여 다중 클래스 분류 문제를 해결하는 경우

- 주로 **다중 클래스분류** 문제를 해결하는데 사용된다.
- **소프트맥스 함수** : 다중 분류에서 여러 선형 방정식의 출력 결과를 정규화하여 합이 1이 되도록 만든다.



가설 : $H(X) = softmax(WX + B)$

소프트맥스 회귀 모델 훈련하기

LogisticRegression 클래스

```
# 7개의 생선을 분류하는 다중 분류 모델 훈련하기 - LogisticRegression클래스
lr = LogisticRegression(C=20, max_iter=1000)
lr.fit(train_scaled, train_target)
print(lr.score(train_scaled, train_target))
print(lr.score(test_scaled, test_target))
```

```
.9327731092436975
.925
```

샘플 예측하기

predict() 사용

```
# 테스트 세트의 처음 5개 샘플에 대한 예측 출력하기
print(lr.predict(test_scaled[:5]))
```

```
'Perch' 'Smelt' 'Pike' 'Roach' 'Perch']
```

클래스별 확률값 출력하기

predict_porba() 사용

```
# 테스트 세트 처음 5개 샘플에 대한 예측 확률 출력하기
proba = lr.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=3))
```

```
[0.     0.014 0.841 0.    0.136 0.007 0.003]
[0.     0.003 0.044 0.    0.007 0.946 0.   ]
[0.     0.    0.034 0.935 0.015 0.016 0.   ]
[0.011 0.034 0.306 0.007 0.567 0.    0.076]
[0.     0.    0.904 0.002 0.089 0.002 0.001]]
```

```
# 클래스 정보 확인하기 - classes_속성
print(lr.classes_)
```

```
'Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']
```

```
# 다중 분류의 선형 방정식 coef_, interecpt_의 크기 출력하기
print(lr.coef_.shape, lr.intercept_.shape)
# 출력값) 7:클래스 개수 / 5:특성 개수
```

```
7, 5) (7,)
```

Z값 계산 및 확률 변환하기

decision_function() / softmax() 사용

```
# LogisticRegression모델로 z값 계산하기 - decision_function() = z값 반환

# z1~z7값 구하기
decision = lr.decision_function(test_scaled[:5])
print(np.round(decision, decimals=2))
```

```
[ -6.5    1.03   5.16  -2.73   3.34   0.33  -0.63]
[-10.86   1.93   4.77  -2.4    2.98   7.84  -4.26]
[ -4.34  -6.23   3.17   6.49   2.36   2.42  -3.87]
[ -0.68   0.45   2.65  -1.19   3.26  -5.75   1.26]
[ -6.4   -1.99   5.82  -0.11   3.5   -0.11  -0.71]]
```

```
# decisions배열의 z값을 확률로 변환하기 - softmax() = scipy라이브러리의 소프트맥스 함수
from scipy.special import softmax
proba = softmax(decision, axis=1)
print(np.round(proba, decimals=3))
```

소프트맥스 함수

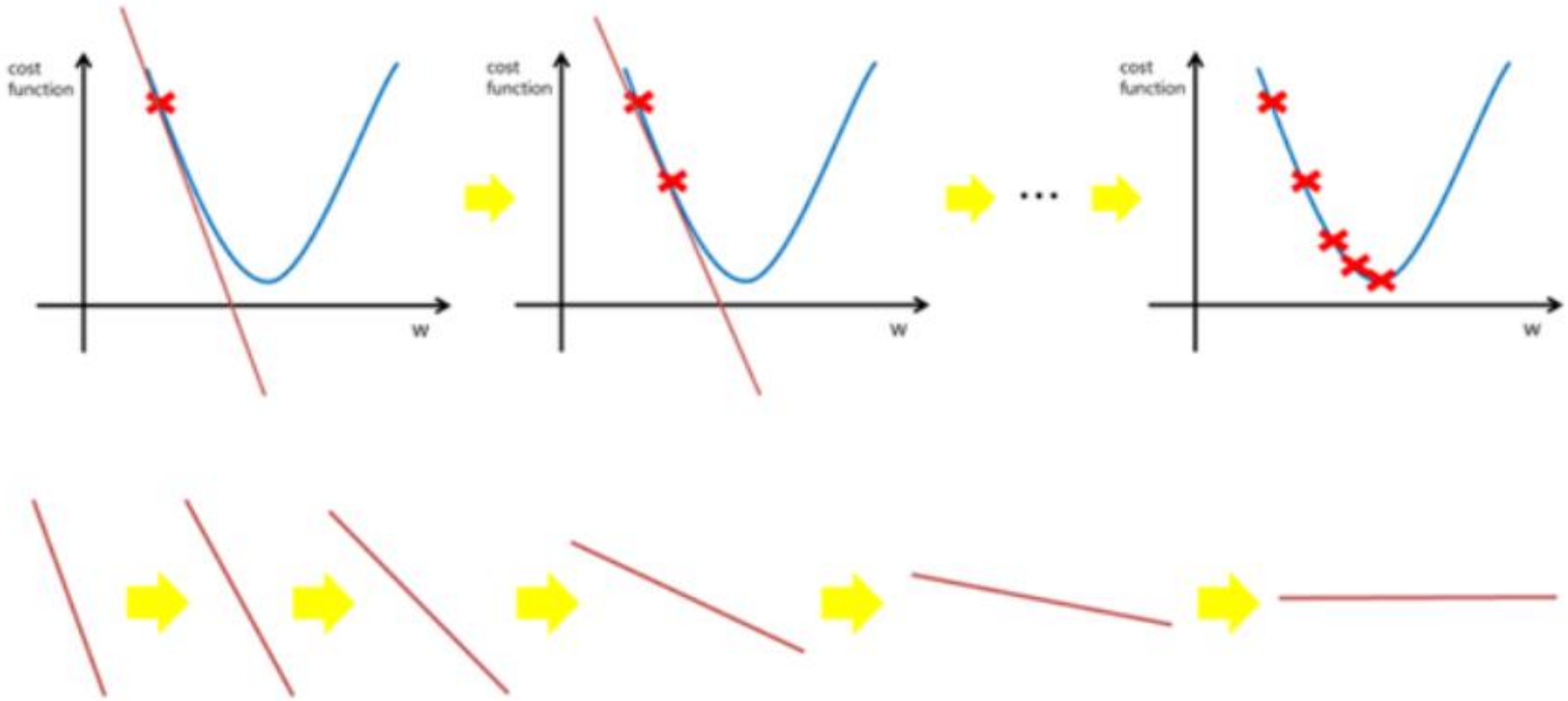
```
[0.     0.014 0.841 0.     0.136 0.007 0.003]
[0.     0.003 0.044 0.     0.007 0.946 0.    ]
[0.     0.    0.034 0.935 0.015 0.016 0.    ]
[0.011 0.034 0.306 0.007 0.567 0.    0.076]
[0.     0.    0.904 0.002 0.089 0.002 0.001]]
```

4

확률적 경사 하강법

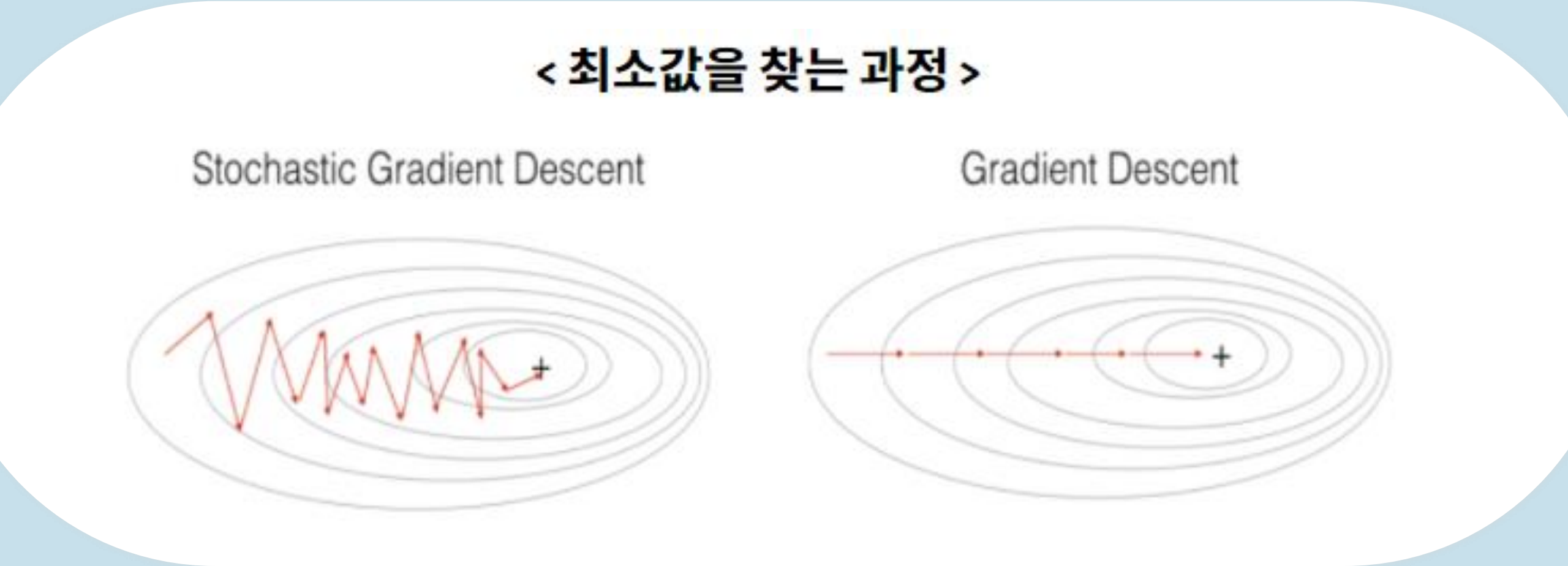
경사하강법(Gradient Descent)

- 손실을 줄이는 알고리즘
- **최적화** : 오류 값을 계산하고 해당 값을 최소화하기 위해 가중치를 변경하는 것
- **문제점** : 전체 데이터를 모두 사용해서 기울기를 계산 -> 학습 시간 많이 필요하다.



확률적 경사하강법(SGD, Stochastic Grandient Descent)

- 경사하강법의 문제점을 보완하기 위해 사용한다.
- 매 step에서 **한 개의 샘플**을 **무작위**로 선택하고 그 샘플의 기울기를 계산한다.



Thank You