week3. 다양한 분류 알고리즘

scikit-learn

- LogisticRegression : 로지스틱 회귀를 위한 클래스.
- C 매개변수에서 규제의 강도를 제어, 기본 값은 1.0 이며 값이 작을수록 규제 강해짐.
- penalty 매개변수에서 L2 규제 (릿지 방식) 과 L1 규제 (리쏘 방식) 선택 가능 / 기본 값은 L2.

predict_proba()

- 예측 확률을 반환하는 메세드
- 이진 분류 : 샘플마다 음성 클래스와 양성 클래스에 대한 확률 반환
- 다중 분류 : 샘플마다 모든 클래스에 대한 확률 반환

decision_function()

- 모델이 학습한 선형 방정식의 출력을 반환
- 이진 분류 : 양성 클래스의 확률 반환, 값이 0보다 크면 양성 클래스, 작거나 같으면 음성 클래스로 예측
- 다중 분류 : 각 클래스마다 선형 방정식을 계산, 가장 큰 값의 클래스가 예측 클래스가 됨.

▼ 럭키백의 확률

unique 함수 : 어떤 종류 생선 있는지 확인

```
import pandas as pd
fish = pd.read_csv('https://bit.ly/fish_csv_data') # csv 파일 데이터 직접 읽음
fish.head()

# 어떤 종류의 생선이 있는지 unique 함수 사용하여 확인
print(pd.unique(fish['Species']))

# Species 열을 타깃으로 만들고 나머지 5개 열은 입력 데이터로 사용하기 - 원하는 열을 리스
fish_input = fish[['Weight', 'Length', 'Diagonal', 'Height', 'Width']].to_
print(fish_input[:5])
```

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

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

▼ k-최근접 이웃 분류기의 확률 예측

- 타깃 데이터에 2개 이상의 클래스가 포함된 문제를 '다중 분류' 라고 부름.
- 타깃값을 그대로 사이킷런 모델에 전달하면 순서가 자동으로 알파벳 순으로 매겨짐.
- pd.unique(fish['Species']) 로 출력했던 순서와는 상이함.

```
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))

print(kn.classes_)
-> ['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']

print(kn.predict(test_scaled[:5])) # 테스트 세트에 있는 처음 5개 샘플의 타깃값 예
-> ['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']
```

사이킷런 분류 모델 :

- predict_proba() 메서드로 클래스별 확률값을 반환.

```
import numpy as np
proba = kn.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=4)) # 소수점 네번째자리까지 표기, 다섯자리에서 반올
```

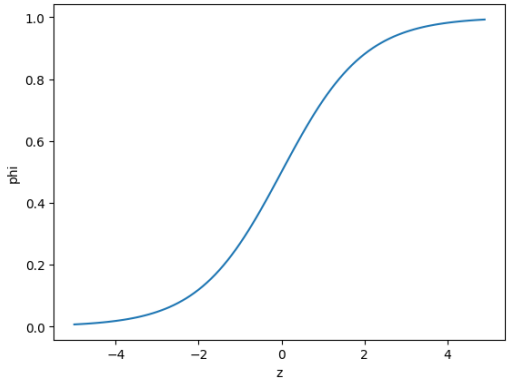
```
distances, indexes = kn.kneighbors(test_scaled[3:4]) # 슬라이싱 연산자는 하나의
print(train_target[indexes])
```

로지스틱 회귀

선형 방정식을 사용한 분류 알고리즘.

선형 회귀와 달리 시그모이드 함수나 소프트맥스 함수를 사용하여 클래스 확률을 출력함.

- 시그모이드 함수
- 확률이 아주 큰 음수일 때 0으로, 아주 큰 양수일 때 1이 되도록 하는 방법.



▼ 로지스틱 회귀로 이진 분류 (불리언 인덱싱)

시그모이드 함수의 출력이 0.5보다 크면 양성 클래스, 0.5보다 작으면 음성 클래스

```
# 로지스틱 회귀로 이진 분류 수행하기 : 불리언 인덱싱
char_arr = np.array(['A','B','C','D','E'])
print(char_arr[[True, False, True, False, False]])
-> ['A' 'C']

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]

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(train_bream_smelt, target_bream_smelt)
```

```
# 훈련 모델을 사용해 train_bream_smelt 에 있는 처음 5개 샘플 예측
print(lr.predict(train_bream_smelt[:5])) # 두번째 샘플 제외하고 모두 도미로 예측

# 예측 확률은 predict_proba() 메서드에서 제공
print(lr.predict_proba(train_bream_smelt[:5]))

# 사이킷런은 타깃값을 알파벳순으로 정렬하여 사용
print(lr.classes_)

print(lr.coef_, lr.intercept_)
-> [[-0.40451732 -0.57582787 -0.66248158 -1.01329614 -0.73123131]] [-2.161

# LogisticRegression 모델로 z 값을 계산할 수 있을까?
# LogisticRegression 클래스는 decision_function() 메서드로 z 값 출력 가능
decisions = lr.decision_function(train_bream_smelt[:5])
print(decisions)
-> [-6.02991358  3.57043428 -5.26630496 -4.24382314 -6.06135688]

# z 값을 시그모이드 함수에 통과, 확률 얻을 수 있음. expit() 사용.
from scipy.special import expit
print(expit(decisions))
-> [0.00239993 0.97262675 0.00513614 0.01414953 0.00232581]
```

▼ 로지스틱 회귀로 다중 분류 (소프트맥스 함수)

```
LogisticRegression 클래스 : 반복적인 알고리즘 사용, 기본값은 100.
릿지 회귀와 같이 계수의 제곱을 규제.(L2 규제)
릿지 회귀 : alpha 매개변수로 규제 양 조절
C는 alpha 와 반대로 작을수록 규제가 커짐.
```

```
lr = LogisticRegression(C=20, max_iter=1000)
lr.fit(train_scaled, train_target)

print(lr.coef_.shape, lr.intercept_.shape)
# 다중 분류는 클래스마다 z 값을 하나씩 계산.
다중분류는 '소프트맥스(지수함수)함수' 사용.

from scipy.special import softmax

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

-> [[0.    0.014 0.842 0.    0.135 0.007 0.003]
 [0.    0.003 0.044 0.    0.007 0.946 0.   ]
```

```
[0.    0.    0.034 0.934 0.015 0.016 0.   ]
[0.011 0.034 0.305 0.006 0.567 0.    0.076]
[0.    0.    0.904 0.002 0.089 0.002 0.001]]
```

scikit-learn

- SGDClassifier : 확률적 경사하강법을 위한 분류 모델을 만든다.
- loss 매개 변수 : 확률적 경사 하강법으로 최적화할 손실함수 지정
- 기본 값 : 서포트 벡터 머신을 위한 'hinge' 손실 함수
- 로지스틱 회귀를 위해서는 'log'로 지정

- penalty 매개변수 : 규제 종류 지정, 기본 값은 'L2' 규제를 위한 'l2'
- 규제 강도 : alpha 매개 변수에서 지정, 기본 값은 0.0001

- max_iter 매개변수 : 에포크 횟수 지정, 기본값은 1000

- tol 매개변수 : 반복 멈출 조건, 기본 값은 0.0001
- n_iter_no_change 매개변수에서 지정한 에포크 동안, 손실이 tol 만큼 줄지 않으면 알고리즘이 중단됨.

- SGDRegressor : 확률적 경사하강법을 사용한 회귀 모델을 만든다.
- loss 매개변수에서 손실 함수 지정, 기본 값은 제곱 오차를 나타내는 'squared_loss'

확률적 경사 하강법

훈련 세트에서 샘플을 하나씩 꺼내 손실함수의 경사를 따라 최적의 모델을 찾는 알고리즘.
샘플을 하나씩 사용하지 않고 여러개 사용하면 미니배치 경사 하강법
한 번에 전체 샘플을 사용하면 배치 경사하강법

손실 함수

:머신러닝이 얼마나 엉터리인지 측정하는 기준. 즉, 손실함수 값이 작을수록 좋음

확률적 경사하강법이 최적화할 대상.

1. 이진 분류 : 로지스틱 회귀(또는 이진 크로스엔트로피) 손실 함수 사용
2. 다중 분류 : 크로스엔트로피 손실 함수 사용
3. 회귀 문제 : 평균 제곱 오차 손실 함수 사용

에포크 : 확률적 경사 하강법에서 전체 샘플을 모두 사용하는 한 번 반복을 의미, 일반적으로 경사 하강법 알고리즘은 수십~수백 번 에포크를 반복

로지스틱 손실 함수

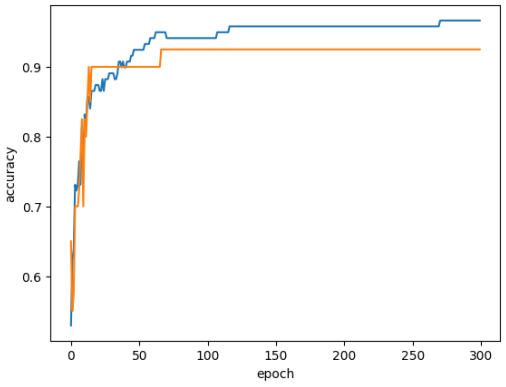
1. 양성 클래스 (타깃=1) 일 때 손실은 -log(예측 확률) 로 계산 : 확률은 1에서 멀어져 0에 가까워질수록 손실은 아주 큰 양수가 됨.
2. 음성 클래스 (타깃=0) 일 때 손실은 -log(1-예측 확률)로 계산 : 확률이 0에서 멀어져 1에 가까워질수록 손실은 아주 큰 음수가 됨

▼ SGDClassifier

```
# 사이킷런 train_test_split() 함수 사용해 훈련세트, 테스트 세트 나눔
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)

# 훈련 세트에서 학습한 통계 값으로 테스트 세트도 변환
from sklearn.preprocessing import StandardScaler

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

# 사이킷런에서 확률적 경사하강법 제공하는 분류용 클래스 : SGDClassifier
from sklearn.linear_model import SGDClassifier

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


# 모델을 이어서 훈련할 때에는 partial_fit() 메서드 사용 :
fit() 메서드와 사용법 같지만, 호출할 때마다 1 에포크씩 이어서 훈련 가능
sc.partial_fit(train_scaled, train_target)
```

▼ 에포크 , 과대/과소 적합

```
# 과대 적합이 시작하기 전에 훈련을 멈추는 것 : 조기 종료
# fit() 메서드 대신 partial_fit() 메서드 사용

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

# for 문 사용하여 300번 에포크 동안 훈련 반복
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()
```



100 번째 이후에는 점수가 벌어지는 경향. 즉, 최적의 수는 100이다.

```
sc = SGDClassifier(loss='log_loss', max_iter=100, tol=None, random_state=4
sc.fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
print(sc.score(test_scaled, test_target))
-> 0.957983193277311
   0.925
```