점진적인 학습

확률적 경사 하강법!

1

만약에…

- 훈련데이터가 한 번에 준비되는 것이 아니라 조금씩 전달된다면?
- 기존의 훈련 데이터에 새로운 데이터 추가해서 매일매일 훈련한다면?
  - 새로운 데이터를 활용해 모델을 훈련할 수 있음
  - 시간이 지날수록 데이터가 늘어나는 단점 → 데이터 용량 증가
  - 데이터 용량 증가로 인한 학습/검증용 서버 성능/용량 증설 필요
  - 지속적인 방법은 아님

2

만약에…

- 새로운 데이터를 추가할 때 이전 데이터를 버림
  - 훈련 데이터 크기를 일정하게 유지 → 데이터셋 크기 커지지 않은 장점
  - 버리는 데이터에서 중요한 데이터가 포함될 수 있음
  - 이렇게 생성된 모델은 버려진 데이터 클래스는 예측하지 못하는 문제 발생
- 그럼 앞서 훈련한 모델을 버리지 않고 새로운 데이터에 대해서만 조금씩 훈련 방법은 없을까?
  - 이렇게 되면 훈련에 사용한 데이터 모두 유지할 필요 없고
  - 앞서 학습한 클래스 데이터를 유실할 일이 없음

3

점진적 학습(온라인 학습)

- 학습이 끝나 제품화가 된 모델에 대해 미니 배치(Mini-batch)라 부르는 작은 묶음 단위의 데이터를 주입해 모델을 학습시키는 방법
- 미니 배치의 크기가 작기 때문에 학습 단계가 빠르고 비용이 적게 들어 모델은 데이터가 도착하는 대로 즉시 학습 할 수 있음
- 점진적으로 학습이 일어나기 때문에 이러한 명칭이 붙음
- 온라인 학습이라고 불리기도 하지만 오프라인으로도 시행됨(외부 메모리 학습)

내용 출처 : https://gooopy.tistory.com/123

4

점진적 학습(온라인 학습)

- 연속적으로 데이터를 받고 빠른 변화에 스스로 적응해야 하거나, 자원이 매우 한정된 환경에 적합함
- 새로운 데이터 샘플을 학습하면, 학습이 끝난 데이터는 더 이상 필요하지 않기 때문에 보관하지 않아도 됨으로 저장공간을 절약할 수 있음
- 대표 알고리즘
  - 확률적 경사 하강법, 미니배치 경사 하강법, 배치 경사 하강법

내용 출처 : https://gooopy.tistory.com/123

5

확률적 경사 하강법

- 경사를 따라 내려가는 방법
  - 확률 : '무작위' 또는 '랜덤'의 기술적인 표현
  - 경사 : 언덕의 기울기
  - 하강법 : 내려가는 방법

- 가장 가파른 경사를 따라 원하는 지점에 도달하는 것이 목표
  - 등산할 때 가장 빠른 길은 경사가 가장 가파른 길!
  - 하지만 한번에 걸음이 크면 경사를 따라 내려가지 못하고 올라갈 수 있음!
  - 가장 가파른 길을 찾아 내려오지만 조금씩 내려오는 것이 중요

6

확률적 경사 하강법

- 경사 하강법 모델 훈련 방법
  - 가장 가파른 길을 찾아 내려오지만 조금씩 내려 오는 것
- 확률적?
  - 훈련 세트를 사용해 모델을 훈련하기 때문 → 훈련 세트를 사용해 가장 가파른 길을 찾을 것
  - 그런데 전체 샘플을 사용하지 않고 딱 **하나의 샘플**을 훈련세트에서 랜덤하게 골라 가장 가파른 길 찾음
- **훈련세트에서 랜덤하게 하나의 샘플을 고르는 것!**

7

확률적 경사 하강법

- 만약 모든 샘플을 사용해 산을 못 내려오면?
  - 처음부터 다시 시작 → 훈련 세트에 모든 샘플 다시 채워넣음
  - 다시 랜덤하게 하나의 샘플을 선택해 경사를 내려감
  - 만족할만한 위치에 도달할 때까지 계속 내려감
- 에포크 Epoch
  - 훈련 세트를 한 번 모두 사용하는 과정
- 확률적 경사 하강법은 수십, 수백 번 이상의 에포크 수행함

8

미니배치 경사 하강법

- 경사를 아주 조금씩 내려가는 방법
- 무작위로 몇 개의 샘플을 선택해 경사를 따라 내려가는 방법
- 여러 개의 샘플을 사용해 경사 하강법을 수행하는 방식

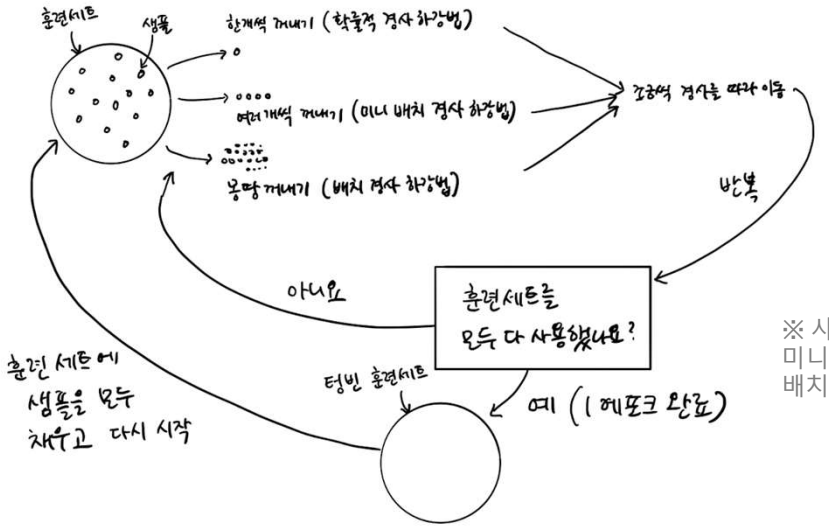
9

배치 경사 하강법

- 극단적으로 한 번 경사로를 따라 이동하기 위해 전체 샘플을 사용
- 전체 데이터를 사용하기 때문에 가장 안정적인 방법
- 전체 데이터를 사용하면 그만큼 컴퓨터 자원을 사용함
- 어떨 경우에는 데이터가 너무 많아서 한 번에 전체 데이터를 모두 못 읽을 수 있는 경우도 있음

10

경사 하강법 총정리!



※ 사이킷런에선
미니배치경사하강법,
배치하강법은 제공하지 않음

11

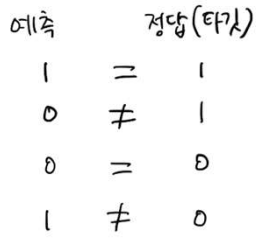
손실 함수

- 어떤 문제에서 머신러닝 알고리즘이 얼마나 엉터리인지 측정하는 기준
  - 손실 함수 값이 작을수록 좋음
  - 근데 어떤 값이 최솟값인지 알 순 없음
  - 가능한 많이 찾아보고 만족할만한 수준이면 산을 다 내려온 걸로 인정 해야함
  - 이런 면에서 조금씩 이동하는 확률적 경사 하강법이 적절함

12

손실 함수

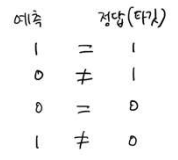
- 분류에서 손실 → 정답을 못 맞추는 것
- 예) 이진 분류 문제
  - 도미 : 양성 클래스(1)
  - 빙어 : 음성 클래스(0)
  - (우측 그림)결과 정확도 : 2 / 4 = 0.5



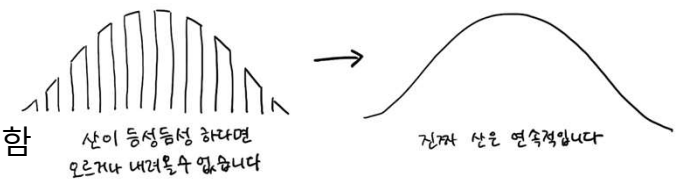
13

손실 함수

- (해당) 정확도를 손실 함수 적용
  - 음수를 취하면 -1.0이 가장 낮고, -0.0이 가장 높음
  - 정확도에 단점이 있음 4개의 샘플에서는 정확도가 다섯까지 뿐 (0, 0.25, 0.5, 0.75, 1)



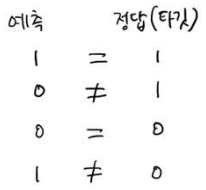
- 정확도가 듬성듬성 존재하면 경사 하강법을 이용해 조금씩 움직일 수 없음
  - 경사 하강법을 사용할 때 아주 조금씩 내려오기 때문
  - 산의 경사면은 연속적이여야 함



14

로지스틱 손실 함수

- 로지스틱 회귀 모델
  - 예측은 0 또는 1
  - 확률은 0~1 사이의 어떤 값도 될 수 있음 → 연속적



- 예시) 4개의 샘플 [0.9, 0.3, 0.2, 0.8]
  - **첫번째 샘플**의 예측 0.9, 양성 클래스 타깃 1과 곱한 다음 음수 변경
  - 이때 예측이 1에 가까울 수록 좋은 모델
  - 예측이 1에 가까울 수록 예측 타깃과 타깃의 곱의 음수는 점점 작아짐

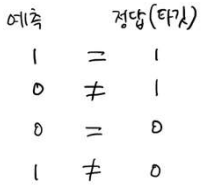
예측 정답(타깃)

0.9 × 1 → -0.9

15

로지스틱 손실 함수

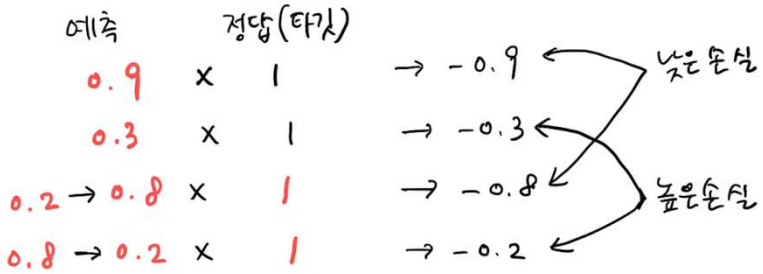
- 예시) 4개의 샘플 [0.9, 0.3, 0.2, 0.8] (계속)
  - **두번째 샘플**의 예측 0.3, 양성 클래스 타깃 1과 곱한 다음 음수 변경(첫번째 샘플보다 높은 손실)
  - **세번째 샘플**의 예측 0.2, 음성 클래스 타깃 0
  - 이 값을 예측 확률인 0.2와 그대로 곱하면 무조건 0이 됨
  - 타깃을 마치 양성 클래스처럼 바꾸어 1로 만듬
  - 대신 양성 클래스에 대한 예측으로 변경 : 1 – 0.2 = 0.8 사용
  - 곱하고 음수로 바꾸는 것은 동일 : -0.8
  - 음성 클래스 타깃을 맞추었으므로 손실이 낮아야 함
  - **네번째 샘플**도 세번째 샘플과 동일한 방법으로 적용 : -0.2



16

로지스틱 손실 함수

- 예시) 4개의 샘플 [0.9, 0.3, 0.2, 0.8] (계속)
  - 이렇게 하면 연속적인 손실함수 얻을 수 있음

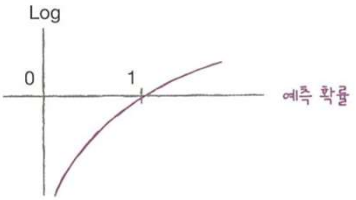


17

로지스틱 손실 함수

또는 '이진 크로스엔트로피 손실 함수'

- 예측확률을 로그 함수를 적용하면 더 좋음
  - 예측확률의 범위는 0~1 사이인데
    로그 함수는 이 사이에서 음수가 되어,
    최종 손실 값은 양수가 됨 → 손실 값이 양수가 되면 이해하기 쉬움
  - 로그 함수가 0에 가까울수록 큰 음수가 되기 때문에
    손실을 아주 크게 만들어 모델에 영향을 미칠 수 있음



타깃 = 1일 때
→ −log(예측 확률)

확률이 1에 멀어질수록
손실은 아주 큰 양수가 됨

타깃 = 0일 때
→ −log(1−예측 확률)

확률이 0에 멀어질수록
손실은 아주 큰 양수가 됨

18

로지스틱 손실 함수

- 사이킷런에서 제공 → SGDClassifier
- 이진 분류 : '손실 함수' 사용
- 다중 분류: '크로스엔트로피 손실함수' 사용

19

데이터 전처리

```
import pandas as pd

fish = pd.read_csv('https://raw.githubusercontent.com/Gonteer/2024DongALINC/main/001ML/00104_Regression_Logistic/fish.csv')
```

✓ 1.5s

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

✓ 0.0s

```
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)
```

✓ 1.8s

```
from sklearn.preprocessing import StandardScaler

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

✓ 0.0s

20

SGDClassifier

```
from sklearn.linear_model import SGDClassifier
```
0.1s

손실함수 지정
(로지스틱 손실함수)
수행 Epoch 지정

```
sc = SGDClassifier(loss='log_loss', max_iter=10, random_state=42)
sc.fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```
0.0s

```
0.773109243697479
0.775
```

```
c:\Python312\Lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:744:
ConvergenceWarning: Maximum number of iteration reached before convergence.
Consider increasing max_iter to improve the fit. warnings.warn(
```

이건 학습 Epoch 횟수가 충분하지 않아 생기는 에러임
→여기선 무시하면됨

21

SGDClassifier

- *partial_fit()* 메소드는 fit() 메소드와 호출방법 동일
- *partial_fit()* 메소드는 호출할 때 마다 1 epoch씩 이어 훈련 실시

```
sc.partial_fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
```
0.0s

```
0.8151260504201681
0.85
```

22

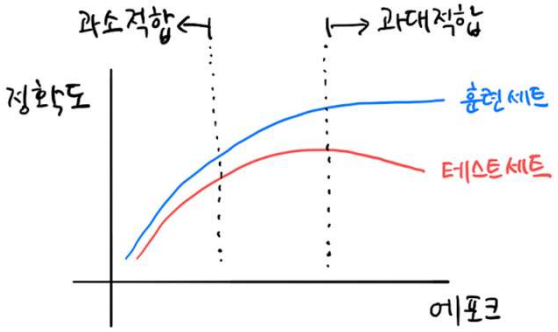
에포크와 과대/과소 적합

- 확률적 경사 하강법을 사용한 모델은 Epoch 횟수에 따라 과소적합 또는 과대적합 발생
- Epoch 횟수가 적으면 모델이 훈련 세트를 덜 학습
  - 산을 다 내려오지 못 하고 훈련을 마치는 경우
- 에포크 횟수가 충분히 많으면 훈련 세트를 완전히 학습 된 것
- **적은 Epoch 횟수**를 가진 훈련된 모델은 훈련 세트와 테스트 세트에 잘 맞지 않은 **과소적합**된 모델인 가능성 높음
- **많은 Epoch 횟수**를 가진 훈련된 모델은 훈련 세트에 너무 잘 맞아 테스트 세트에는 오히려 나쁜 **과대적합**된 모델이 될 가능성 높음

23

에포크와 과대/과소 적합

- 아래 그래프는 Epoch가 잘 진행됨에 따라 모델의 정확도를 나타냄
  - 훈련 세트 점수는 에포크가 진행될 수록 꾸준히 증가
  - 테스트 세트 점수는 어느 순간 감소 시작
  - 이 지점이 과대적합 시작 구간



- 과대적합이 시작하기 전 멈추는 것은 '**조기종료**'라고 함

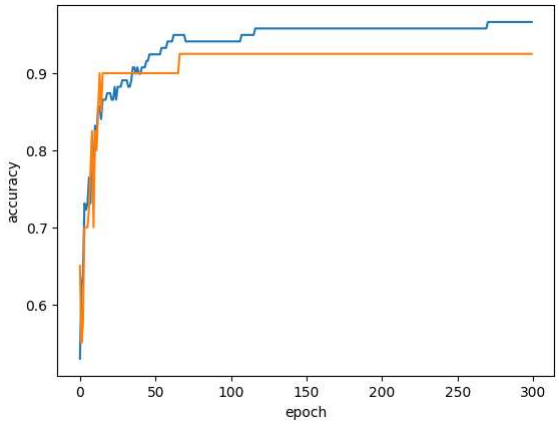
24

조기종료

```
import numpy as np

sc = SGDClassifier(loss='log_loss', random_state=42)

train_score = []
test_score = []

classes = np.unique(train_target)
```

```
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))
```

```
import matplotlib.pyplot as plt

plt.plot(train_score)
plt.plot(test_score)
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```



25

조기종료

- SDGClassifier는 일정 Epoch 동안 성능이 향상되지 않으면 더 훈련하지 않고 종료함
- tol 매개변수에는 향상될 최솟값을 지정할 수 있음

최소 Epoch 횟수

```
sc = SGDClassifier(loss='log_loss', 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))
```

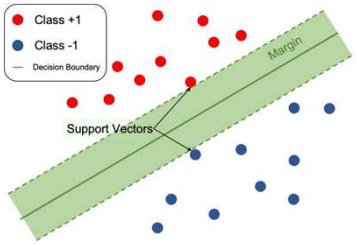
✓ 0.0s

```
0.957983193277311
0.925
```

26

힌지 손실 hinge loss

- SDGClassifier의 loss 매개변수 기본값은 'hinge'(힌지 손실)
- 힌지 손실은 '서포트 벡터 머신'이라 불림
  - '서포트 벡터 머신'은 또 다른 머신 러닝 알고리즘을 위한 손실 함수
  - 머신러닝 알고리즘에서 널리 사용되고 있음
- 이외에도 SDGClassifier의 loss 매개변수는 여러가지 있음



27

힌지 손실 hinge loss

```
sc = SGDClassifier(loss='hinge', 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))
```

✓ 0.0s

```
0.9495798319327731
0.925
```

28

감사합니다

내용 출처 정보 : https://www.hanbit.co.kr/store/books/look.php?p_code=B2002963743 https://gooopy.tistory.com/123

29