Chapter 14. scikit-learn 로지스틱회귀_다중분류

- Logistic Regression (이진분류)
- sklearn preprocessing 정규화 전처리
- multinomial LogisticRegression(다중분류)

sklearn(사이킷런 : scikit-learn)
- Logistic Regression : 로지스틱 회귀 (분류)

- 회귀를 활용 데이터의 범주가 속할 확률을 0 ~ 1 사이의 값으로 예측하고 그 확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해주는 지도 학습 알고리즘(예, 합격/불합격 , 스팸메일/정상메일, 금리 up/down 등)

- 선형 회귀와 달리 종속 변수가 범주형 데이터 일때 사용, 결과가 특정 카테고리로 분류되기 때문에 분류 모델

- 종속 변수(출력결과)에 따라 이진 분류, 다중 분류로 나뉨

- 이진 분류 (시그모이드 함수), 다중 분류 (소프트맥스 함수)
  - 이진 분류일 경우 시그모이드 함수의 출력이 0.5 이상 이면 양성 클래스(1)
  - 이진 분류일 경우 시그모이드 함수의 출력이 0.5 이하 이면 음성 클래스(0)

• 시그모이드(sigmoid) 함수

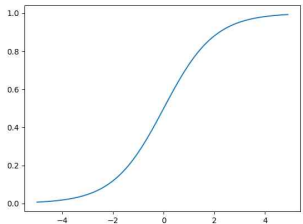
특정 값을 0 ~ 1 사이 값으로 출력

공식 : $y = \frac{1}{1 + e^{-x}}$

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-5,5,0.1)

y = 1/(1+np.exp(-x))

print(y)
plt.plot(x, y)
plt.show()
```



로지스틱 회귀_시그모이드(sigmoid) 함수
(이진 클래스 분류를 위해 예측값을 도출하는 가설함수)

입력데이터(X)

선형방정식 훈련
(ax + b)
a : 가중치 , b : 절편

출력 z = ax + b

z 에 대한 결과
출력 함수
(Sigmoid 함수 적용)

0 ~ 1
사이 값
생성

해당값은

양성(1) 클래스
에 대한 확률

≠

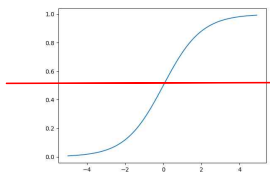
음성(0) 클래스 확률 :
1 - 양성(1) 클래스 확률

$$y(\text{출력값}) = \frac{1}{1 + e^{-(ax+b)}}$$

(동일)

z = ax + b

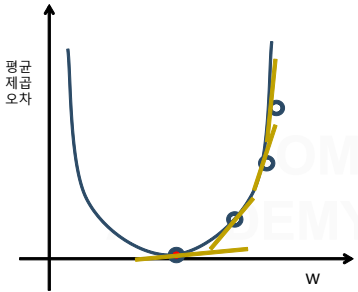
$$y = \text{sigmoid}(z) = \frac{1}{1 + e^{-(z)}}$$



시그모이드 계산값이 0.5 이상
이면 1 이 나올 확률이 높다고
판단하여 1(양성)로 예측

시그모이드 계산값이 0.5 미만
이면 0 이 나올 확률이 높다고
판단하여 0(음성)로 예측

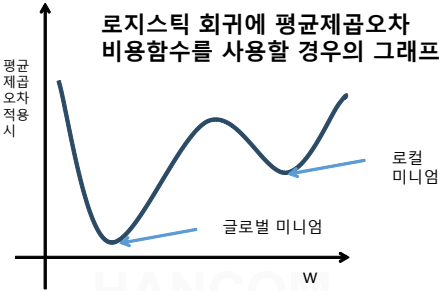
- 시그모이드 계산값(sigmoid(z))은 결과가 나타날 확률을 의미



선형 회귀의 경사하강법에 따른 평균제곱오차

평균제곱오차 비용함수는 아래쪽으로 볼록한 함수,
경사하강법을 사용해 최저의 에러를 갖는 회귀계수를 찾을 수 있음!!

로지스틱 회귀에 평균제곱오차 비용함수를 사용할 경우의 그래프



로지스틱 회귀 = 선형회귀 + 시그모이드
시그모이드 영향으로 평균제곱오차 그래프는 한 개 이상의 로컬 미니엄을 가질 수 있음
따라서, 평균제곱오차는 로지스틱회귀에 적합한 비용함수가 아님!!

크로스 엔트로피 비용함수 사용

크로스 엔트로피 (cross entropy 비용(손실)함수)

: 서로 다른 두 확률 분포의 차이를 의미
: 로지스틱 회귀 관점에서 **모델 예측값의 확률과 실제값 확률의 차이**
: **예측값과 실제값의 차이를 가장 최소화(손실 최소화)하는 w를 구하는 방법**

* cross entropy 비용 함수 종류

- 이진 분류 - **binary_crossentropy 비용 함수**

- 다중 분류 - **categorical_crossentropy 비용 함수 (타깃 : one-hot 인코딩 상태)**

- 다중 분류 - **sparse_categorical_crossentropy 비용 함수 (타깃 : 일반적인 정수 형태(1,2,3..))**

2진 분류를 위해 예측값 도출하는가설함수를 **sigmoid** 함수 사용시의 손실함수

로지스틱 회귀 손실 함수 (binary cross-entropy)

공식 : $-\frac{1}{n}\sum \underbrace{ylogh}_{\text{(y=1 경우)}} + \underbrace{(1-y)log(1-h)}_{\text{(y=0 경우)}}$

y : 실제 타깃값 (0 또는 1)
h : 출력 예측 값

로그 손실함수는 최소값을 찾는 정규방정식 이 없기 때문에 **경사하강법으로 최적화 진행**

오차 역전파 편미분 계산 , y: 실제 타깃 값 , P : 출력 예측값 , x : 입력데이터
E(y, P) = - (ylog(p) + (1-y)log(1-p))

if y = 1 --> cost(P(x),y) = -log(P(x))
if y = 0 --> cost(P(x),y) = -log(1-P(x))

결론, 가중치 a 에 대한 편미분 : $\frac{\partial E}{\partial a}$ = (P - y)x

절편 b 에 대한 편미분 : $\frac{\partial E}{\partial b}$ = (P - y)

P 출력 예측값은 선형회귀와 달리 논리적으로 0, 1 값을 갖기위해 sigmod 활성 함수 적용
z = ax + b ==> sigmod(z) == > P : sigmod(ax + b)

가중치(a) , 절편(b) 업데이트
a_up = (P - y) x = (sigmod(ax + b) - y) x
b_up = (p - y) = (sigmod(ax + b) - y)

a(가중치) = a - lr * a_up
b(절편) = b - lr * a_up

오차가 가장 적은 최적의 a(가중치), b(절편) 을 찾음

* lr : 학습률

https://asthtls.tistory.com/996
미분 공식 참조

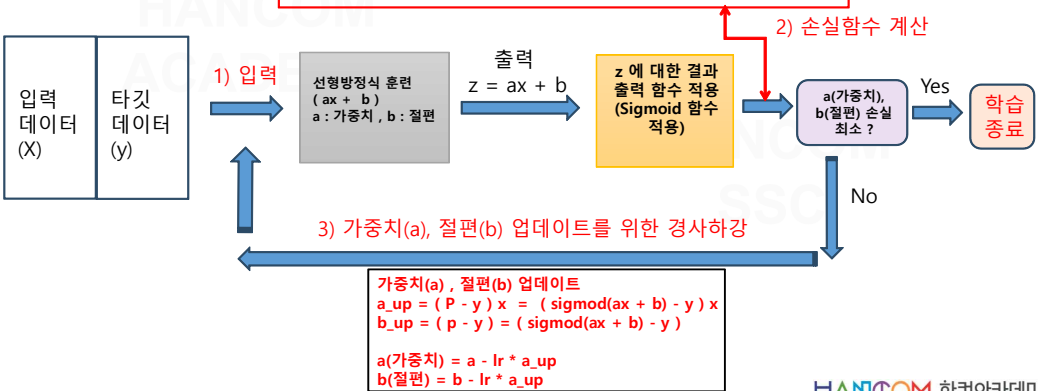
PPT12장 심층심경망
: 구현코드 참조

로지스틱 회귀 전체 훈련모델 (a, b 업데이트 과정)

이진 크로스엔트로피 손실 함수

$$-\frac{1}{n}\sum ylogh + (1-y)log(1-h)$$

y : 실제 타깃값 (0 또는 1)
h : 출력 예측 값



다중분류를 위한 categorical 크로스 엔트로피 비용함수

공식 : $-\sum_{x} p(x)\log q(x)$ p(x) : 실제 데이터의 분포 , q(x) : 모델의 예측값의 분포

실 데이터 ohe-hot = [0, 1, 0]
softmax 예측 출력값 = [0.2, 0.7, 0.1]

손실 계산

$$-\sum P(x)logq(x)$$
$$= -(0 * log0.2 + 1 * log0.7 + 0 * log0.1)$$
$$= -log0.7 \approx \mathbf{0.357\ (loss)}$$

다중분류를 위해 예측값 도출하는 출력단의 활성화함수를 소프트맥스 가설함수 사용 시 ==> (categorical cross entropy 손실함수 사용)

$$\mathbf{cost(w)} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_j^{(i)}\, log(p_j^{(i)})$$

y_j : 실제값 원-핫벡터, P_j : 예측값 확률
k : 클래스 수 , n : 데이터셋의 수

다중 분류 : (다중분류, categorical cross entropy 손실함수)

cost(w) = $-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_j^{(i)} \log(p_j^{(i)})$

loss 계산 예)

(실제값) y_j = [0 1 0] , (예측값) p_j = [0 1 0] 일 경우

$$-\begin{bmatrix}0\\1\\0\end{bmatrix} \times \begin{bmatrix}log0\\log1\\log0\end{bmatrix} = \begin{bmatrix}0\\1\\0\end{bmatrix} \times \begin{bmatrix}-\infty\\0\\-\infty\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix} = 0$$

예측값과 실제값이 같아 손실이 0

(실제값) y_j = [1 0 0] , (예측값) p_j = [0.7 0.2 0.1] 일 경우

$$-\begin{bmatrix}1\\0\\0\end{bmatrix} \times \begin{bmatrix}log(0.7)\\log(0.2)\\log(0.1)\end{bmatrix} = -log(0.7) = 0.35$$

예측값과 실제값 차이로 손실이 0.35

예측값과 결과값이 비슷할수록 cost(loss)는 줄어듬, 즉 실제값이 1인 위치의 예측값을 가능한 1로 가깝게 만들어주어 손실을 0으로 만들가는게 목표!
cost의 값을 줄여 나가는 방향으로 경사하강법 적용하여 최적화

예제 코드

```
import numpy as np
import pandas as pd

pd.set_option('display.max_rows',20)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width',1000)


titanic_df = pd.read_csv('titanic_passengers.csv')
print(titanic_df.shape)
print(titanic_df.head())

# 사이킷런 타깃값 문자열 데이터 사용 가능 체크
# Survived 컬럼 데이터를 target 데이터로 활용
# 1 : 생존 survival, 0 : 별세 fail
titanic_df['Survived'] = titanic_df['Survived'].map({1:'survival',0:'fail'})  # 타깃 데이터 문자열 변경

# 분석에 사용할 특징 데이터 셋 선택
# Sex, Age, Pclass 컬럼 데이터셋이 생존에 영향을 주는걸로 설정  # 입력 데이터

# Sex 컬럼 'female', 'male' 문자열 데이터를  여성 : 1, 남성 : 0 으로 변경
titanic_df['Sex'] = titanic_df['Sex'].map({'female':1, 'male':0})
```

```
# Pclass : 티겟의 클래스(1등석,2등석..)
# SibSp : 함께 탑승한 형제와 배우자의 수
# Parch : 함께 탑승한 부모, 아이의 수
# Fare : 탑승료
# Cabin : 객실 번호
# Embarked : 탑승 항구(C=Cherbourg, Q=Queenstown..)
```

예제 코드


```
# 결측치 검사
print(titanic_df.info())
print(titanic_df.loc[titanic_df['Age'].isnull(),['Age']])  # 177개 NAN

# 결측치 채우기 : 평균 값
titanic_df['Age'].fillna(value=titanic_df['Age'].mean(), inplace=True)
print(titanic_df.head(10))

# PClass ==> 1등석, 2등석, 3등석 구분  : 판다스 원핫인코딩
onehot_Pclass = pd.get_dummies(titanic_df['Pclass'], prefix='Class')
print(onehot_Pclass)  # Class_1  Class_2  Class_3 컬럼 생성

# titanic_df 와 onehot_Pclass 와 결합
titanic_df = pd.concat([titanic_df,onehot_Pclass], axis=1)
print(titanic_df.head(10))

# 데이터셋 준비
# 입력 데이터 셋 : [Sex, Age, Class_1 ,  Class_2 ] 컬럼
# 타깃 데이터 셋 : [Survived] 컬럼
titanic_Info = titanic_df[['Sex','Age','Class_1','Class_2']]
titanic_survival = titanic_df['Survived']
print(titanic_Info.head(5))
print(titanic_survival.head(5))
```

입력데이터

	Sex	Age	Class_1	Class_2
0	0	22.0	0	0
1	1	38.0	1	0
2	1	26.0	0	0
3	1	35.0	1	0
4	0	35.0	0	0

타깃데이터

0	fail
1	survival
2	survival
3	survival
4	fail

예제 코드

```
# train dataset / test dastaset : 훈련,테스트 데이터셋 분리
from sklearn.model_selection import train_test_split
train_input,test_input,train_target,test_target = ₩
    train_test_split(titanic_Info, titanic_survival, random_state=42)

# 특성 데이터 스케일 변환
# StandardScaler : 모든 값이 평균 0, 표준편차가 1인 정규분포로 변환
# MinMaxScaler : 최소값 0, 최대값 1로 변환
# RobustScaler : 중앙값 과 IQR(interquartile range): 25%~75% 사이의 범위 사용해 변환

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
train_scaled = scaler.fit_transform(train_input)
# fit() 했음으로 transform만 하면됨
test_scaled = scaler.transform(test_input)  # 동일 스케일로 test 데이터도 변환
print(train_scaled) # numpy.ndarray 변환 출력
print(train_scaled[:,0].std()) # 0번째 열(Sex 특성) 데이터 표준편차 1
```

```
[[-0.72224656  0.0171447   1.84248116 -0.51725447]
 [-0.72224656 -0.34431976 -0.54274639 -0.51725447]
 [ 1.38456873 -0.42124153 -0.54274639  1.93328442]
 ...
 [-0.72224656  0.88642859 -0.54274639 -0.51725447]
 [ 1.38456873 -1.19045925  1.84248116 -0.51725447]
 [-0.72224656 -0.65200684  1.84248116 -0.51725447]]
```

```
1.0
```

예제 코드

```
# 모델 생성 및 평가
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()  # 모델 생성
lr_model.fit(train_scaled, train_target)  # 훈련

# 훈련 셋 평가
print(lr_model.score(train_scaled, train_target)) # 0.7949101796407185
# 테스트 셋 평가
print(lr_model.score(test_scaled, test_target)) # 0.8026905829596412

# 모델 훈련 계수(가중치, 절편)
print(lr_model.coef_, lr_model.intercept_)
```

sklearn LogisticRegression : 경사하강법 이용해 최적화 진행
max_iter(기본 100) :경사하강 반복회수

```
[[ 1.21279289 -0.36626498  0.87326543  0.5229584 ]] [-0.66707021]
```

회귀모델 학습 방정식
: z = 1.21279289 * (Sex) - 0.36626498 * (Age)
+ 0.87326543 * (Class_1) + 0.5229584 * (Class_2) - 0.66707021
의미 : 여성(1) 과 Class_1(일등석) 여부가 생존 확률에 중요
: 나이에 대한 계수는 음수임으로 나이가 많을 수록 생존 확률 저하

예제 코드

```
# 예측
print(lr_model.predict(train_scaled[:5]))

# 예측 확률
print(lr_model.predict_proba(train_scaled[:5]))

print(lr_model.classes_)
```

['fail' 'fail' 'survival' 'fail' 'survival']

예측확률

```
[ [0.55250885 0.44749115]
  [0.89672535 0.10327465]
  [0.15401175 0.84598825]
  [0.88863133 0.11136867]
  [0.354343   0.645657  ] ]
```

(음성 클래스 확률)

(양성 클래스 확률)

(예측으로 사용한 클래스 정보 확인
음성, 양성 클래스 구분
: 사이킷런 알파벳순으로 정렬)

['fail' 'survival']

음성 클래스

양성 클래스

예제 코드

```
# z 값 계산
decisions = lr_model.decision_function(train_scaled[:5])
print(decisions)
```

양성 클래스에 대한
선형 방정식 z값 계산

```
[-0.21081267 -2.16135767  1.7034766  -2.07683641  0.60000303]
```

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

```
[0.44749115 0.10327465 0.84598825 0.11136867 0.645657  ]
```

predict_proba() 메서드로 확인한 예측 확률에서
양성 클래스에 대한 예측 확률과 동일 결과

임의의 데이터로 생존 여부 예측

```
sampledf = pd.DataFrame({'Sex':[1,1,0],'Age':[15, 30, 60],'Class_1':[1,0,1],
                         'Class_2':[0,1,0]}, index=['Kim','Hong','Park'])
print(sampledf)

# 스케일 변환
sample_scaled = scaler.transform(sampledf)
print(sample_scaled)

# 예측
print(lr_model.predict(sample_scaled))

# 예측 확률 확인
print(lr_model.predict_proba(sample_scaled))
```

	Sex	Age	Class_1	Class_2
Kim	1	15	1	0
Hong	1	30	0	1
Park	0	60	1	0

['survival' 'survival' 'fail']

```
[[0.05961135 0.94038865]
 [0.17734585 0.82265415]
 [0.74354998 0.25645002]]
```

Kim 생존

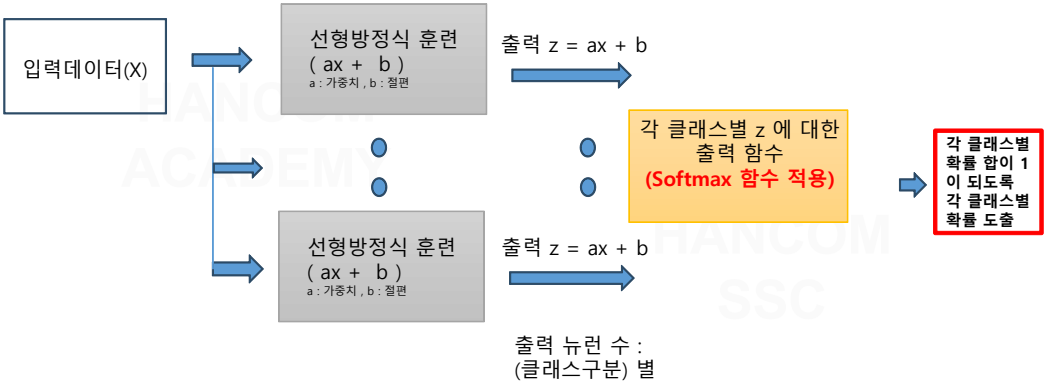
Hong 생존

Park 미생존

훈련 데이터에서 여성 : 1, 남성 : 0 으로 변경하여 모델 학습 함

* 여성 이면서 나이가 젊을 수록 생존 확률이 높음

소프트맥스(softmax) 함수 (다중 분류 가설함수)



• 소프트맥스(Softmax) 함수

: 여러 선형 방정식 출력값을 0 ~ 1로 압축하고 전체 합이 1이 되도록 만듬
: 지수 함수 활용하기 때문에 정규화된 지수 함수

예) 각 클래스별 z 출력 값 : z1 ~ z7

e_num = $e^{z1} + e^{z2} + e^{z3} + e^{z4} + e^{z5} + e^{z6} + e^{z7}$ ⇐ [지수 함수 적용 np.exp(x)]

확률 s1 ~ s7 은 $e^{z1} \sim e^{z7}$ **를 e_num 으로 각각 나누어 주면 됨**

s1 = $\frac{e^{z1}}{e\_num}$ **, s2 =** $\frac{e^{z2}}{e\_num}$ **. . . s7 =** $\frac{e^{z7}}{e\_num}$

```
import numpy as np

# softmax 함수 동작
# 여러개의 선행방정식 출력값을 0 ~ 1 로 압축하고 전체 합이 1이 되도록 만듬
# 이를 위해 지수 함수를 사용하기 때문에 정규화된 지수 함수

mylist = [-6.5, 1.03, 5.16, -2.73, 3.34, 0.33, -0.63]  # 예를 들어) 각 클래스별  Z 값
arr = np.array(mylist)

exp_a = np.exp(arr)  # 각 클래스별 Z값 지수함수 적용
sum_exp_a = np.sum(exp_a)  # 각 클래스별 지수함수 적용 합 계산

y = exp_a / sum_exp_a  # 각 클래스별 지수함수 / 합  ==> 확률 계산
print(y)
print(np.round(y, decimals=3))  # 소수점 네 번째 자리에서 반올림
```

```
[7.25685867e-06 1.35202933e-02 8.40663757e-01 3.14803000e-04
 1.36209176e-01 6.71397897e-03 2.57073478e-03]
```

```
[0.    0.014 0.841 0.    0.136 0.007 0.003]
```

클래스 중 확률이 가장 큰 값으로 예측

예제 코드

```
import numpy as np
import pandas as pd

# 과학적 표기법 대신 소수점 8자리까지 나타낸다.
np.set_printoptions(precision=8, suppress=True)

fish_data = pd.read_csv('fish_data.csv')  # 캐글 fish market  dataset

fish_input = fish_data[['Weight','Length','Diagonal','Height','Width']].to_numpy()
print(fish_input[:5])
fish_target = fish_data['Species'].to_numpy()
print(fish_data['Species'].unique())

# 훈련데이터 / 테스트데이터 셋 분리
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)

print(train_input.shape)
print(test_input.shape)
```

입력 특성 데이터 넘파이 배열 변환

타깃 데이터 넘파이 배열 변환

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

(119, 5)
(40, 5)

예제 코드

```
# StandardScaler 클래스 활용 표준화 전처리
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
Scaler.fit(train_input)
train_scaled = Scaler.transform(train_input)
test_scaled = Scaler.transform(test_input)


# 훈련 모델 생성
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(multi_class='multinomial' , C=20,
max_iter=1000)

# 훈련
lr_model.fit(train_scaled,train_target)

# 평가
print(lr_model.score(train_scaled,train_target))
print(lr_model.score(test_scaled,test_target))

# 예측
print(lr_model.predict(test_scaled[:5]))
```

다중분류를 위한 옵션

규제 제어 변수 : C
C의 기본값 1, C 가 작을수록 규제 커짐
C = 20 으로 규제 완화

max_iter : 훈련 반복 회수 지정
반복 회수 부족하면 경고 발생

0.9327731092436975
0.925

['Perch' 'Smelt' 'Pike' 'Roach' 'Perch']

(5개 샘플 데이터에 대해 각각의 확률로 샘플 데이터 예측)

예제 코드

```
# 예측 확률
print(lr_model.predict_proba(test_scaled[:5]))
```

Perch

Smelt

```
[[0.00000725 0.01351229 0.84127186 0.00031433 0.13567105 0.00667127 0.00255194]
 [0.00000001 0.00255611 0.04390867 0.0000338  0.00731105 0.94618511 0.00000526]
 [0.00001866 0.0000028  0.03406    0.93480449 0.01504749 0.0160369  0.00002966]
 [0.01093268 0.03404993 0.30554461 0.00660919 0.56657573 0.00006873 0.07621913]
 [0.00000449 0.00036729 0.90400204 0.00241275 0.08947489 0.00240967 0.00132886]]
```

```
print(lr_model.classes_)
```

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

(예측으로 사용한 클래스 속성 정보 확인
: 알파벳 순으로 정렬)

```
# 각 클래스 별 Z 값 계산
decision = lr_model.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]]
```

샘플 5개 데이터에 대한
각 클래스별 z 값 출력
높은 z 값 출력 클래스가 예측 클래스가 됨

```
from scipy.special import softmax
proba = softmax(decision, axis=1)
print(np.round(proba, decimals=5))
```

predict_proba() 출력결과 일치

```
[[0.00001 0.01351 0.84127 0.00031 0.13567 0.00667 0.00255]
 [0.      0.00256 0.04391 0.00003 0.00731 0.94619 0.00001]
 [0.00002 0.      0.03406 0.9348  0.01505 0.01604 0.00003]
 [0.01093 0.03405 0.30554 0.00661 0.56658 0.00007 0.07622]
 [0.      0.00037 0.904   0.00241 0.08947 0.00241 0.00133]]
```

Thank You

HANCOM

Template Visual Guide, version 1.0