7주차 진도 세션

손실 함수, 경사 하강법

5조 최대상, 심정훈, 조은정, 조예진

1. 손실 함수

손실 함수(Loss function)

정의 하나의 데이터 포인트에 대해, 실제 값과 예측 값의 차이를 표현하도록 계산하는 함수

손실 함수는 유일하지 않고, 장단점이 존재 → 문제 상황(분류, 회귀 등)에 맞게 선택해서 사용

$$L(\hat{y}, y)$$

$$L_{abs}(\hat{y}, y) = |\hat{y} - y|$$

$$L_{square}(\hat{y}, y) = (\hat{y} - y)^2$$

⋮

비용 함수(Cost function)

정의 전체 데이터에 대해, 실제 값과 예측 값의 차이를 표현하도록 계산하는 함수

- ❑ 손실 함수로부터 정의하는 것이 일반적 → 목적에 따라 잘 정의해서 사용
- ❑ 모델의 overfitting을 조절하기 위해 규제(Regularization)항 추가 가능
- ❑ 동일한 손실 함수에 대해 다른 비용 함수 존재 가능
- ❑ 비용 함수를 통해 모델을 학습 → 최적화
- ❑ 테스트 데이터에 대한 비용 함수를 통해 모델의 성능 측정 및 다른 모델과 비교
- ❑ 미분가능성(모델의 최적화 과정에서 비용 함수의 매개변수 미분 값을 이용)

$$Cost(f, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} L\left(\hat{y}_i, y_i\right)$$

$$Cost(f, \mathcal{D}) = \text{median}\left\{Loss\left(\hat{y}_i, y_i\right)\right\}_{i=1}^{n}$$

$$J(f, \mathcal{D}) = Cost(f, \mathcal{D}) + Regularizer(f)$$

손실 함수 vs. 비용 함수 vs. 목적 함수

목적 함수(Objective function) 모델의 학습에서 최적화하는 함수를 통칭하는 일반적인 개념

✓ 개념의 일반성: 손실 함수 → 비용 함수 → 목적 함수

예시 회귀문제에서 일반적으로 사용하는 손실 함수와 비용 함수(MSE)

$$LossFunction = (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$CostFunction = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \qquad (m = number\ of\ sample\ data)$$

⚠ 편의상 지금부터는 세 가지 용어를 엄밀히 구분하지 않고 혼용

대표적인 손실 함수

다행히 주로 많이 다루는 문제에 필요한 손실 함수가 이미 정의되어 있음

회귀(Regression)

$$MSE = \frac{1}{N}\sum_{i}^{N}(pred_i - target_i)^2$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i}^{N}(pred_i - target_i)^2}$$

$$MAE = \frac{1}{N}\sum_{i}^{N}|(pred_i - target_i)|$$

분류(Classification)

$$logloss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij})$$

N is the number of rows
M is the number of classes

Hinge loss: $\mathcal{L}(Y, \dot{Y}) = \frac{1}{N}\sum_{n=1}^{N}\max(0, 1 - \dot{y}_n \cdot y_n)$

크로스 엔트로피(Cross Entropy)

분류 문제와 딥러닝(Deep Learning)에서 대표적으로 사용하는 손실 함수

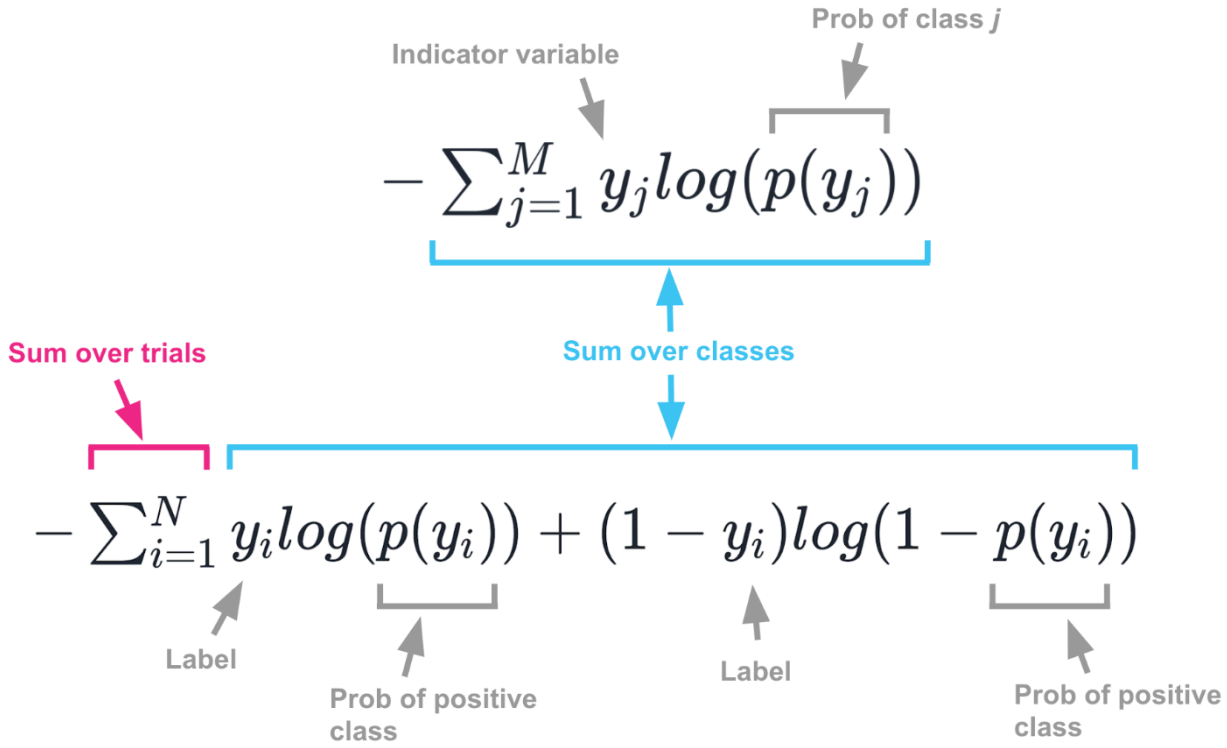
- 이진 분류 → Binary Cross Entropy
- 다중 분류 → (Multiclass) Cross Entropy

$$logloss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij})$$

N = number of sample data(rows)

M = number of classes

→ [M=2] Binary Cross Entropy(BCE)



Notation 차이로 인한 수식 혼동 주의 ! → 클래스와 확률 모두 벡터로 생각하면 편함 ☺

BCE 직접 계산 예시

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1 - t_j)\log(1 - p_j)\right]\right]$$

ID	Actual	Predicted probabilities
ID6	1	0.94
ID1	1	0.90
ID7	1	0.78
ID8	0	0.56
ID2	0	0.51
ID3	1	0.47
ID4	1	0.32
ID5	0	0.10

Corrected Probabilities
0.94
0.90
0.78
0.44
0.49
0.47
0.32
0.90

- 예측 확률: 클래스 1에 대한 확률
- 수정 확률:
  실제 타겟의 클래스로 예측할 확률

BCE 직접 계산 예시

Corrected Probabilities	Log
0.94	-0.0268721464
0.90	-0.0457574906
0.78	-0.1079053973
0.44	-0.3565473235
0.49	-0.30980392
0.47	-0.3279021421
0.32	-0.4948500217
0.90	-0.0457574906

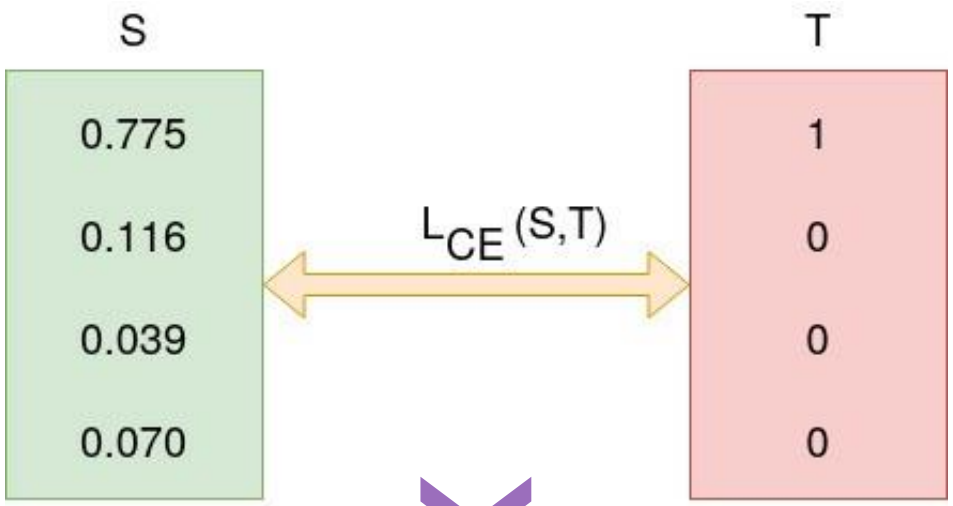
$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1 - t_j)\log(1 - p_j)\right]\right]$$

0.214…

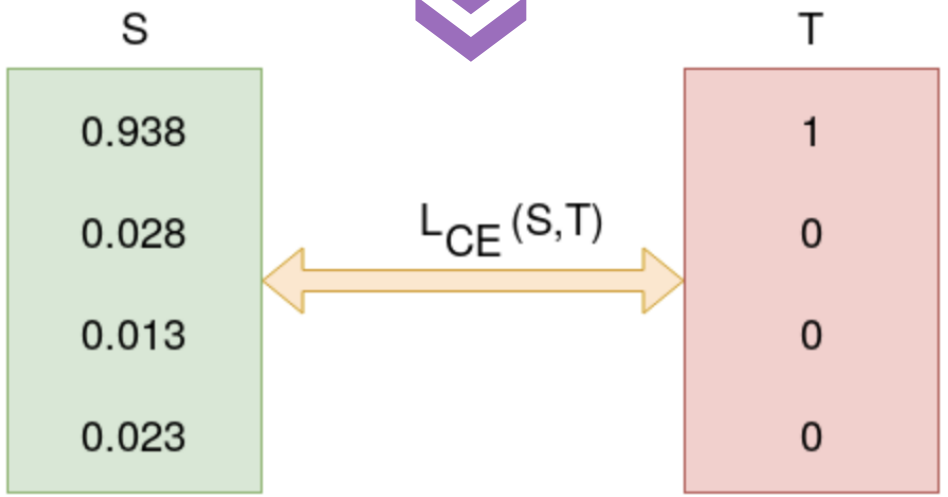
8개의 데이터에 대한 계산이므로, 총합을 8로 나누고 부호를 바꿔준다

[참고] 비슷한 계산 과정을 교재 p.204~206 에서 확인 가능

데이터 포인트에 대한 CE 직접 계산 및 모델 최적화 예시

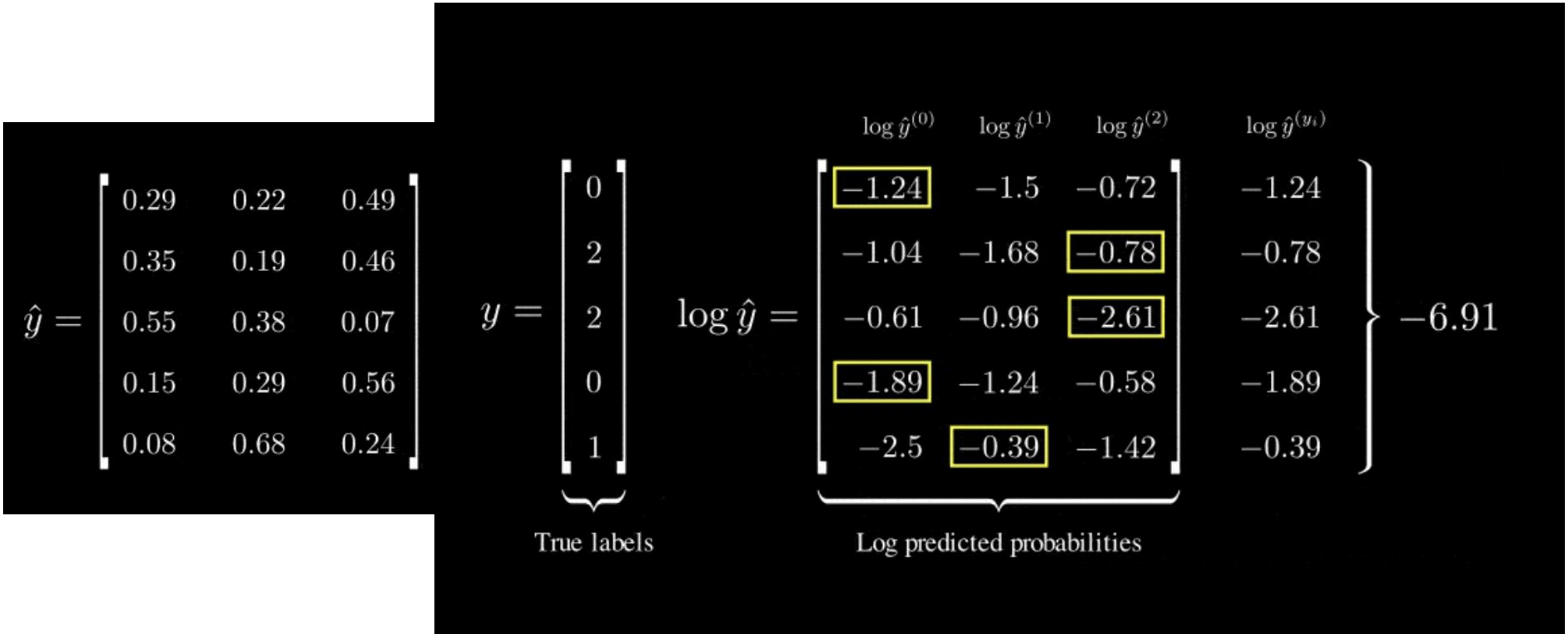


$$L_{CE} = -\sum_{i=1} T_i \log(S_i)$$
$$= -\left[1\log_2(0.775) + 0\log_2(0.126) + 0\log_2(0.039) + 0\log_2(0.070)\right]$$
$$= -\log_2(0.775)$$
$$= 0.3677$$



$$L_{CE} = -1\log_2(0.936) + 0 + 0 + 0$$
$$= 0.095$$

CE 계산의 개요



지금까지의 세 가지 예시에서 로그의 밑은 각각 10, 2, e로 다르게 계산
밑이 1보다 크기만 하면 손실 함수로서 최적화 가능

Cross Entropy와 Log Likelihood

우도(Likelihood)

관측된 표본 데이터의 분포가 모분포에서 가장 흔하게 관측된다(확률이 높다)는 가정
→ 미지의 모수를 통해 계산한 그 확률이 최대가 되도록 모수를 추정하고자 정의하는 개념

우도

$$\mathbb{P}(\mathcal{D}|\theta) = \prod_{i=1}^{n} \hat{y}_{\theta,i}^{y_i}(1-\hat{y}_{\theta,i})^{1-y_i}$$

로그 우도

$$\log \mathbb{P}(\mathcal{D}|\theta) = \sum_{i=1}^{n} \left(y_i \log \hat{y}_{\theta,i} + (1-y_i) \log (1-\hat{y}_{\theta,i}) \right)$$

음의 로그 우도(NLL loss)

$$l(\theta) = -\sum_{i=1}^{n} \left(y_i \log \hat{y}_{\theta,i} + (1-y_i) \log (1-\hat{y}_{\theta,i}) \right)$$

CE를 최소화하는 것은 우도를 최대화하는 것과 동치

Cross Entropy와 KL Divergence

KL Divergence(쿨백-라이블러 발산); $D_{KL}(P||Q)$

두 확률분포의 차이를 계산하는 데에 사용하는 함수
→ 어떤 분포에 근사하는 다른 분포를 사용할 경우 발생하는 정보 엔트로피의 차이

$$
\begin{aligned}
D_{KL}(P||Q) &= -\sum_x P(x)\log\left(\frac{Q(x)}{P(x)}\right) \\
&= -\sum_x P(x)\{\log Q(x) - \log P(x)\} \\
&= -\sum_x \{P(x)\log Q(x) - P(x)\log P(x)\} \\
&= -\sum_x P(x)\log Q(x) + \sum_x P(x)\log P(x) \\
&= H(P,Q) - H(P)
\end{aligned}
$$

학습 과정에서 샘플 데이터의 분포 P(x)는 고정
→ H(P)(엔트로피; 정보이론에서 불확실성의 척도)는 상수

H(P,Q)(크로스 엔트로피)를 최소화 → KLD를 최소화

모델이 예측하는 분포를 실제 분포에 최대한 근사

[참고]
https://hyunw.kim/blog/2017/10/27/KL_divergence.html

2. 경사 하강법

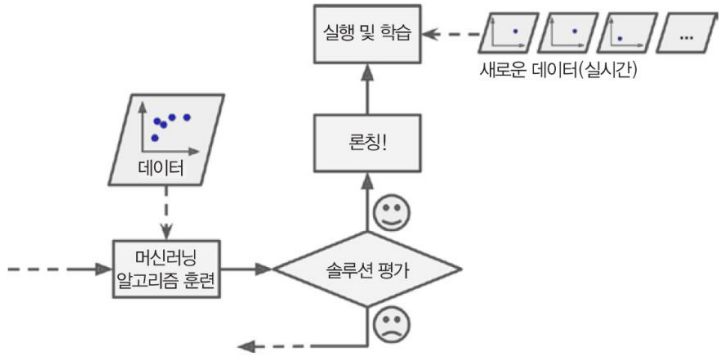
배치 학습 vs. 점진적 학습

배치 학습(오프라인 학습)

- 가용 데이터 모두 사용
- 시간, 자원 소모 큼 -> 보통 오프라인에서 수행
- 한 번 시스템을 훈련시킨 후 제품 시스템에 적용하면 더 이상의 학습없이 실행
- 많은 컴퓨팅 자원(CPU, 메모리 공간, 디스크 공간, 네트워크 IO 등)이 필요함 -> 큰 비용 발생
- 자원이 제한된 시스템(ex. 스마트폰, 화성 탐사 로버)에서 사용 시 문제 발생

점진적 학습(온라인 학습)

- 작은 단위의 데이터를 순차적으로 주입하여 시스템 훈련
- 시간, 비용이 적게 듦->즉시 학습 가능
- 연속적으로 데이터를 받고 빠른 변화에 스스로 적응해야 하는 시스템 (ex. 주식가격) 에 적합
- 컴퓨팅 자원이 제한된 경우에도 좋음
- 학습이 끝난 데이터는 삭제해도 무방 -> 공간 절약
- 대표적 알고리즘: 확률적 경사 하강법



경사 하강법(Gradient Descent)

경사 하강법이란?

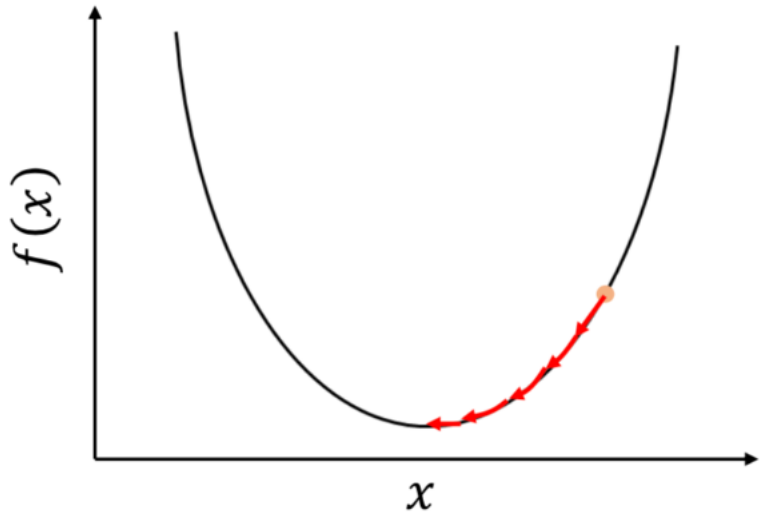
- 여러 종류의 문제에서 최적의 해법을 찾을 수 있는 일반적인 최적화 알고리즘

- 비용함수의 실제값과 예측값의 차이가 최소화되도록
매개변수를 갱신하는데 비용함수의 기울기를 사용하는 것

미분계수

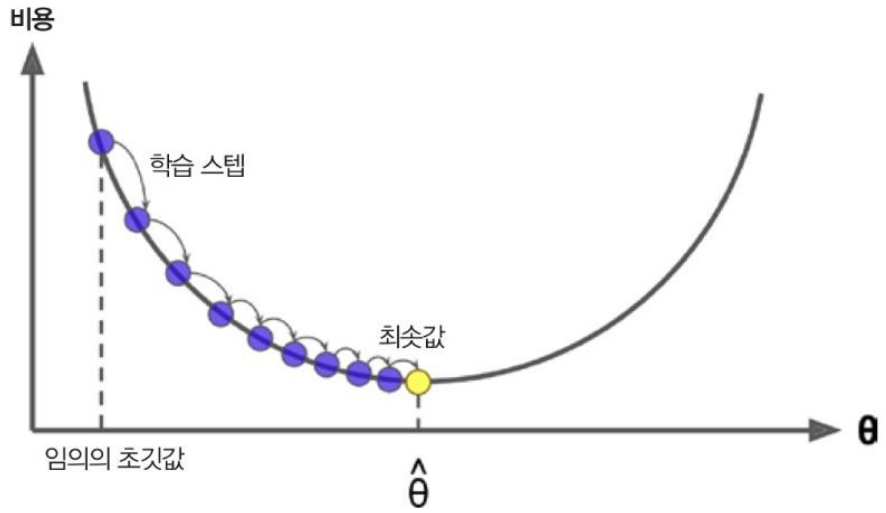
$$x_{i+1} = x_i - \alpha \frac{df}{dx}(x_i)$$

학습률 (learning rate)



경사 하강법(Gradient Descent)

경사 하강법이란?



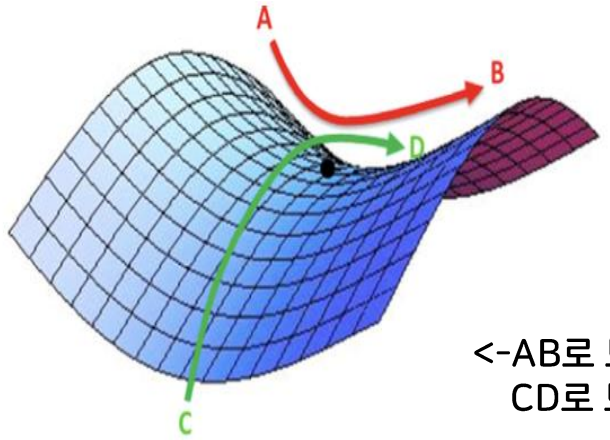
- 비용 함수를 최소화하기 위해 반복해서 파라미터 조정
- 임의의 θ값으로 시작(무작위 초기화)
- 비용함수의 현재 그래디언트를 계산, 감소하는 방향으로 진행
  -> 0이 되면 최소값에 도달한 것
- 종류 3가지 : 배치 경사 하강법, 확률적 경사 하강법,
  미니배치 경사 하강법

경사 하강법(Gradient Descent)

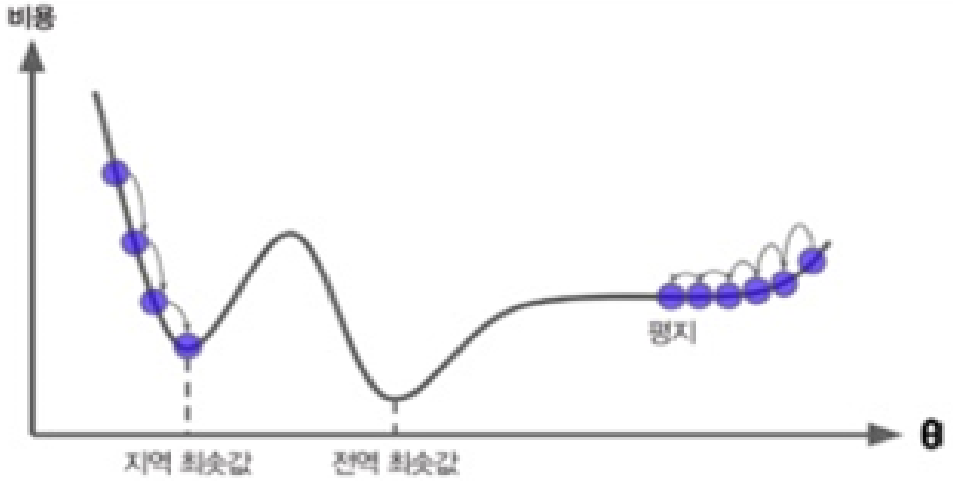
경사 하강법 진행할 때 주의할 점

함수의 기울기가 0이 되는 장소: 극솟값, 최솟값, 안장점

- 극솟값: 국소적인 최솟값,
  즉, 한정된 범위에서의 최솟값인 점
- 안장점: 어느 방향에서 보면 극댓값이고
  다른 방향에서 보면 극솟값이 되는 점



<-AB로 보면 검은 점이 지역 최소
CD로 보면 검은 점이 지역 최대



- 경사 하강법으로 도출한 값이 항상 최솟값인 것은 X
- 최솟값이 아닌 경우에도 기울기 정보로 활용 가능

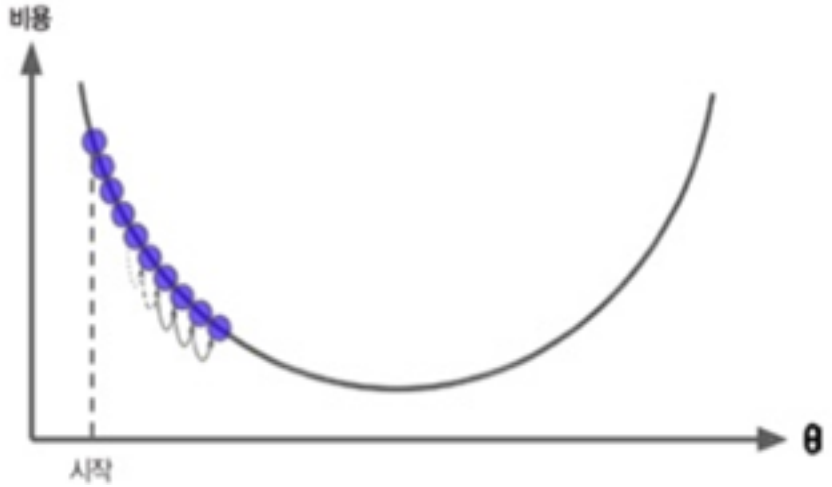
-전역 최소(global minimum): 함수 전체에서 가장 낮은 곳
-지역 최소(local minimum): 함수에서 부분적으로 낮은 곳
(무수히 많음)

경사 하강법(Gradient Descent) - 학습률(learning rate)

학습률(learning rate)

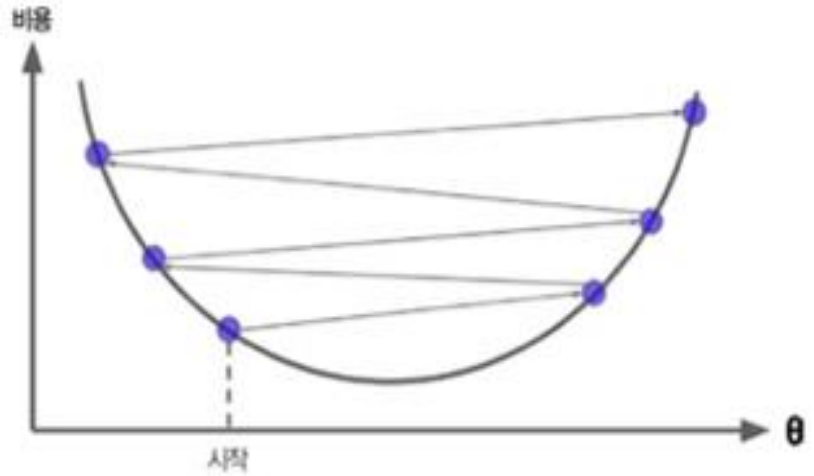
: 매 단계에 적용되는 현재 지점에서 다음 지점까지의 이동거리

i) 학습률이 너무 작은 경우



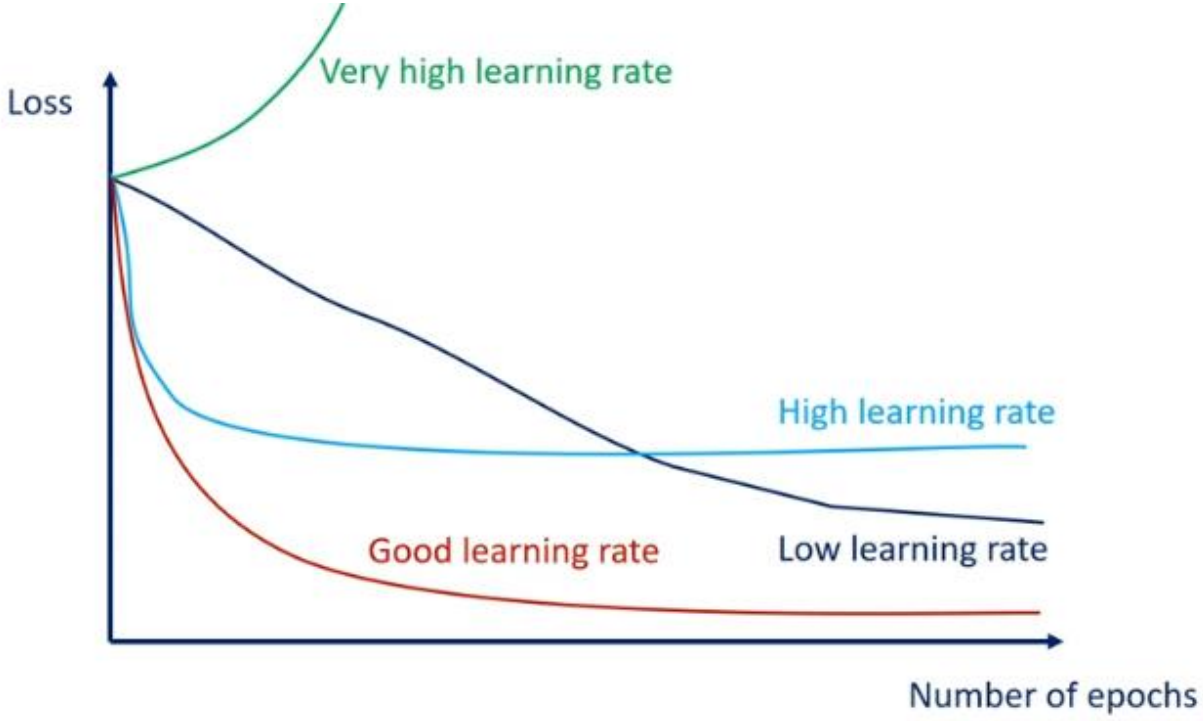
- 학습시간이 매우 오래 걸림
- 최저점에 도달하지 못함

ii) 학습률이 너무 큰 경우



- 데이터가 무질서하게 이탈
- 최저점에 수렴하지 못함

경사 하강법(Gradient Descent) - 학습률(learning rate)



* number of epochs: 전체 데이터 세트를 학습한 횟수
* loss: 손실 함수

-낮은 학습률(Low learning rate):

손실 함수가 선형의 형태를 보이면서 천천히 학습

-높은 학습률(High learning rate):

손실 함수가 지수 형태를 보이며,

구간에 따라 빠른 학습 혹은 정체

-매우 높은 학습률(Very high learning rate):

경우에 따라 손실을 오히려 증가시키는 상황 발생

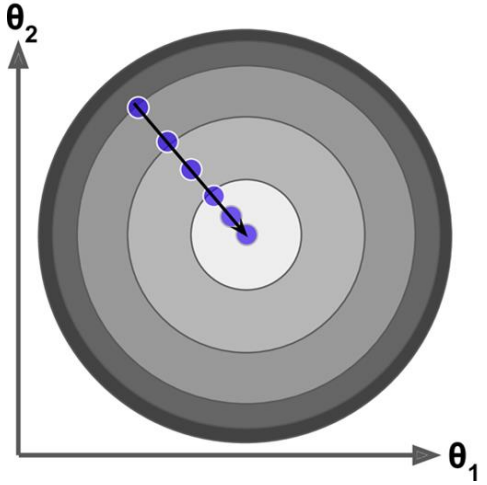
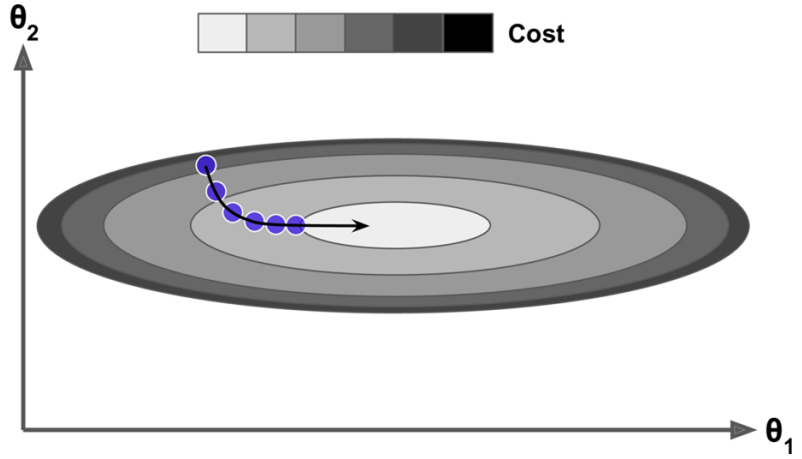
=>Good learning rate:

적절한 학습 곡선의 형태로, learning rate를

조절하면서 찾아야함!

경사 하강법(Gradient Descent) - Scaling

변수들의 스케일이 매우 다른 경우 생기는 문제

스케일이 동일한 경우(특성1=특성2)	스케일이 동일하지 않은 경우 (특성1<특성2)
	
최솟값에 빠르게 도달	시간이 오래 걸림

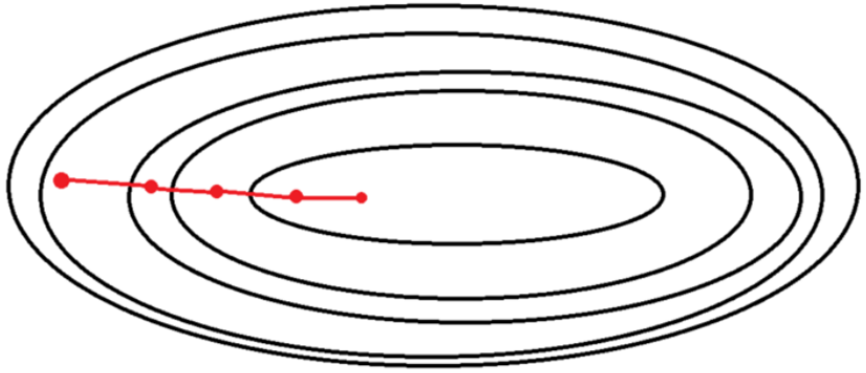
-> 반드시 모든 특성이 같은 스케일을 갖도록 조정 필요

(scikit-learn의 StandardScaler 사용)

배치 경사 하강법 (BGD, Batch Gradient Descent)이란?

: 매개변수를 한 번 갱신하는 데 전체 학습 데이터를 하나의 *배치로 묶어 사용하는 방법

* 배치 (Batch) : GPU가 한번에 처리하는 데이터의 묶음 -> 전체 학습 데이터



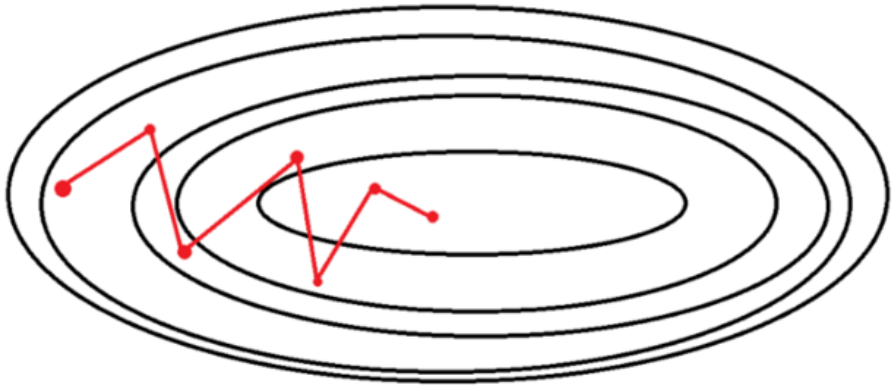
장점
- 적은 업데이트 횟수
- 수렴이 안정적
- GPU를 활용한 병렬처리에 유리

단점
- 계산 시간이 오래걸림
- 지역 최소(local minimum)에 수렴될 경우 탈출이 어려울 수 있음
- 많은 메모리 필요

확률적 경사 하강법 (SGD, Stochastic Gradient Descent)이란?

: 매개변수를 갱신하기 위해 무작위로 샘플링된 한 개의 데이터를 이용하여 경사 하강법을 진행하는 방법

* 전체 학습 데이터 중 무작위로 선택된 하나의 데이터를 이용하기 때문에 "확률적"

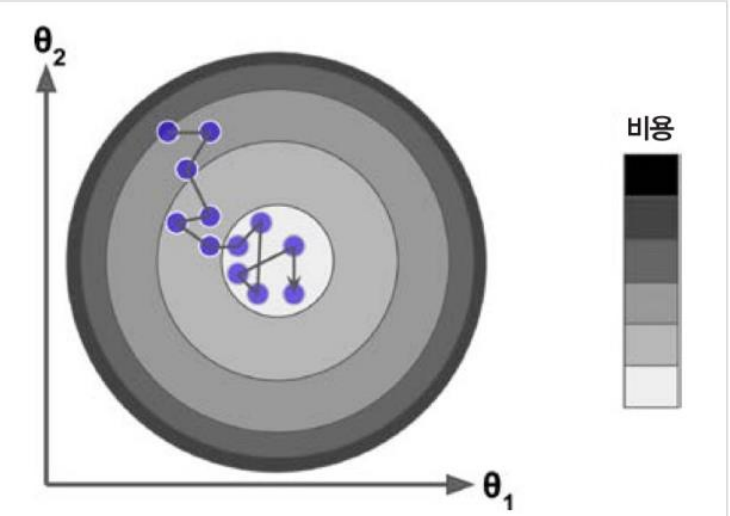


장점
- 배치 경사 하강법에 비해 적은 데이터로 학습, 빠른 속도
- 배치 경사 하강법보다 전역 최솟값을 찾을 가능성 높음

단점
- 배치 경사 하강법보다 불안정 (무작위성)
- 오차율이 크고, GPU의 성능을 전부 활용할 수 없음
- 전역 최솟값에 다다르지 못할 수 있음

무작위성을 해결하기 위해서 어떻게 해야 할까?

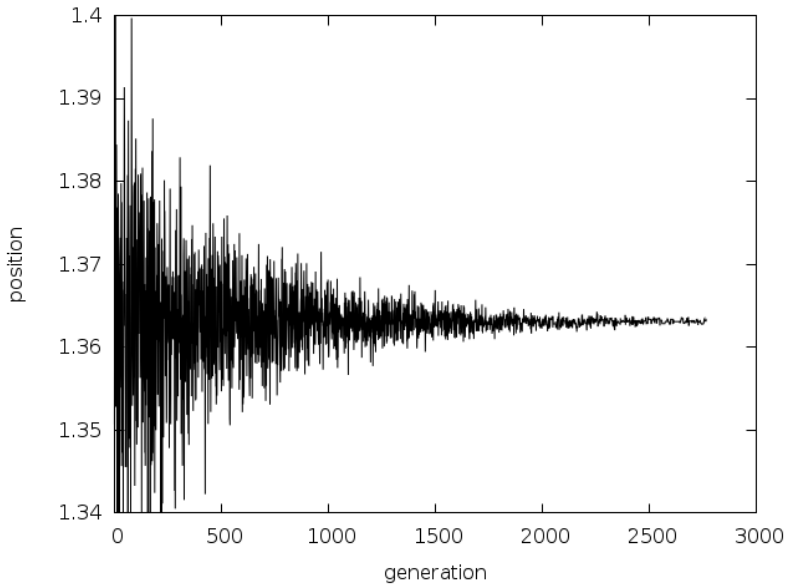
:학습률을 점진적으로 감소시킨다



시작할 때 학습률 크게 -> 빠르게 수렴하여 지역 최솟값에 빠지지 않도록

학습률 점점 작게 -> 전역 최솟값에 도달하도록

=> 담금질 기법(simulated annealing) 알고리즘과 유사



*담금질 기법(simulated annealing) 알고리즘

-전역 최적화 문제에 대한 일반적인 확률적 메타 알고리즘

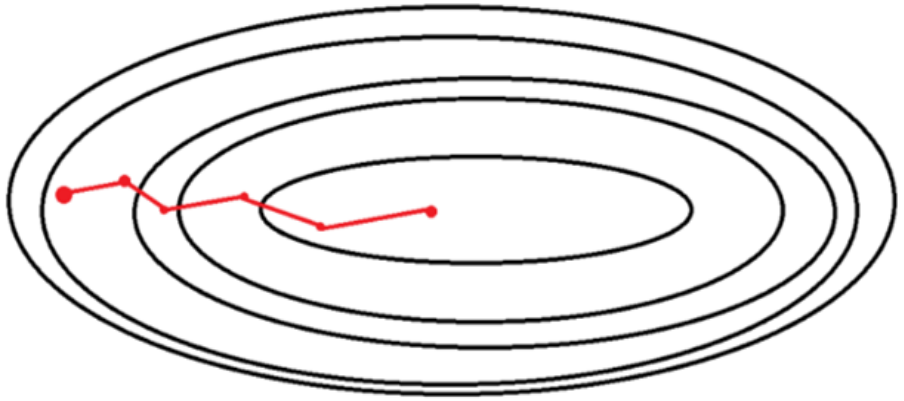
-광대한 탐색 공간 내 주어진 함수의 전역 최적해에 대한 적절한 근사 제공

-annealing: 금속재료를 가열한 다음 조금씩 냉각해 결정을 성장시켜 그 결함을 줄이는 작업

미니배치 경사 하강법(MBGD, Mini-Batch Gradient Descent)이란?

: 전체 경사 하강법과 확률적 경사 하강법 간의 절충안

* 미니배치 : 훈련 데이터로부터 학습을 수행하기 위해 추출된 일부



장점
- 행렬 연산에 최적화된 하드웨어, 특히 GPU를 사용해서 얻는 성능 향상
- 미니배치를 어느 정도 크게 하면 SGD보다 덜 불규칙적
- SGD보다 최솟값에 더 가까이 도달할 수 있음

단점
- SGD에 비해 지역 최솟값에서 빠져나오기 힘들 수도 있음

SGDClassifier

SGDClassifier(lossstr, penalty, alpha, l1_ratio, fit_intercept, max_iter, tol, shuffle, n_jobs, random_state, learning_rate, class_weight, average)

- lossstr : 손실함수 (default='hinge')
- penalty : {'l2', 'l1', 'elasticnet'}, default='l2'
- alpha : 값이 클수록 강력한 정규화(규제) 설정 (default=0.0001)
- l1_ratio : L1 규제의 비율(Elastic-Net 믹싱 파라미터 경우에만 사용) (default=0.15)
- fit_intercept : 모형에 상수항 (절편)이 있는가 없는가를 결정하는 인수 (default=True)
- max_iter : 계산에 사용할 작업 수 (default=1000)
- tol : 정밀도
- shuffle : 에포크 후에 트레이닝 데이터를 섞는 유무 (default=True)
- n_jobs : 병렬 처리 할 때 사용되는 CPU 코어 수
- random_state : 난수 seed 설정
- learning_rate : 학습속도 (default='optimal')
- class_weight : 클래스와 관련된 가중치 {class_label: weight} or “balanced”, default=None
- average : True로 설정하면 모든 업데이트에 대한 평균 SGD 가중치를 계산하고 결과를 coef_속성에 저장 (default=False)

fish_csv_data 로 SGDClassifier 실습하기

```
# fish 데이터 불러오기
import pandas as pd
fish = pd.read_csv('https://bit.ly/fish_csv_data')
```

```
fish_input = fish[['Weight','Length','Diagonal','Height','Width']].to_numpy()
fish_target = fish['Species'].to_numpy()
```

```
# 훈련 세트와 테스트 세트로 나누기
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)
```

fish_csv_data 로 SGDClassifier 실습하기

```
# 훈련 세트와 테스트 세트의 특성 표준화
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

```
#확률적 경사하강법 실행
from sklearn.linear_model import SGDClassifier
sc = SGDClassifier(loss='log', max_iter=10, random_state=42)
sc.fit(train_scaled, train_target)
```

```
# 훈련 세트와 테스트 세트 정확도 점수 출력
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```

```
0.773109243697479
0.775
```

```
# 추가 훈련 후 정확도 점수 출력
sc.partial_fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```

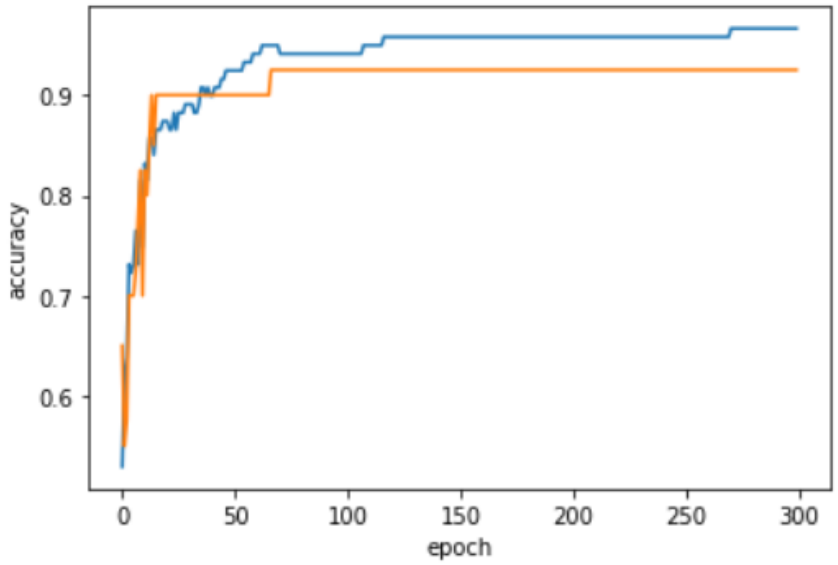
```
0.8151260504201681
0.85
```

fish_csv_data 로 SGDClassifier 실습하기

```
import numpy as np
sc = SGDClassifier(loss='log', random_state=42)
train_score = []
test_score = []
classes = np.unique(train_target)
```

```
# 훈련 진행(에포크 300번)
for _ in range(0, 300):
    sc.partial_fit(train_scaled, train_target, classes=classes)
    train_score.append(sc.score(train_scaled, train_target))
    test_score.append(sc.score(test_scaled, test_target))
```

```
# 훈련 세트와 테스트 세트 점수 그래프 그리기
import matplotlib.pyplot as plt
plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```



fish_csv_data 로 SGDClassifier 실습하기

```
# 에포크 100번으로 설정 후 다시 훈련하기
sc = SGDClassifier(loss='log', max_iter=100, tol=None, random_state=42)
sc.fit(train_scaled, train_target)
print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```

```
0.957983193277311
0.925
```

감사합니다