3️⃣

3주차

날짜	@2024년 10월 15일
상태	완료

▼ 지난 시간 요약

- 선형 회귀

  특성 다량 추가 시 과대 적합 → 규제
- pandas 데이터프레임
- numpy 배열 변환
- 사이킷런 - 변환기 메소드 제공 (훈련, 테스트 모두 사용)
- fit, transform 메소드
- 가중치 찾기 - L2 (Ridge), L1 (Lasso)

Ch 4. 다양한 분류 알고리즘

▼ **4-1) 로지스틱 회귀**

럭키 백 ← 생선

고객에게 확률 정보 제공

확률 계산 (물고기)

- 길이
- 높이
- 대각선
- 두께

- 무게

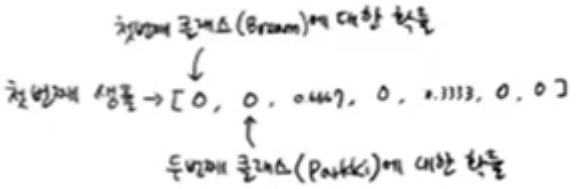
데이터 준비 (pandas)

```
import pandas as pd

fish = pd.read_csv('https://bit.ly/fish_csv_data')
fish.head()

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

k-최근접 이웃의 다중 분류

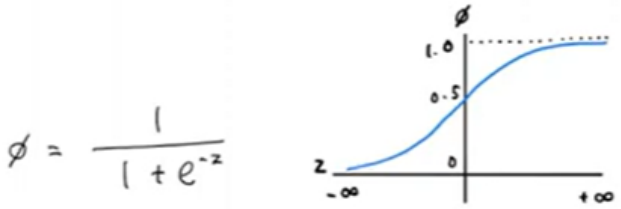
```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(train_scaled, train_target)  # 생선의 이름을 타깃값으로

print(kn.classes_)  # 속성 : 훈련 데이터로부터 추출한 값

print(kn.predict(test_scaled[:5]))  # 5개의 샘플 예측

proba = kn.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=4))
```

- 사이킷런 - 문자열로도 타깃값 레이블링 가능
- class_ : 속성 확인
- predict : 생선의 종류 예측
- predict_proba : 평균



로지스틱 회귀

$$z = a \times \text{무게} + b \times \text{길이} + c \times \text{대각선} + d \times \text{높이} + e \times \text{두께} + f$$



- 선형 함수를 학습
- 확률로 변환할 때 시그모이드 함수 (로지스틱) 사용
- tanh 함수도 시그모이드 함수로 불릴 때가 있음
- 0.5를 기준으로 양성/음성 클래스 판단 (분류 알고리즘)
- z값을 기준으로도 양성/음성 판단 가능 (확률 x)

로지스틱 함수_이진 분류

```
bream_smelt_indexes = (train_target == 'Bream') | (train_t
train_bream_smelt = train_scaled[bream_smelt_indexes]
target_bream_smelt = train_target[bream_smelt_indexes]

from sklearn.linear_model import LogisticRegression  # 로지

lr = LogisticRegression()  # 객체 생성
lr.fit(train_bream_smelt, target_bream_smelt)  # fit에 데이터

print(lr.predict(train_bream_smelt[:5]))  # 예측
```

```
print(lr.predict_proba(train_bream_smelt[:5]))  # 확률 - 첫
```

- z가 0.5 이상이면 양성 클래스(1)로 분류
- z가 0.5 미만이면 음성 클래스(0)로 분류

로지스틱 회귀 계수

```
print(lr.coef_, lr.intercept_)

decisions = lr.decision_function(train_bream_smelt[:5])  #
print(decisions)

from scipy.special import expit  # 시그모이드 함수
print(expit(decisions))
```

로지스틱 회귀 - 다중 분류

```
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))

proba = lr.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=3))

print(lr.coef_.shape, lr.intercept_.shape)
```

- max_iter : 반복 학습 횟수
- C : L2 노름 규제 기본 적용 (기본값 1 - 클수록 규제 풀어줌)

소프트맥스 함수

```
decision = lr.decision_function(test_scaled[:5])
print(np.round(decision, decimals=2))

from scipy.special import softmax  # softmax 함수

proba = softmax(decision, axis=1)
print(np.round(proba, decimals=3))
```

▼ 지난 시간 요약

- 럭키백 - 확률을 보고 고객 구매
- k 최근접 이웃 : 이웃한 샘플을 보고 확률 제공
- 로지스틱 회귀 (선형 방정식을 이용한 분류 알고리즘)
  - 양성 클래스/음성 클래스로 분류
- predict_proba : 시그모이드를 사용하지 않고 계산 효율성을 높임
- 다중 분류의 경우 각 클래스마다 확률 학습
- softmax 함수 : 다중 클래스 분류 모델을 만들 때 결과를 확률로 해석할 수 있도록 변환 - 높은 확률을 가지는 class로 분류

▼ 4-2) 확률적 경사 하강법

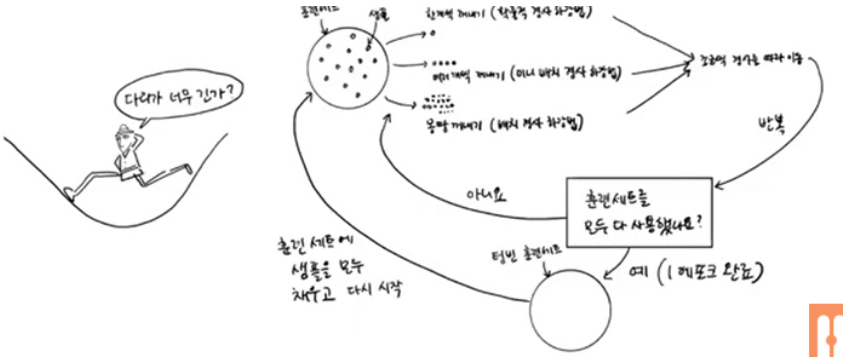
럭키 백 대박!

데이터가 추가될 때마다 새로운 모델 제작, 유지 보수

→ 점진적인 학습 방법을 사용하는 훈련 : 확률적 경사 하강법

확률적 경사 하강법(Stochastic gradient descent)

- ML, DL 최적화 o, 알고리즘 x

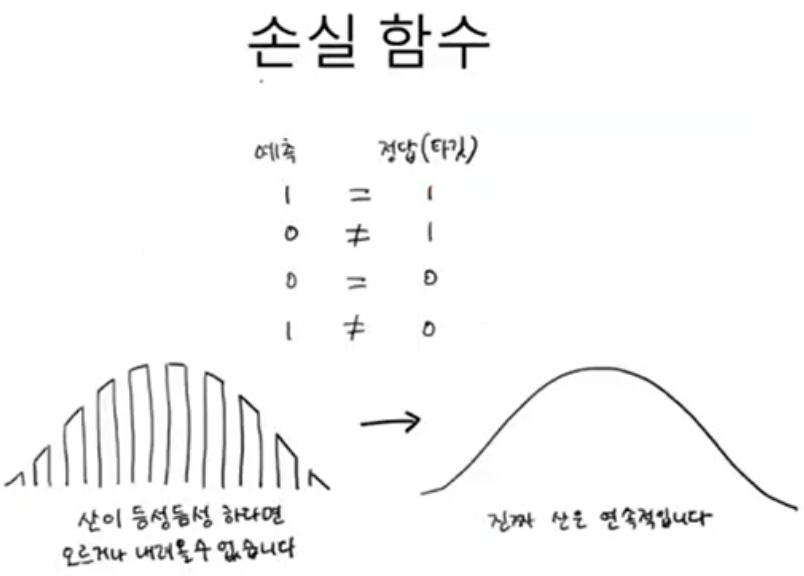


- 가파른 경사를 무작위로 조금씩 내려감 → 최적점 도달
- 과정
  1. 샘플 한 개씩 꺼내기
  2. 과정 반복
  3. 훈련 세트 모두 소진 → 1 에포크 완료
  4. 훈련 세트를 채워 두 번째 에포크 진행

  → 에포크를 여러 번 반복하는 방식
- 미니 배치 경사 하강법

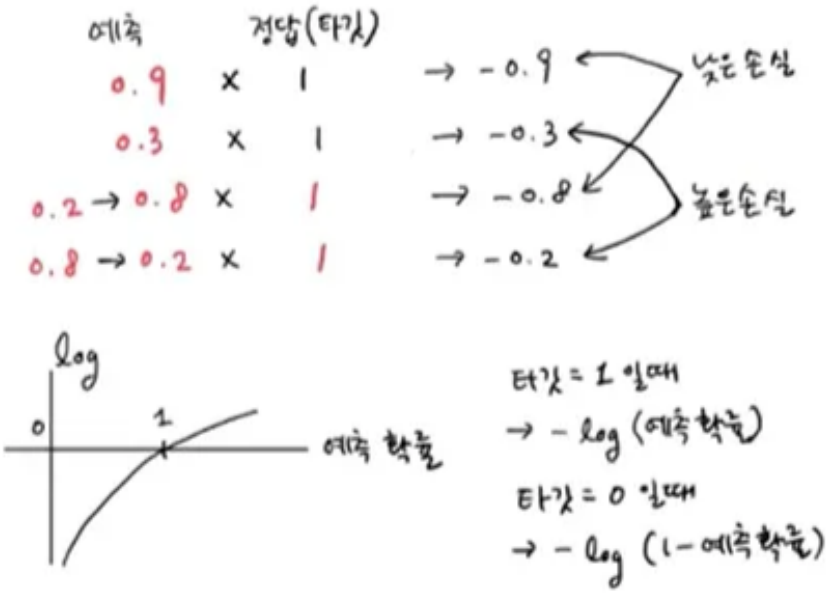
  미니 배치 개수 - 파라미터 (솔루션 지정)
- 배치 경사 하강법 : 한 번에 전체 훈련 세트를 꺼내 훈련

에포크 = 한 턴



손실 함수 : 머신 러닝 알고리즘의 나쁜 정도를 측정

로지스틱 손실 함수 (미분 가능)



- 양수로 바꿔 생각하면, 작은 게 손실이 적다.
- 타깃 값과 멀어질수록 손실이 기하급수적으로 커짐
- = 이진 크로스 엔트로피 손실 함수

데이터 전처리

```
import pandas as pd

fish = pd.read_csv('https://bit.ly/fish_csv_data')
fish_input = fish[['Weight','Length','Diagonal','Height','
fish_target = fish['Species'].to_numpy()

from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train
    fish_input, fish_target, random_state=42)

from sklearn.preprocessing import StandardScaler
```

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

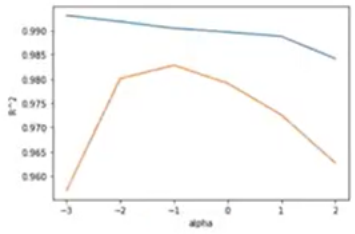
- 특성 - 스케일이 저마다 다름

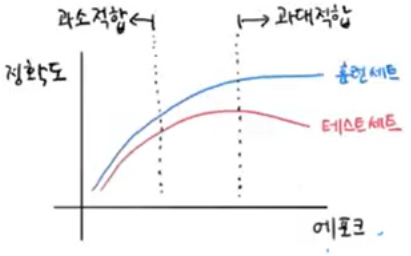
SGDClassfier

```
from sklearn.linear_model import SGDClassifier

sc = SGDClassifier(loss='log_loss', max_iter=10, random_st
sc.fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))

sc.partial_fit(train_scaled, train_target)

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

- 확률적 경사 하강법의 분류 알고리즘
- 로지스틱 손실 함수 지정 (회귀)
- loss 함수 : 최적화할 모델 정함
- max_iter : 점진적 학습 (epoke)
- partial.fit : 기존 학습 내용 유지하며 다시 훈련

에포크와 과대/과소적합





- 규제 값이 커짐 - 훈련, 테스트 모두 점수 감소
- 규제 값이 작아짐 - 훈련 과대적합, 테스트 점수 감소

→ 점수 차가 가장 적은 부분으로 모델 재훈련

- 테스트 세트의 점수가 떨어지기 전, 두 세트의 점수가 비슷할 때의 지점을 찾아 해당하는 에포크로 훈련시킴

→ 조기 종료

조기 종료

```
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=cla

    train_score.append(sc.score(train_scaled, train_target
    test_score.append(sc.score(test_scaled, test_target))

sc = SGDClassifier(loss='log', max_iter=100, tol=None, ran

sc.fit(train_scaled, train_target)
```

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

- 적절한 버퍼 갯수만큼 훈련 진행
- loss
    - hinge : svm(soft vector machine) 알고리즘