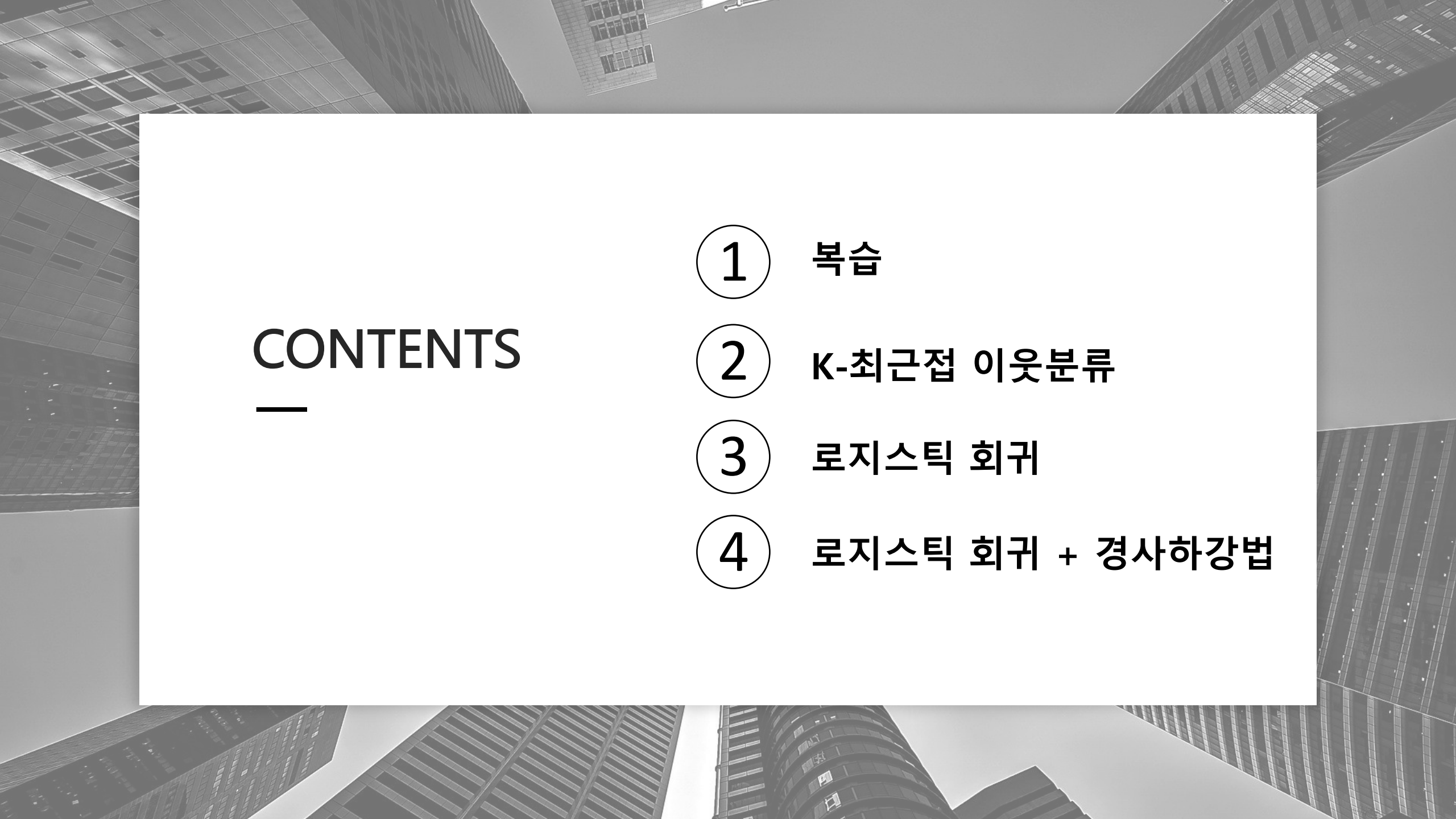
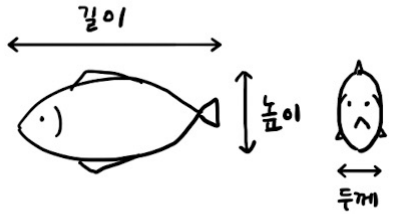
분류: 로지스틱 회귀 적용하기

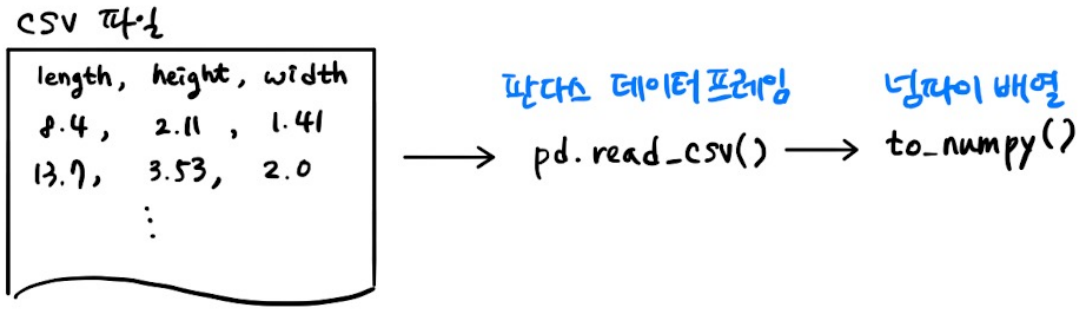
임 경 태



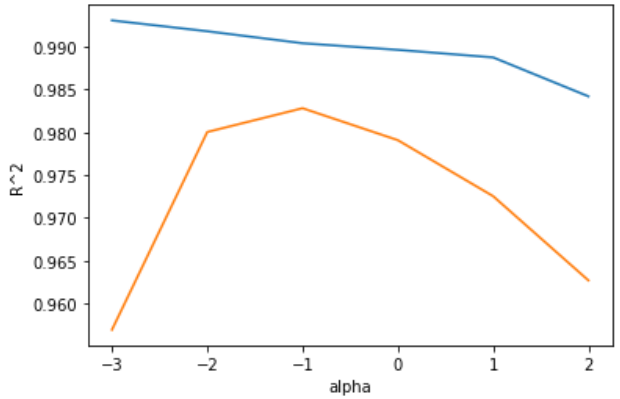
CONTENTS

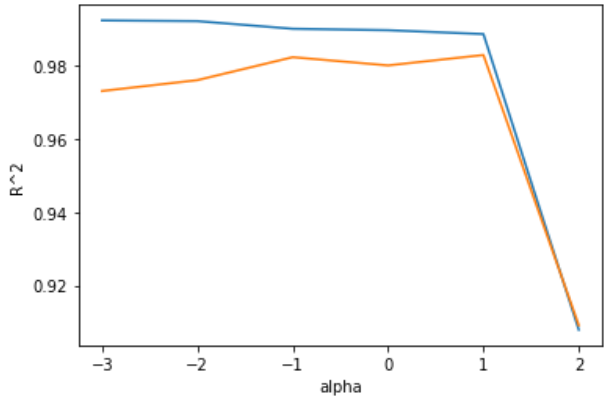
지난 시간 되돌아보기






```
poly = PolynomialFeatures(include_bias=False)

poly.fit(train_input)
train_poly = poly.transform(train_input)

print(train_poly.shape)
(42, 9)

poly.get_feature_names()
['x0', 'x1', 'x2', 'x0^2', 'x0 x1',
 'x0 x2', 'x1^2', 'x1 x2', 'x2^2']

test_poly = poly.transform(test_input)
```





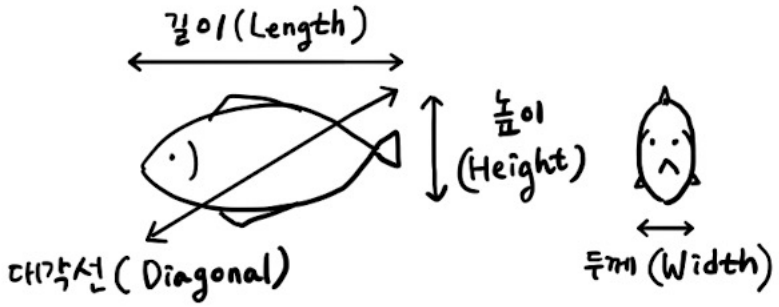
CONTENTS

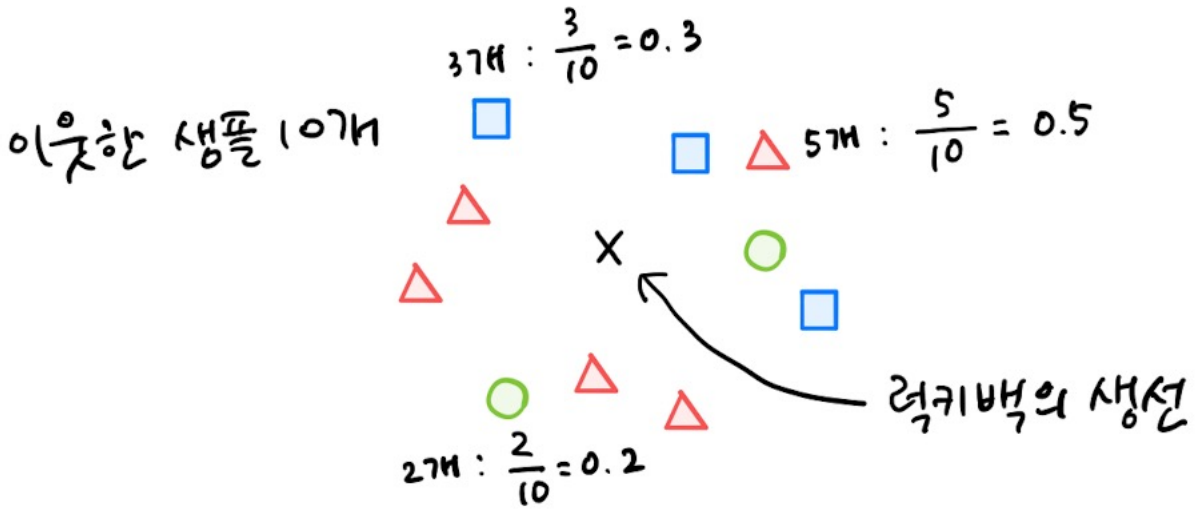
내가 잡은 고기가 어떤 생선인지 분류할 수 있지 않을까?

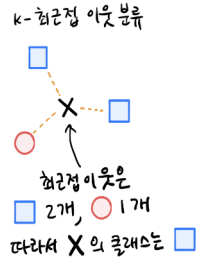
- 목적: 생선 분류기를 만들어 보자
- 입력: 길이, 높이, 두께, 대각선 길이
- 출력: 입력 데이터에 따른 예측한 생선의 종류
- 모델: ?

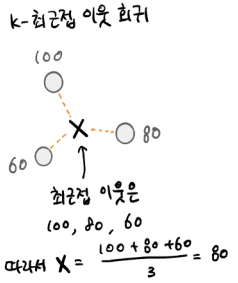
K-최근접 분류 기반의 모델

예측하려는 샘플과 가장 가까운 샘플 K개의 평균을 이용한 분류









데이터 가공

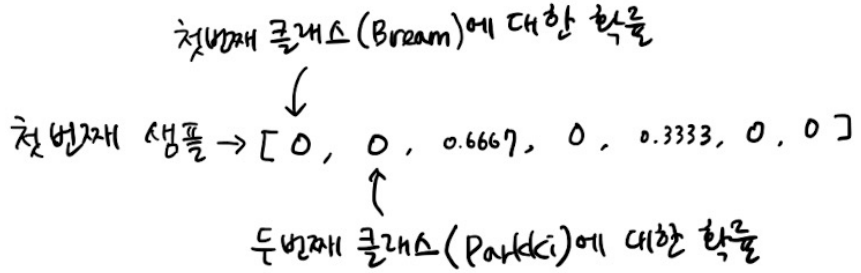
```
import pandas as pd

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

	Species	Weight	Length	Diagonal	Height	Width
0	Bream	242.0	25.4	30.0	11.5200	4.0200
1	Bream	290.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	26.5	31.1	12.3778	4.6961
3	Bream	363.0	29.0	33.5	12.7300	4.4555
4	Bream	430.0	29.0	34.0	12.4440	5.1340

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

fish_target = fish['Species'].to_numpy()
```

KNN 모델 학습 및 결과 확인

```
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(train_scaled, train_target)

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

print(kn.predict(test_scaled[:5]))
['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']

proba = kn.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=4))
[[0.     0.     1.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     1.     0.    ]
 [0.     0.     0.     1.     0.     0.     0.    ]
 [0.     0.     0.6667 0.     0.3333 0.     0.    ]
 [0.     0.     0.6667 0.     0.3333 0.     0.    ]]
```

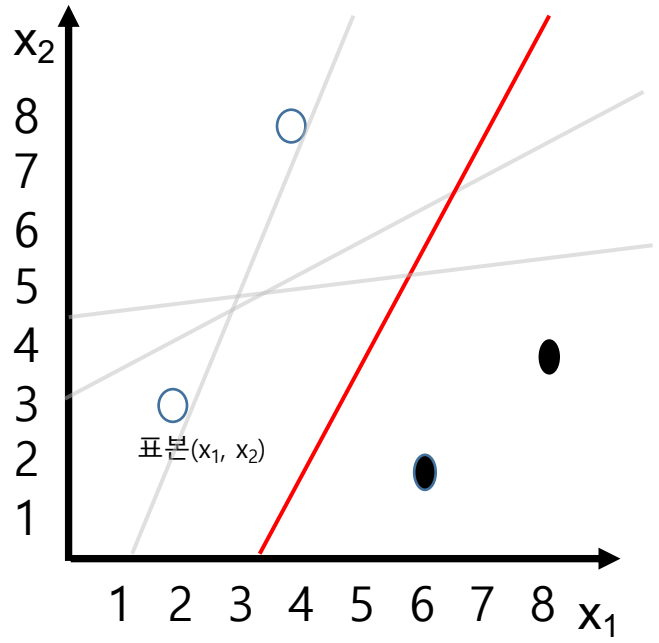


CONTENTS

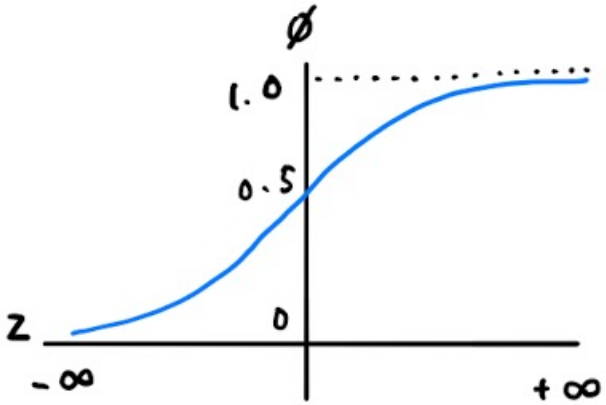
예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 입력: 선형회귀의 결과를 0~1사이로 변경하는 sigmoid 활용

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



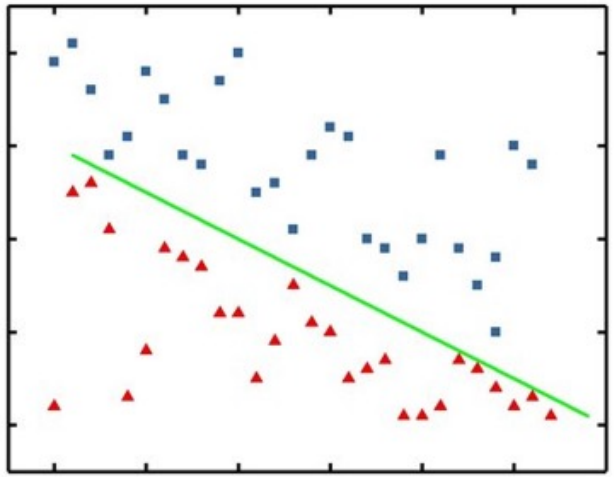
$$\phi = \frac{1}{1 + e^{-z}}$$



예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 일단 도미와 빙어를 구분하는 이진분류(Binary Logistic Regression) 모델을 만들어보자

```
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)

print(lr.predict(train_bream_smelt[:5]))
['Bream' 'Smelt' 'Bream' 'Bream' 'Bream']

print(lr.predict_proba(train_bream_smelt[:5]))
[[0.99759855 0.00240145]
 [0.02735183 0.97264817]
 [0.99486072 0.00513928]
 [0.98584202 0.01415798]
 [0.99767269 0.00232731]]
```



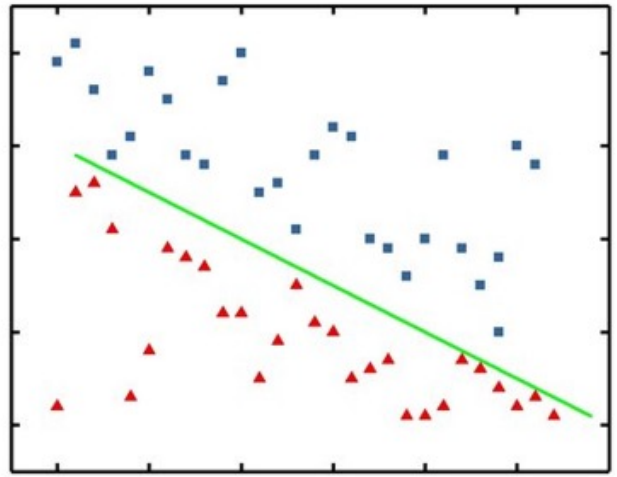
(a) Logistic Regression

예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 일단 도미와 빙어를 구분하는 이진분류(Binary Logistic Regression) 모델을 만들어보자

- Linear 계수 확인: z = -0.43*weight - 0.57*length -0.66*diagonal -1.013*height -0.731*width - 2.1615

```
print(lr.coef_, lr.intercept_)
[[-0.4037798  -0.57620209 -0.66280298 -1.01290277 -0.73168947]] [-2.16155132]
```



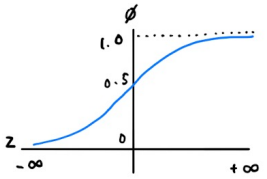
(a) Logistic Regression

- 각 샘플의 z값 확인

```
decisions = lr.decision_function(train_bream_smelt[:5])
print(decisions)
[-6.02927744  3.57123907 -5.26568906 -4.24321775 -6.0607117 ]

from scipy.special import expit
print(expit(decisions))
[0.00240145 0.97264817 0.00513928 0.01415798 0.00232731]
```

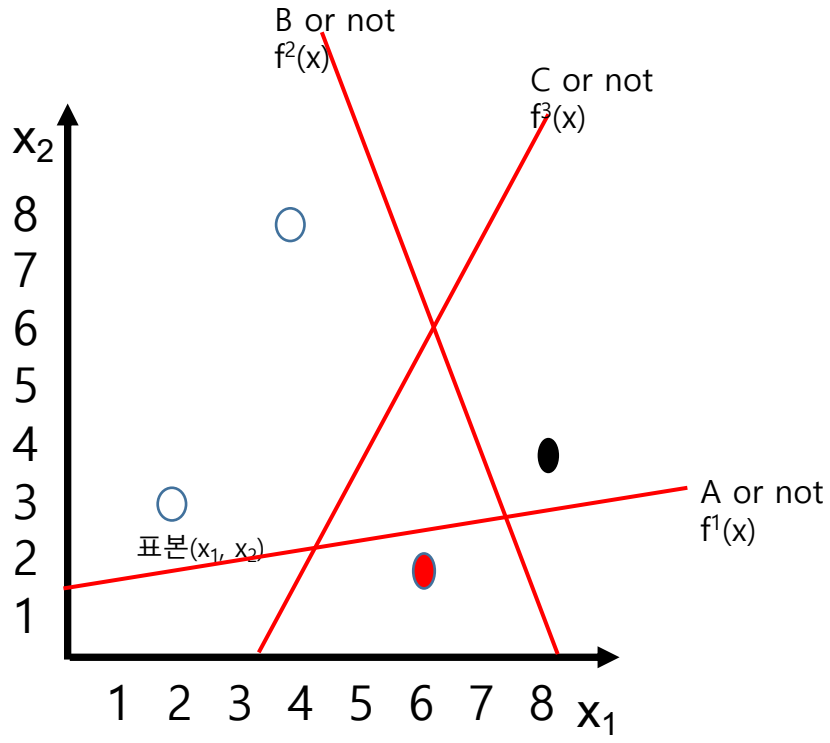
- 각 샘플의 z값에 sigmoid를 적용한 값

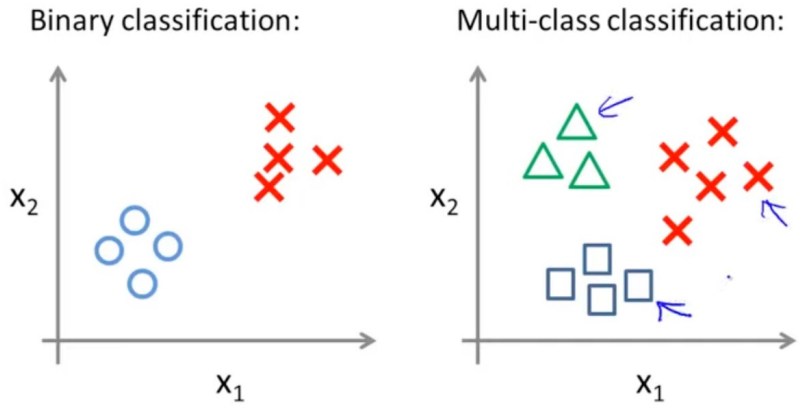
$$\phi = \frac{1}{1 + e^{-z}}$$



예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 일단 도미와 빙어 꽁치 등 7개의 생선을 구분하는 다중분류(Multinomial Logistic Regression) 모델을 만들어보자



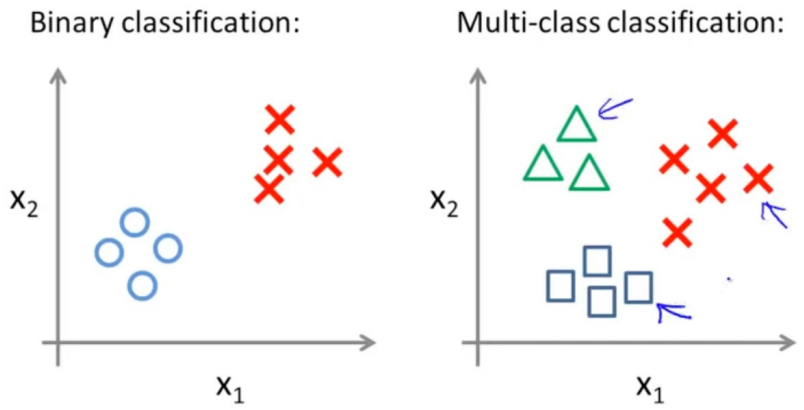


예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 일단 도미와 빙어 꽁치 등 7개의 생선을 구분하는 다중분류(Multinomial Logistic Regression) 모델을 만들어보자

C: 릿지회귀에서 설명한 L2규제 기본값은 10이나 20으로 설정 (규제를 조금 완화하기위해)

```
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))
0.9327731092436975
0.925

proba = lr.predict_proba(test_scaled[:5])
print(np.round(proba, decimals=3))
[[0.    0.014 0.841 0.    0.136 0.007 0.003]
 [0.    0.003 0.044 0.    0.007 0.946 0.   ]
 [0.    0.    0.034 0.935 0.015 0.016 0.   ]
 [0.011 0.034 0.306 0.007 0.567 0.    0.076]
 [0.    0.    0.904 0.002 0.089 0.002 0.001]]

print(lr.coef_.shape, lr.intercept_.shape)
(7, 5) (7,)
```



예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 일단 도미와 빙어 꽁치 등 7개의 생선을 구분하는 다중분류(Multinomial Logistic Regression) 모델을 만들어보자

  - Softmax 함수를 이용해서 출력값을 확률 분포값으로 변경하자!

```
decision = lr.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]]
```

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{x_j}}$$

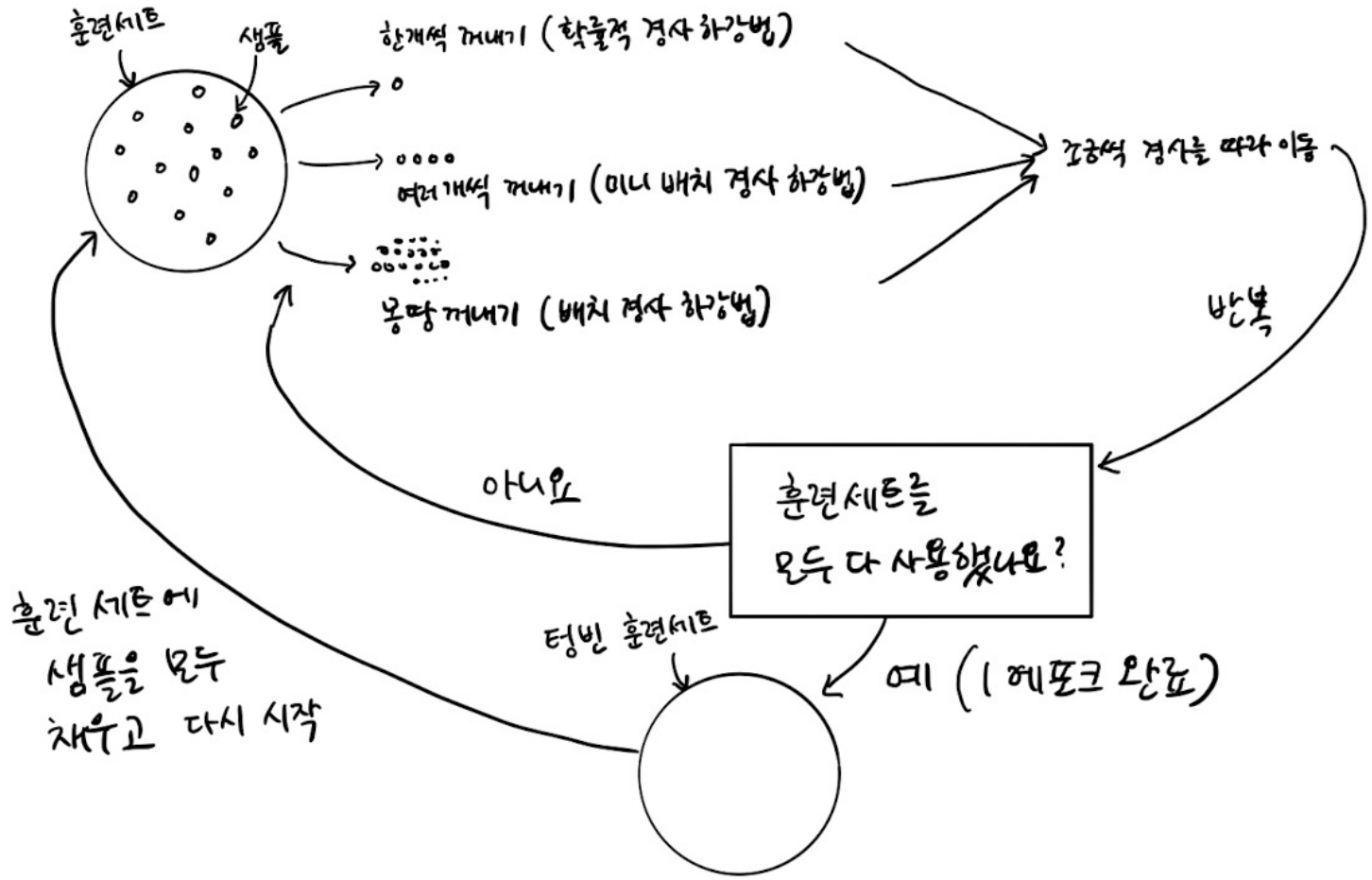
```
from scipy.special import softmax

proba = softmax(decision, axis=1)
print(np.round(proba, decimals=3))
[[0.    0.014 0.841 0.    0.136 0.007 0.003]
 [0.    0.003 0.044 0.    0.007 0.946 0.   ]
 [0.    0.    0.034 0.935 0.015 0.016 0.   ]
 [0.011 0.034 0.306 0.007 0.567 0.    0.076]
 [0.    0.    0.904 0.002 0.089 0.002 0.001]]
```

$$S1 = \frac{e^{z1}}{e\_sum}, S2 = \frac{e^{z2}}{e\_sum}, \cdots, S7 = \frac{e^{z7}}{e\_sum}$$

CONTENTS

경사하강법+로지스틱 회귀 기반의 모델

경사하강법을 활용해 예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

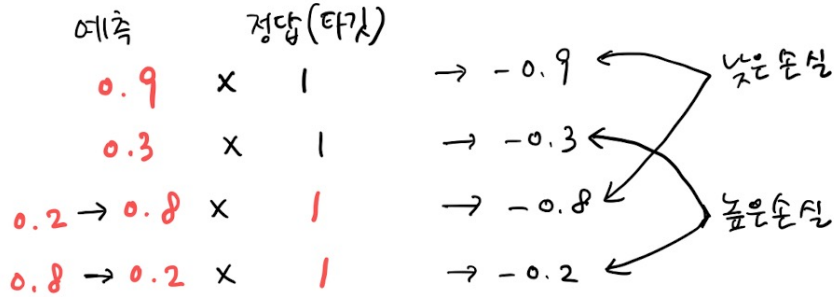


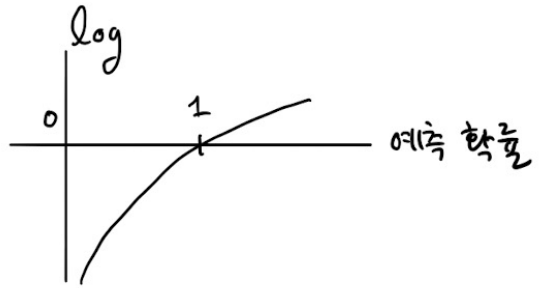
경사하강법+로지스틱 회귀 기반의 모델

경사하강법을 활용해 예측하려는 데이터를 가장 잘 구분하는 Linear 선을 이용한 분류

- 경사하강법을 활용한 모델은 정답과 예측값의 차이인 손실값 (Loss, Cost)을 활용해 모델 파라미터 (w,b)값을 업데이트 한다. 따라서 손실값 정의 필요.

Binary Cross Entropy
손실함수





타깃 = 1 일때
→ - log (예측확률)
타깃 = 0 일때
→ - log (1 - 예측확률)

Cross Entropy
손실함수

$$CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{N_c} y_i \log(\hat{y}_i)$$

데이터 가공

```
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()


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)


from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss.fit(train_input)
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

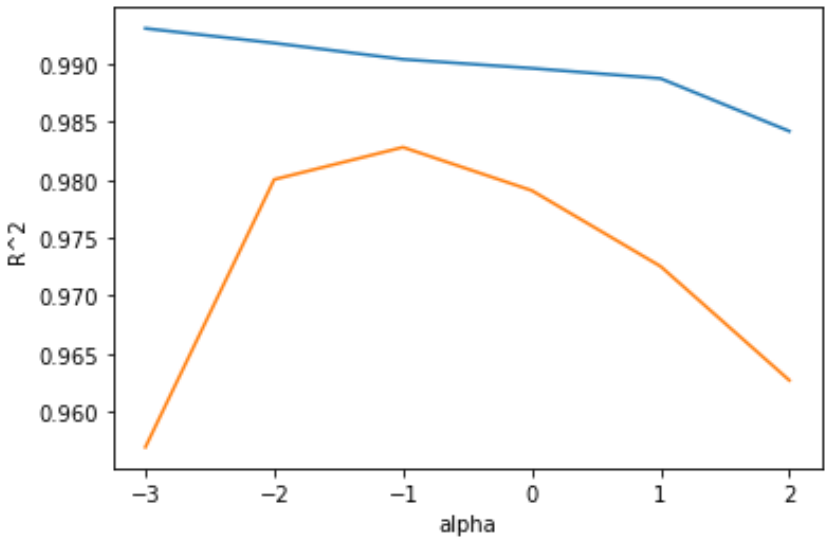
모델학습 및 성능평가

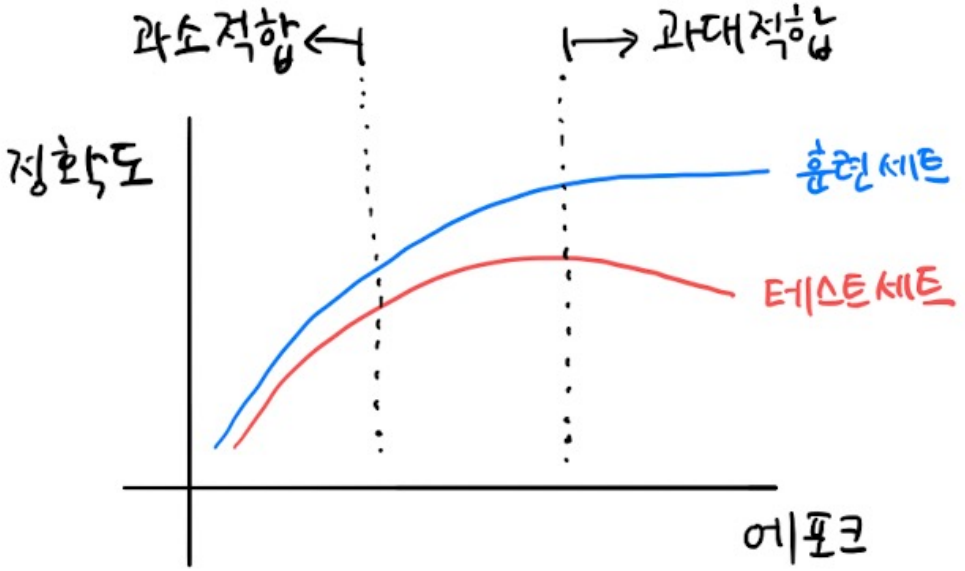
```
from sklearn.linear_model import SGDClassifier

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

print(sc.score(train_scaled, train_target))
0.773109243697479
print(sc.score(test_scaled, test_target))
0.775
```

```
sc.partial_fit(train_scaled, train_target)

print(sc.score(train_scaled, train_target))
0.8151260504201681
print(sc.score(test_scaled, test_target))
0.825
```

분석 및 피드백

- 학습을 오랫동안 진행할 경우 테스트 데이터의 성능이 떨어지는 과대 적합 발생

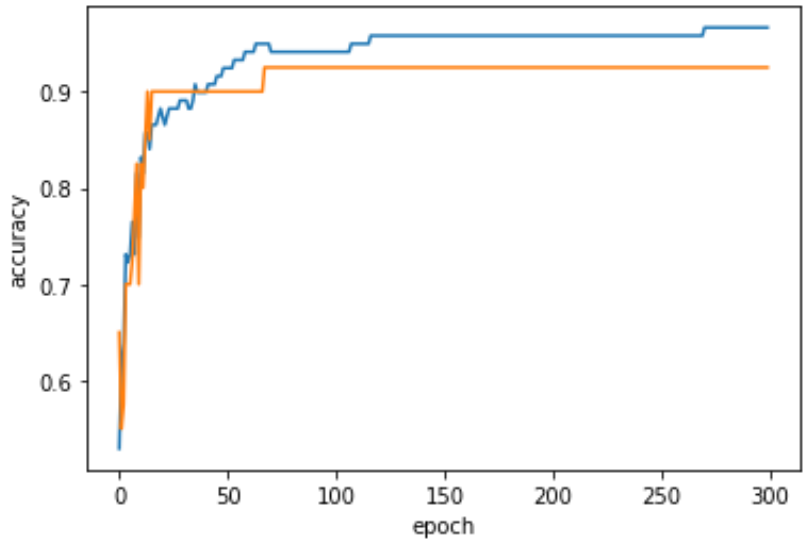




분석 및 피드백 반영 (Early Stopping)

- 조기종료 방법을 이용해 과대적합 해결 (과대적합이 시작하기 전에 훈련을 멈추자)
  - tol 매개 변수를 사용하여 일정 에포크 동안 성능이 향상되지 않을 경우 자동으로 멈추 도록할 수 있음

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


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))
0.957983193277311
print(sc.score(test_scaled, test_target))
0.925
```



감사합니다.