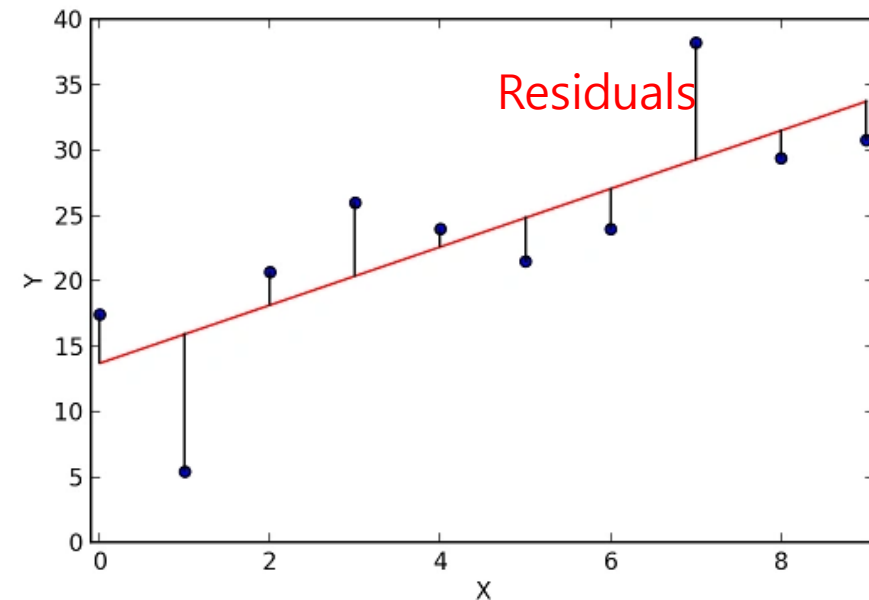


How to Find the Line of Best Fit for Lasso and ridge regression



- The main purpose of the **best fit line** is that our predicted values should be closer to our actual or the observed values
- **Minimize** the difference between the values predicted by us and the observed values, and which is actually termed as **error**



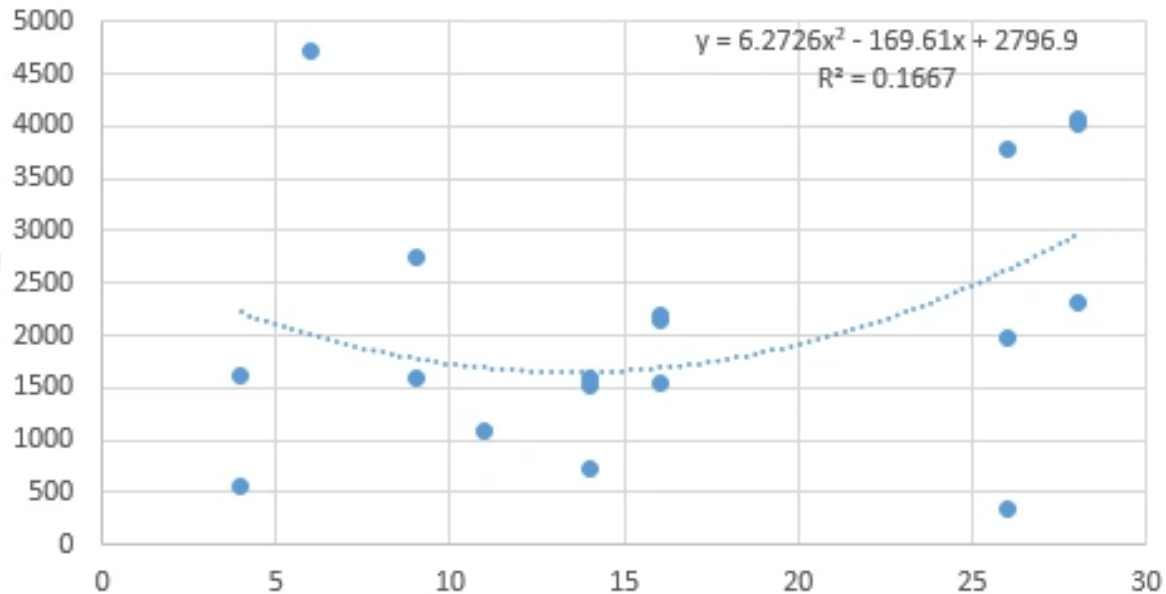
$$SS_{residuals} = \sum_{i=1}^m (h(x) - y)^2$$

Sum of square of residuals

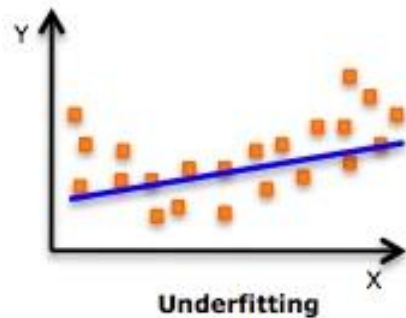
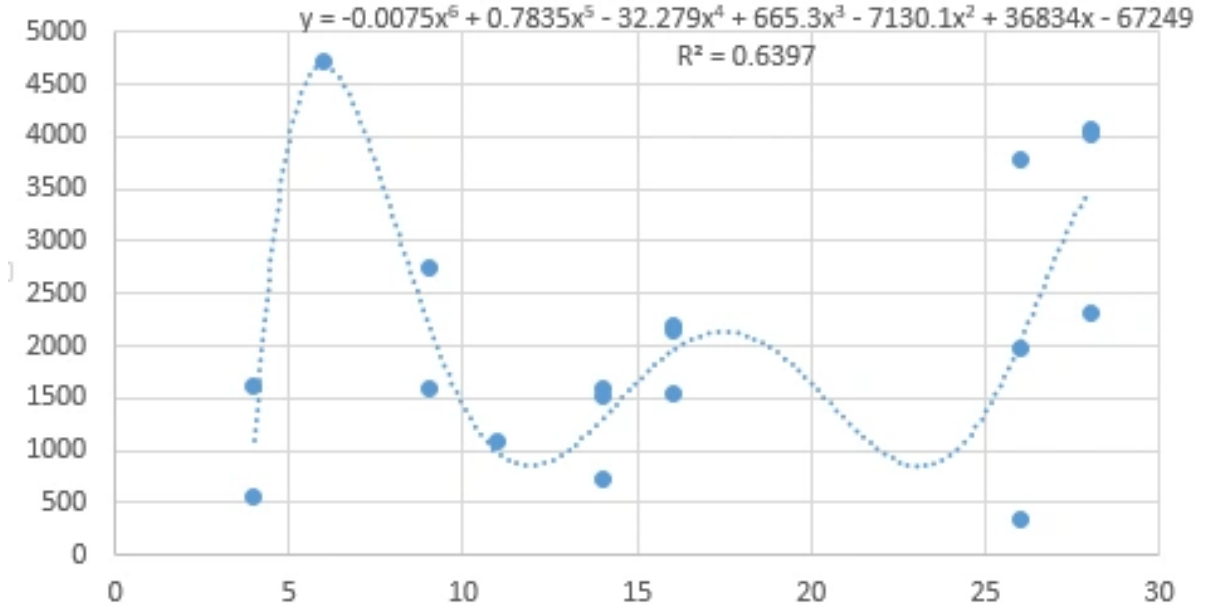
- **penalize higher error value much more as compared to a smaller one**, so that there is a significant difference between making big errors and small errors, which makes it easy to differentiate and select the best fit line.

Polynomial Regression

Sales vs Establishment Year



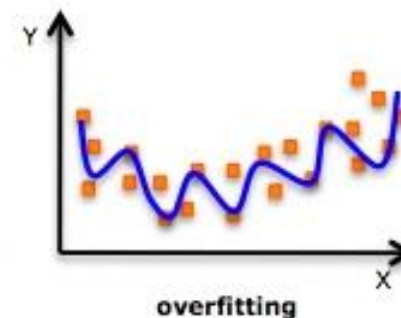
Sales vs Establishment Year



high bias and low variance

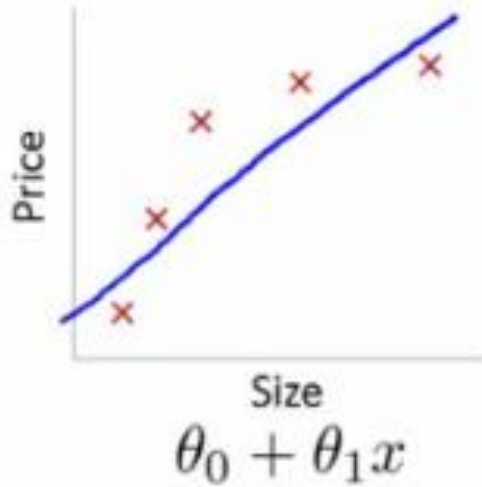


Just right!

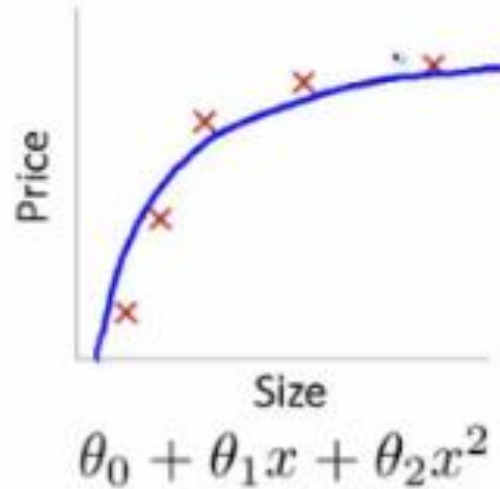


high variance and low bias

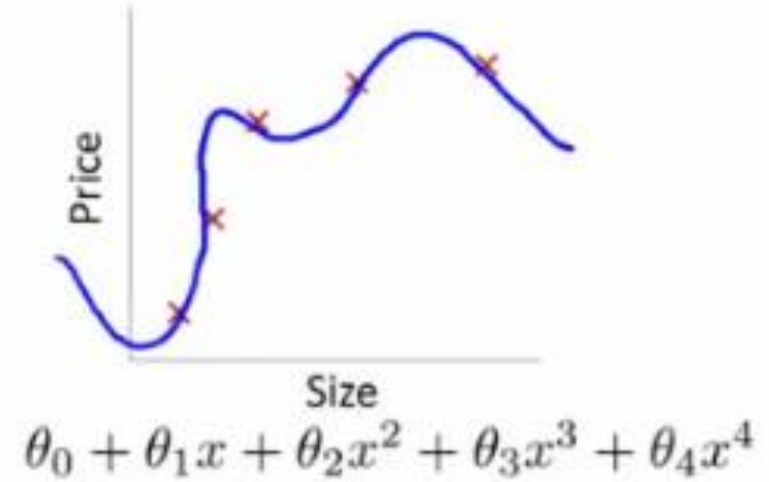
Polynomial Regression



High bias
(underfit)



“Just right”



High variance
(overfit)

Ridge and Lasso Regression (Advanced)

Ridge Regression (L2 regularization)

- A penalty equal to the ***square** of the magnitude of coefficients to the loss function
- Ridge regression shrinks the coefficients towards zero, it does not set any coefficients exactly to zero, thus **retaining all predictors** in the model.

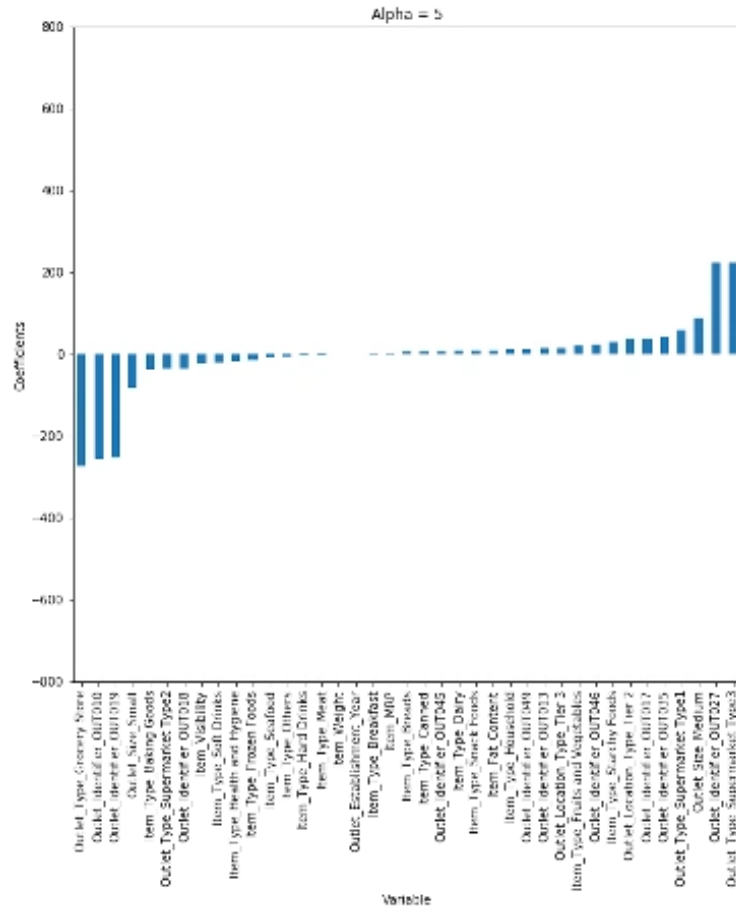
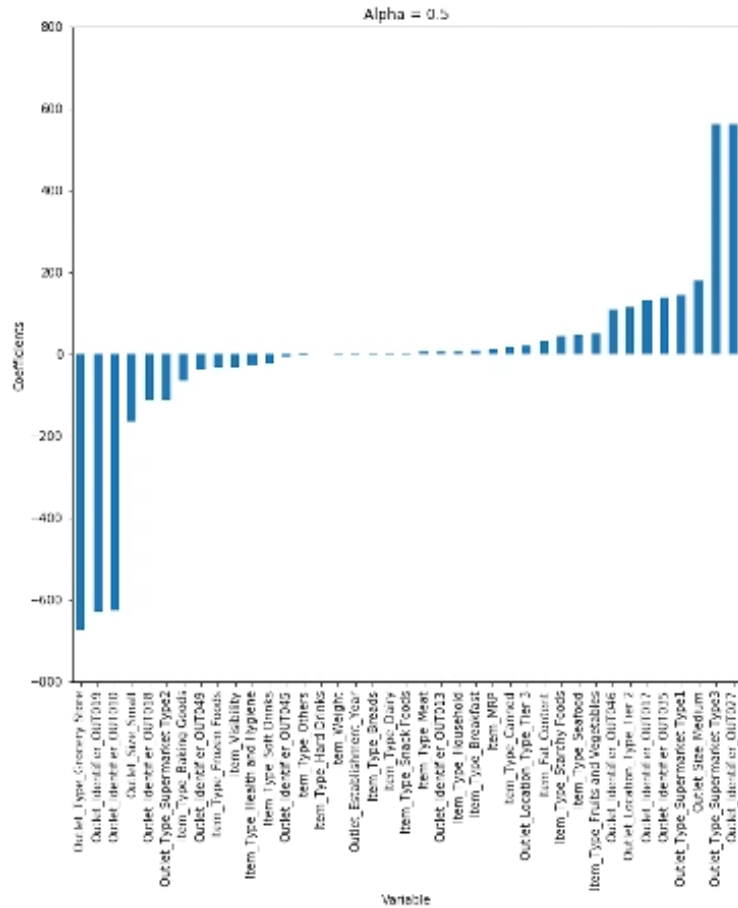
*0⁰은 일반적으로 정의되지 않는 값으로, 극한에서 대표적인 부정형 중 하나

Lasso Regression (L1 regularization)

- A penalty equal to the **absolute value** of the coefficients
- Lasso to set some coefficients exactly to zero, effectively performing **feature selection** and simplifying the model by excluding less important predictors

Ridge and Lasso Regression (Advanced)

Ridge Regression (L2 regularization)

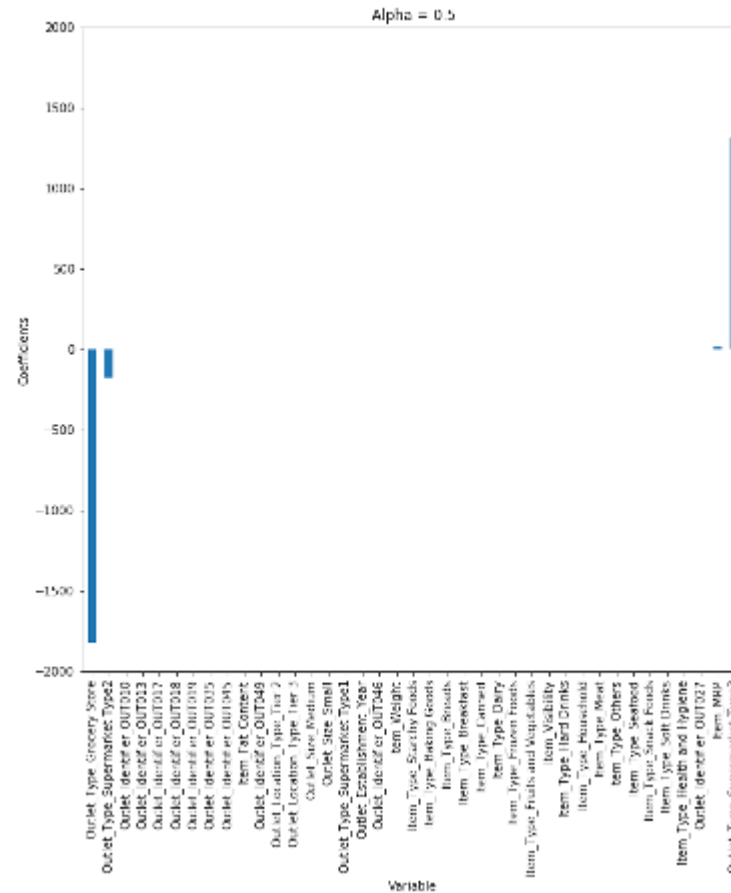


$$\min \left(||Y - X(\theta)||_2^2 + \lambda ||\theta||_2^2 \right)$$

- It shrinks the parameters, therefore it is mostly used to prevent multicollinearity.
- It reduces the model complexity by coefficient shrinkage.
- It uses L2 regularization technique.

Ridge and Lasso Regression (Advanced)

Lasso Regression (L1 regularization)



$$\min \left(\|Y - X\theta\|_2^2 + \lambda \|\theta\|_1 \right)$$

- It uses L1 regularization technique
- It is generally used when there are more features, as it automatically performs feature selection.

Bias-Variance Tradeoff

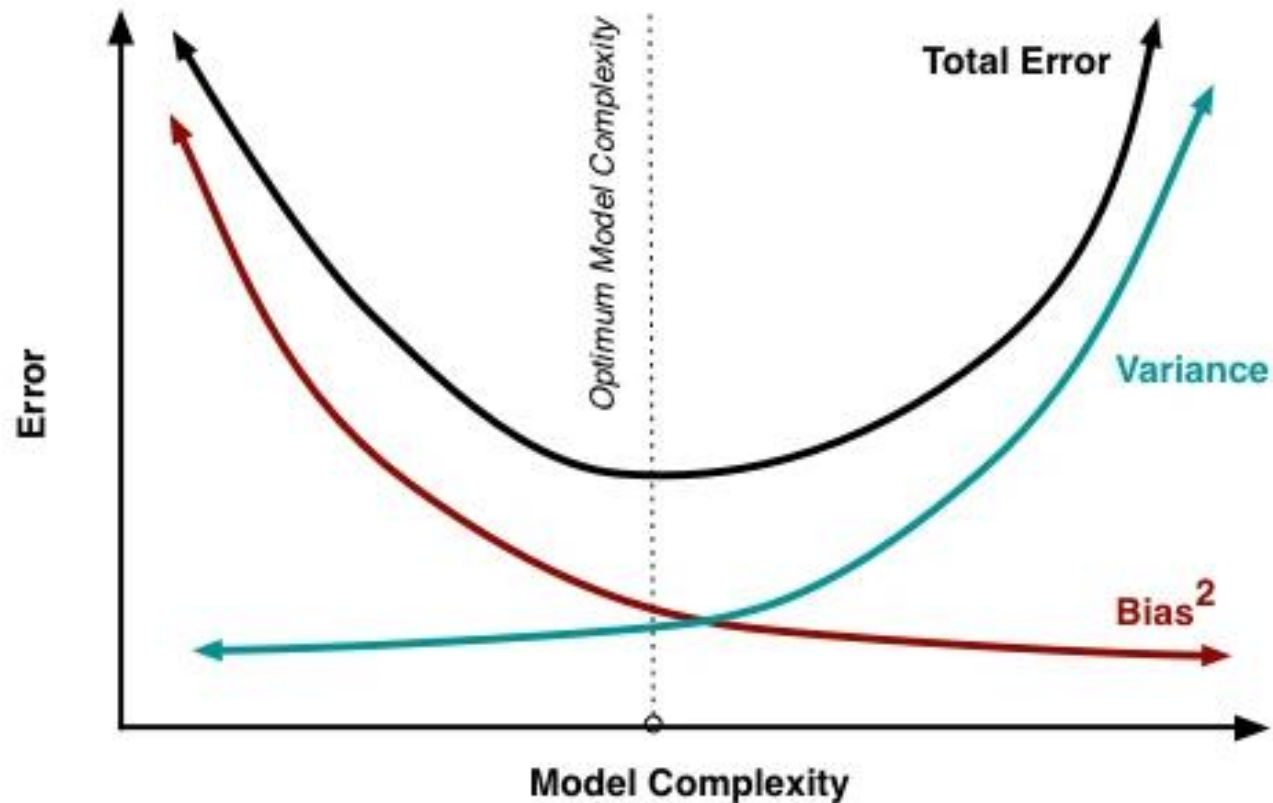
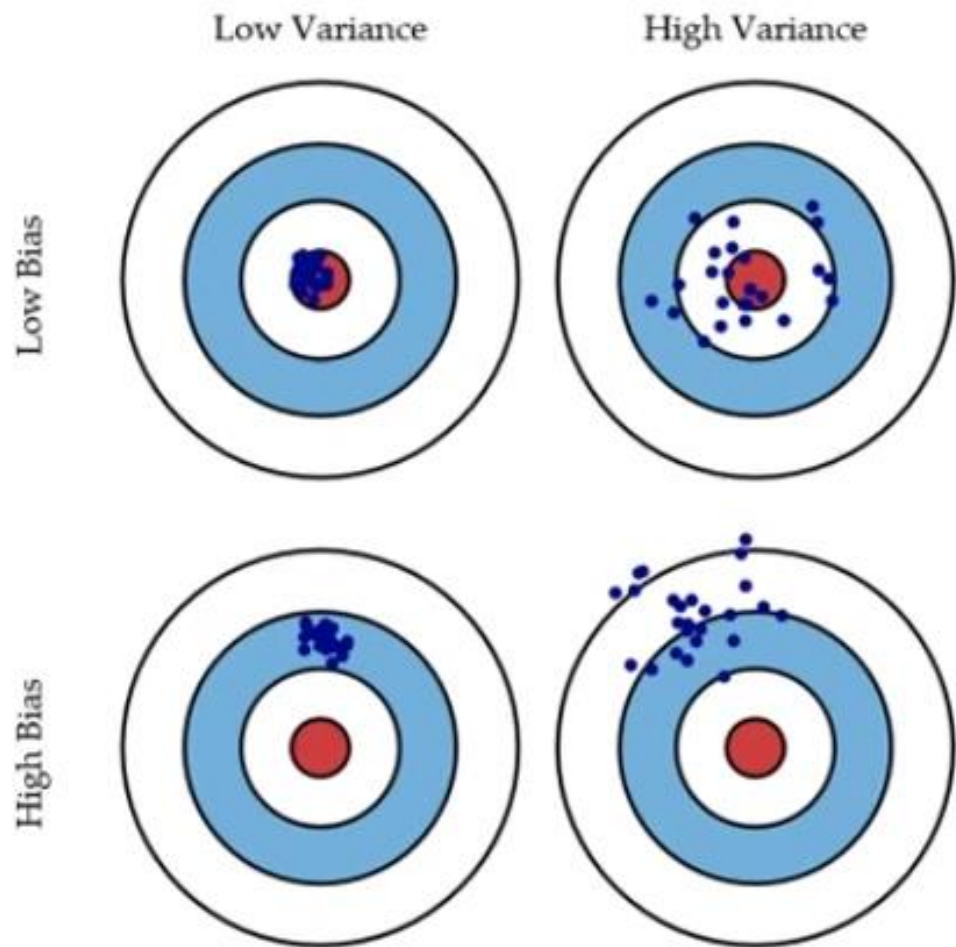
Ridge Regression (L2 regularization)

- Tends to perform better when there are **many predictors** that contribute to the outcome, as it shrinks coefficients but retains all variables
- **Lower variance** in predictions, especially in high-dimensional spaces, but may not simplify the model significantly

Lasso Regression (L1 regularization)

- Reduce variance by eliminating irrelevant features, which can also lead to a **simpler and more interpretable** model
- Introduce **higher bias** if important predictors are mistakenly set to zero due to the L1 penalty

Ridge and Lasso Regression (Advanced)



Ridge and Lasso Regression (Advanced)

Ridge Regression (L2 regularization)

- A penalty equal to the ***square** of the magnitude of coefficients to the loss function
- Ridge regression shrinks the coefficients towards zero, it does not set any coefficients exactly to zero, thus **retaining all predictors** in the model.

*0⁰은 일반적으로 정의되지 않는 값으로, 극한에서 대표적인 부정형 중 하나

Lasso Regression (L1 regularization)

- A penalty equal to the **absolute value** of the coefficients
- Lasso to set some coefficients exactly to zero, effectively performing **feature selection** and simplifying the model by excluding less important predictors

Types of Regularization Techniques (advanced)

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

y is the predicted output.

x_1 and x_2 are input features (independent variables).

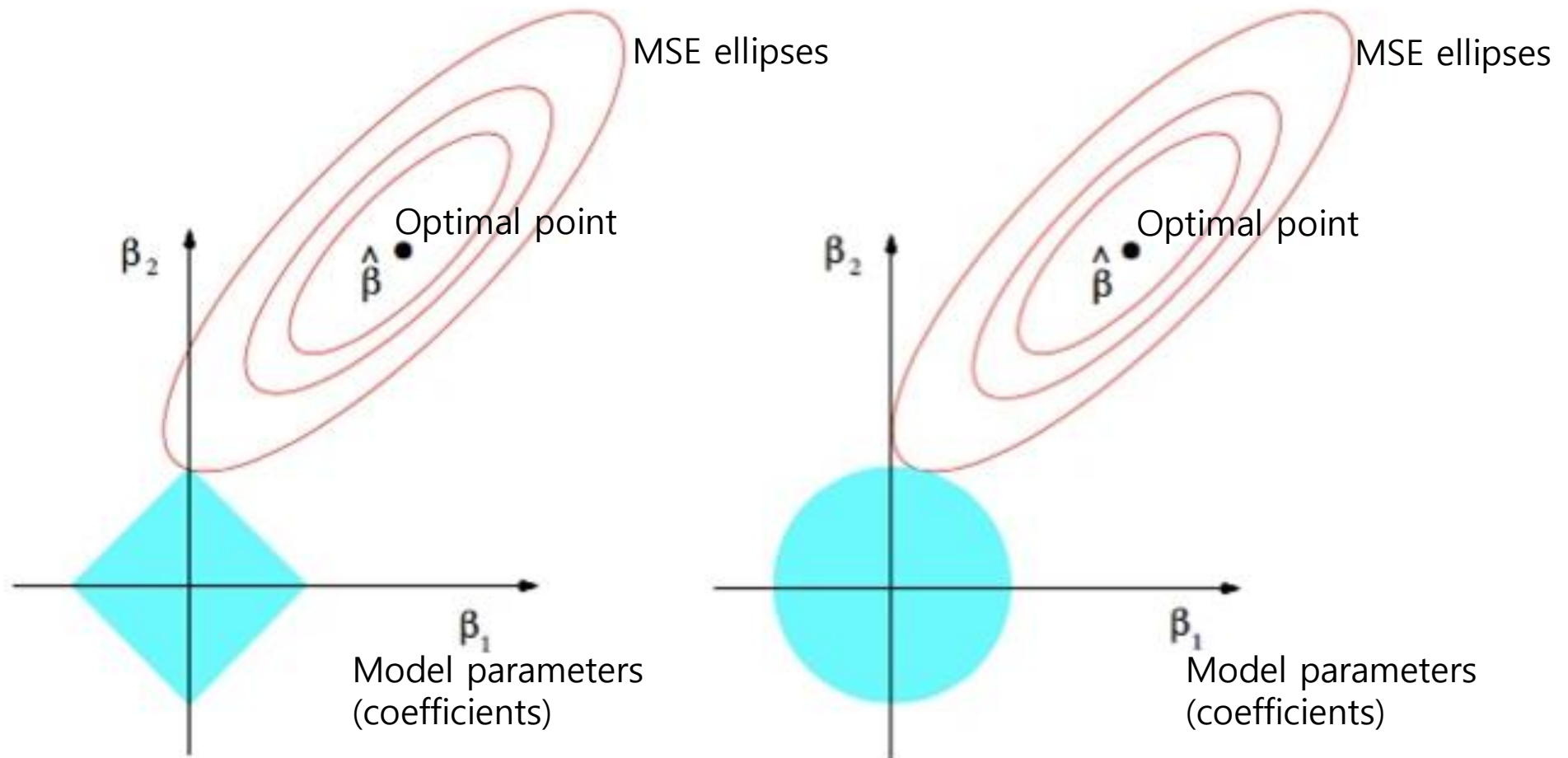
β_0 is the intercept (bias).

β_1 and β_2 are the coefficients (weights) for feature 1 and feature 2 respectively.

The goal is to find the optimal combination of β_1 and β_2 that minimizes the loss function (MSE + regularization)

Types of Regularization Techniques

This is visually equivalent to finding the first point where the MSE contour touches (is tangent to) the constraint region (blue shape).



Types of Regularization Techniques

A linear model with two parameters, β_1 and β_2 :

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}))^2$$

Fix the bias term β_0 (or just ignore it for now), this becomes a quadratic function of β_1 and β_2 .

$$\text{MSE} = a\beta_1^2 + b\beta_2^2 + c\beta_1\beta_2 + d\beta_1 + e\beta_2 + \text{constant}$$

This is the general form of a multivariable quadratic function — and when you plot such a function in 2D (β_1 vs β_2), its contour lines (i.e., lines of equal MSE) form ellipses.

Types of Regularization Techniques

$$\text{L1 penalty} = \|\beta\|_1 = |\beta_1| + |\beta_2|$$

L1 constraint defines a region: $|\beta_1| + |\beta_2| \leq t$ Where t represents constraint budget — max allowed "size" of β

The corners of this diamond lie on the axes (where either β_1 or $\beta_2 = 0$).

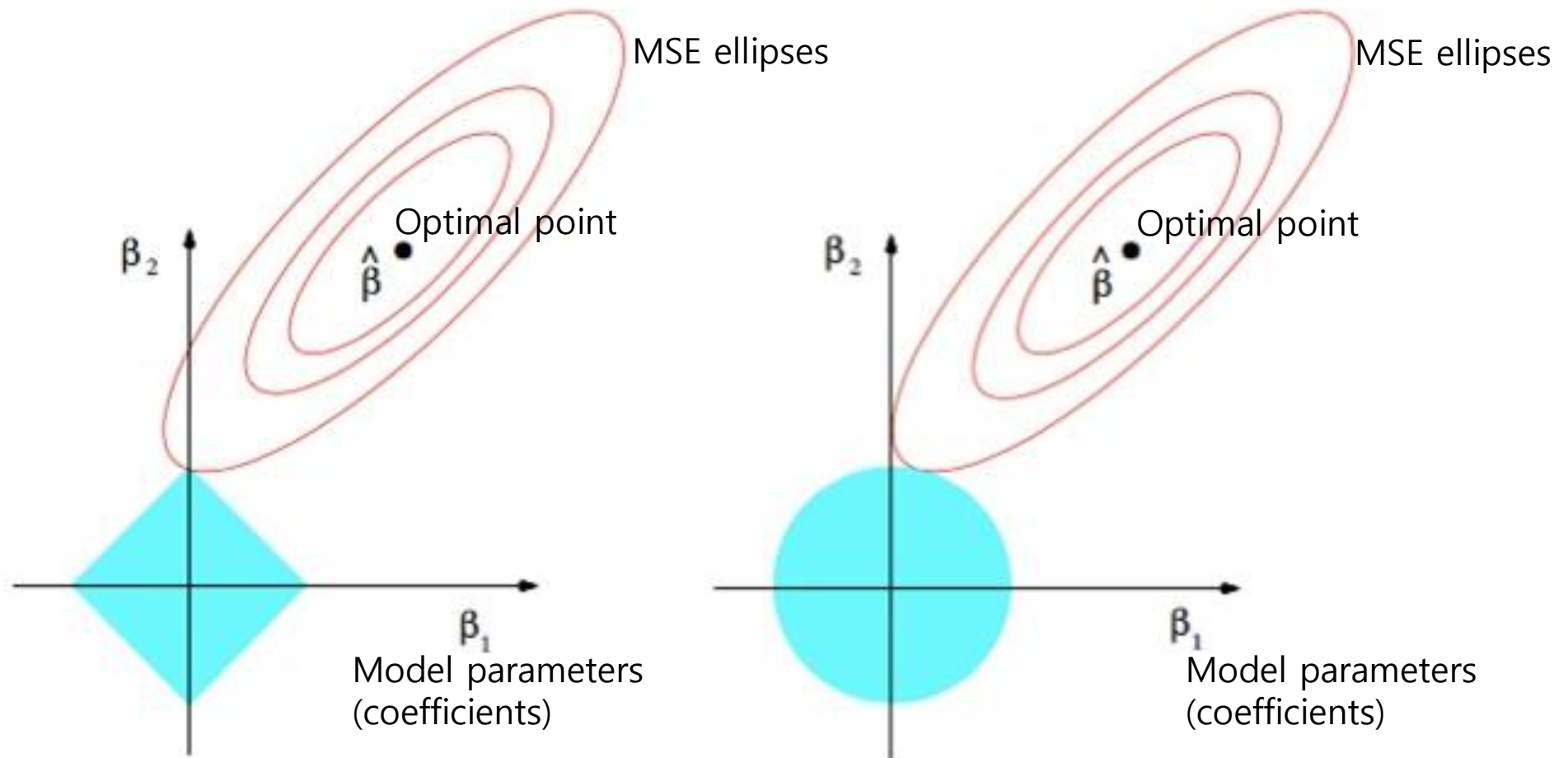
$$\text{L2 penalty} = \|\beta\|_2^2 = \beta_1^2 + \beta_2^2$$

L2 constraint defines a region: $\beta_1^2 + \beta_2^2 \leq t$ Where t represents constraint budget — max allowed "size" of β

This is the equation of a circle (or a sphere in higher dimensions), centered at the origin

Types of Regularization Techniques

This is visually equivalent to finding the first point where the MSE contour touches (is tangent to) the constraint region (blue shape).



학습목표

- 로지스틱 회귀, 확률적 경사 하강법과 같은 분류 알고리즘을 배웁니다.
- 이진 분류와 다중 분류의 차이를 이해하고 클래스별 확률을 예측합니다.

Chapter

04

다양한 분류 알고리즘

러키백의 확률을 계산하라!

04-1

로지스틱 회귀

핵심 키워드

로지스틱 회귀

다중 분류

시그모이드 함수

소프트맥스 함수

로지스틱 회귀 알고리즘을 배우고 이진 분류 문제에서 클래스 확률을 예측합니다.

시작하기 전에

혼공머신은 분류와 회귀 문제까지 다룰 수 있는 경험을 쌓았고 특성값을 전처리하거나 특성을 조합하여 새로운 특성을 만들 수 있는 경지까지 올랐습니다. 한빛 마켓의 마케팅 팀은 다가오는 명절 특수에 고객의 이목을 끌 새로운 이벤트를 기획했습니다. 그 이름은 바로 ‘한빛 럭키백’입니다!

럭키백은 구성품을 모른 채 먼저 구매하고, 배송받은 다음에야 비로소 구성품을 알 수 있는 상품입니다. 기간 한정으로 판매하는 럭키백의 물품은 생선으로 한정하기로 했습니다. 아주 독특한 아이디어죠? 요즘에는 이렇게 튀어야 입소문이 나는 법이죠. 하지만 럭키백 이벤트 소식을 들은 고객 만족 팀은 강하게 반대했습니다.

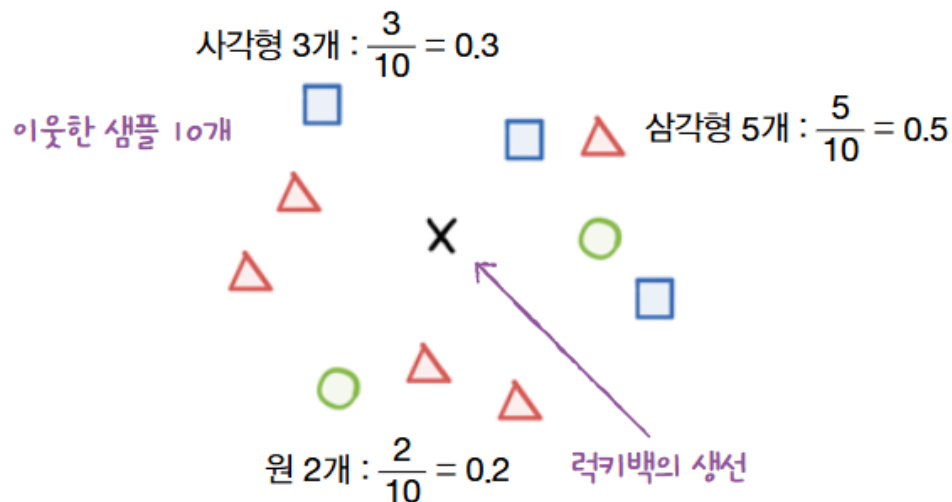


어쩔 수 없이 마케팅 팀은 럭키백에 포함된 생선의 확률을 알려주는 방향으로 이벤트 수정안을 내놓았습니다. 가령 A 럭키백에 도미 확률이 높다고 표시하면 도미를 원하는 고객은 A 럭키백을 구매할 겁니다. 음... 그렇다면 어떻게 생선의 확률을 구할지가 문제겠네요. 혹시 머신러닝으로 럭키백의 생선이 어떤 타겟에 속하는지 확률을 구할 수 있을까요? 김 팀장이 혼공머신을 다시 부릅니다.



도미일 확률 : 72%
빙어일 확률 : 16%
⋮

“k-최근접 이웃은 주변 이웃을 찾아주니까 이웃의 클래스 비율을 확률이라고 출력하면 되지 않을까?”



그림을 보면 샘플 X 주위에 가장 가까운 이웃 샘플 10개를 표시했습니다. 사각형이 3개, 삼각형이 5개, 원이 2개입니다. 이웃한 샘플의 클래스를 확률로 삼는다면 샘플 X가 사각형일 확률은 30%, 삼각형일 확률은 50%, 원일 확률은 20%입니다. 아주 쉽네요.

데이터 준비하기

모델 훈련에 사용할 데이터를 만들어 보겠습니다. 3장처럼 판다스를 사용할 텐데, 이번에도 인터넷에서 직접 CSV 데이터를 읽어 들일 겁니다. 판다스의 `read_csv()` 함수로 CSV 파일을 데이터프레임으로 변환한 다음 `head()` 메서드로 처음 5개 행을 출력해 보겠습니다.

note 이 파일 내용을 직접 보고 싶다면 https://bit.ly/fish_csv_data로 접속해 보세요.

손코딩

```
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

가장 왼쪽의 0, 1, 2는 행 번호는 Pandas의 index임. CSV파일의 첫 줄을 인식해서 열 제목으로 만듦.

손코딩

```
print(pd.unique(fish['Species']))
```



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

이 데이터프레임에서 Species 열을 타겟으로 만들고 나머지 5개 열은 입력 데이터로 사용하겠습니다. 데이터프레임에서 열을 선택하는 방법은 간단합니다. 데이터프레임에서 원하는 열을 리스트로 나열하면 됩니다. Species 열을 빼고 나머지 5개 열을 선택해 보겠습니다.

손코딩

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

손코딩

```
print(fish_input[:5])
```

```
[[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 ]]
```

앞에서 fish 데이터프레임을 출력한 값과 비교해 보세요. 입력 데이터가 잘 준비되었는지 확인했으면 이제 동일한 방식으로 타깃 데이터를 만들겠습니다.

note Species 열을 선택할 때 fish[['Species']]와 같이 두 개의 괄호를 사용하지 않도록 주의하세요. 이렇게 하면 fish_target이 2차원 배열이 됩니다.

손코딩

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

이제 데이터를 훈련 세트와 테스트 세트로 나눕니다. 이제 이런 작업이 익숙하게 느껴지면 좋겠습니다.

2장에서 배웠듯이 머신러닝에서는 기본으로 데이터 세트 2개가 필요합니다.

손코딩

```
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)
```

그다음 사이킷런의 StandardScaler 클래스를 사용해 훈련 세트와 테스트 세트를 표준화 전처리하겠습니다. 여기에서도 훈련 세트의 통계 값으로 테스트 세트를 변환해야 한다는 점을 잊지 마세요.

손코딩

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

k-최근접 이웃 분류기의 확률 예측

2장에서 했던 것처럼 사이킷런의 KNeighborsClassifier 클래스 객체를 만들고 훈련 세트로 모델을 훈련한 다음 훈련 세트와 테스트 세트의 점수를 확인해 보겠습니다. 최근접 이웃 개수인 k를 3으로 지정하여 사용합니다.

손코딩

```
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))
```



0.8907563025210085

0.85

앞서 fish 데이터프레임에서 7개의 생선이 있었던 것을 기억하나요? 타깃 데이터를 만들 때 fish['Species']를 사용해 만들었기 때문에 훈련 세트와 테스트 세트의 타깃 데이터에도 7개의 생선 종류가 들어가 있습니다. 이렇게 타깃 데이터에 2개 이상의 클래스가 포함된 문제를 다중 분류 multi-class classification라고 부릅니다.

하지만 조금 전 코드에서 보듯이 2장에서 만들었던 이진 분류와 모델을 만들고 훈련하는 방식은 동일합니다. 이진 분류를 사용했을 때는 양성 클래스와 음성 클래스를 각각 1과 0으로 지정하여 타깃 데이터를 만들었습니다. 다중 분류에서도 타깃값을 숫자로 바꾸어 입력할 수 있지만 사이킷런에서는 편리하게도 문자열로 된 타깃값을 그대로 사용할 수 있습니다.

이때 주의할 점이 하나 있습니다. 타깃값을 그대로 사이킷런 모델에 전달하면 순서가 자동으로 알파벳 순으로 매겨집니다. 따라서 pd.unique(fish['Species'])로 출력했던 순서와 다릅니다. KNeighborsClassifier에서 정렬된 타깃값은 classes_ 속성에 저장되어 있습니다.


손코딩

```
print(kn.classes_)
```



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

'Bream'이 첫 번째 클래스, 'Parkki'가 두 번째 클래스가 되는 식입니다. `predict()` 메서드는 친절하게도 타깃값으로 예측을 출력합니다. 테스트 세트에 있는 처음 5개 샘플의 타깃값을 예측해 보겠습니다.

 손코딩

```
print(kn.predict(test_scaled[:5]))
```



```
['Perch' 'Smelt' 'Pike' 'Perch' 'Perch']
```

이 5개 샘플에 대한 예측은 어떤 확률로 만들어졌을까요? 사이킷런의 분류 모델은 `predict_proba()` 메서드로 클래스별 확률값을 반환합니다. 테스트 세트에 있는 처음 5개의 샘플에 대한 확률을 출력해 보죠. 넘파이 `round()` 함수는 기본으로 소수점 첫째 자리에서 반올림을 하는데, `decimals` 매개변수로 유지할 소수점 아래 자릿수를 지정할 수 있습니다.

손코딩

```
import numpy as np
```

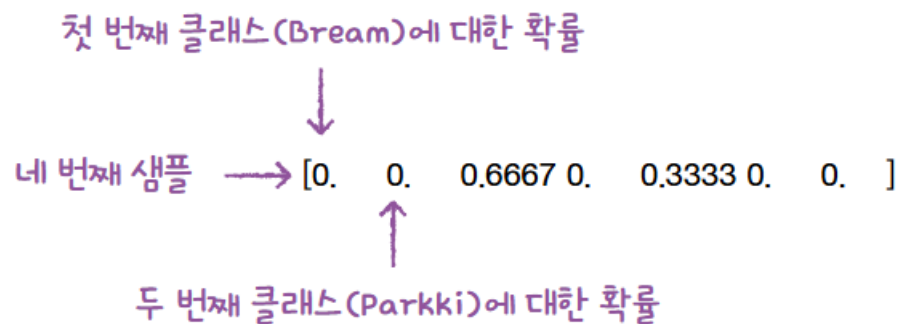
```
proba = kn.predict_proba(test_scaled[:5])
```

```
print(np.round(proba, decimals=4))
```

소수점 네 번째 자리까지 표기합니다.
다섯 번째 자리에서 반올림합니다.

```
[[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.   ]]
```

`predict_proba()` 메서드의 출력 순서는 앞서 보았던 `classes_` 속성과 같습니다. 즉 첫 번째 열이 'Bream'에 대한 확률, 두 번째 열이 'Parkki'에 대한 확률입니다.



이 모델이 계산한 확률이 가장 가까운 이웃의 비율이 맞는지 확인해 보죠. 네 번째 샘플의 최근접 이웃의 클래스를 확인해 보겠습니다.

note kneighbors() 메서드의 입력은 2차원 배열이어야 합니다. 이를 위해 넘파이 배열의 슬라이싱 연산자를 사용했습니다. 슬라이싱 연산자는 하나의 샘플만 선택해도 항상 2차원 배열이 만들어집니다. 여기에서는 네 번째 샘플 하나를 선택했습니다.

손코딩

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



```
[[ 'Roach' 'Perch' 'Perch' ]]
```

성공입니다! 아주 쉽게 클래스 확률을 예측했습니다. 번거로운 계산은 사이킷런이 수행해 주므로 우리는 `predict_proba()` 메서드를 호출하면 그만입니다.

그런데 뭔가 좀 이상하군요. 혼공머신이 잠시 생각해 보니 3개의 최근접 이웃을 사용하기 때문에 가능한 확률은 $0/3$, $1/3$, $2/3$, $3/3$ 이 전부겠군요. 만약 럭키백의 확률을 이렇게만 표시한다면 마케팅 팀이 만족하지 않을 것 같습니다. 확률이라고 말하기 좀 어색하네요. 뭔가 더 좋은 방법을 찾아야 할 것 같습니다.

로지스틱 회귀

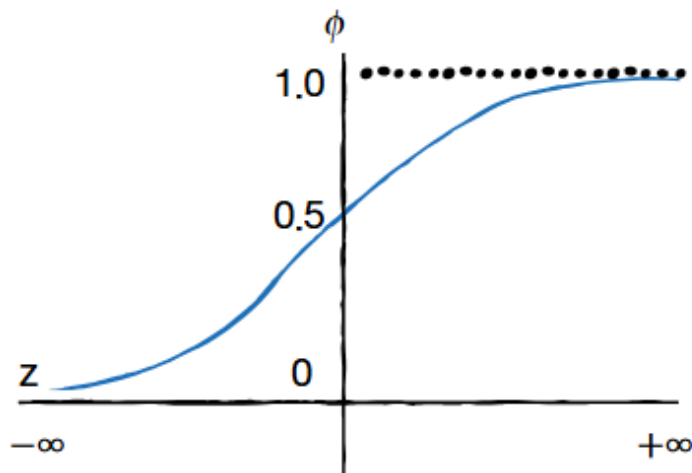
로지스틱 회귀 logistic regression는 이름은 회귀이지만 분류 모델입니다. 이 알고리즘은 선형 회귀와 동일하게 선형 방정식을 학습합니다. 예를 들면 다음과 같습니다.

$$z = a \times (\text{Weight}) + b \times (\text{Length}) + c \times (\text{Diagonal}) + d \times (\text{Height}) + e \times (\text{Width}) + f$$

여기에서 a, b, c, d, e 는 가중치 혹은 계수입니다. 특성은 늘어났지만 3장에서 다룬 다중 회귀를 위한 선형 방정식과 같습니다. z 는 어떤 값도 가능합니다. 하지만 확률이 되려면 0~1 (또는 0~100%) 사이 값이 되어야 합니다. z 가 아주 큰 음수일 때 0이 되고, z 가 아주 큰 양수일 때 1이 되도록 바꾸는 방법은 없을까요? **시그모이드 함수** sigmoid function (또는 **로지스틱 함수** logistic function)를 사용하면 가능합니다.

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

시그모이드 함수




시그모이드 그래프

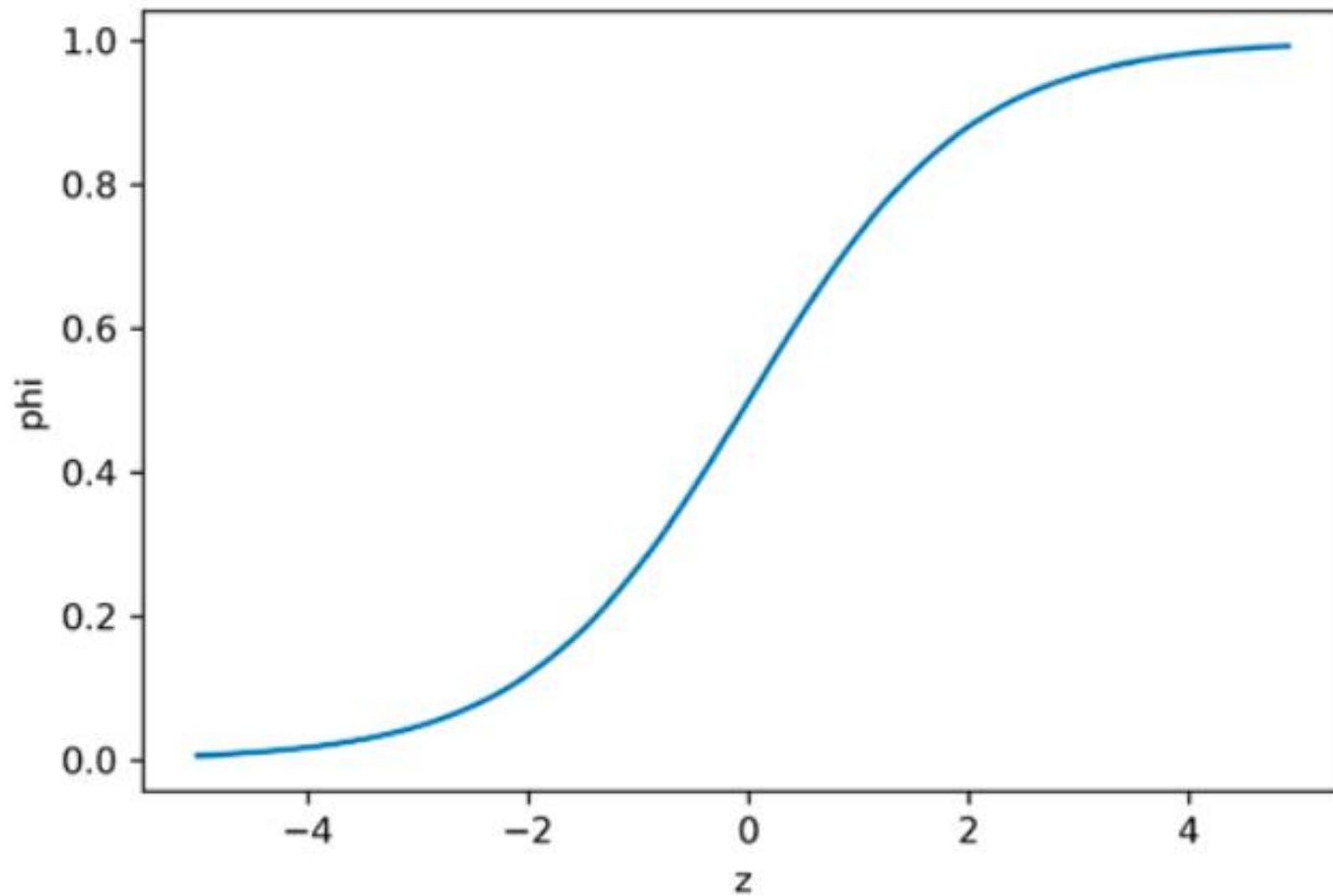
왼쪽의 식이 시그모이드 함수입니다. 선형 방정식의 출력 z 의 음수를 사용해 자연 상수 e 를 거듭제곱하고 1을 더한 값의 역수를 취합니다. 이렇게 복잡하게 계산한 이유는 오른쪽과 같은 그래프를 만들 수 있기 때문입니다.

z 가 무한하게 큰 음수일 경우 이 함수는 0에 가까워지고, z 가 무한하게 큰 양수가 될 때는 1에 가까워집니다. z 가 0이 될 때는 0.5가 되죠. z 가 어떤 값이 되더라도 ϕ 는 절대로 0~1 사이의 범위를 벗어날 수 없습니다. 그렇다면 0~1 사이 값을 0~100%까지 확률로 해석할 수 있겠군요!

넘파이를 사용하면 그래프를 간단히 그릴 수 있습니다. -5와 5 사이에 0.1 간격으로 배열 z 를 만든 다음 z 위치마다 시그모이드 함수를 계산합니다. 지수 함수 계산은 `np.exp()` 함수를 사용합니다.

손코딩

```
import numpy as np
import matplotlib.pyplot as plt
z = np.arange(-5, 5, 0.1)
phi = 1 / (1 + np.exp(-z))
plt.plot(z, phi)
plt.xlabel('z')
plt.ylabel('phi')
plt.show()
```

+ 여기서 잠깐

딱 0.5일 때는 어떻게 되나요?

정확히 0.5일 때 라이브러리마다 다를 수 있습니다. 사이킷런은 0.5일 때 음성 클래스로 판단합니다.

로지스틱 회귀로 이진 분류 수행하기


넘파이 배열은 True, False 값을 전달하여 행을 선택할 수 있습니다. 이를 **불리언 인덱싱**^{boolean indexing}이라고 합니다. 간단한 예를 보면 금방 이해할 수 있습니다. 다음과 같이 'A'에서 'E'까지 5개의 원소로 이루어진 배열이 있습니다. 여기서 'A'와 'C'만 골라내려면 첫 번째와 세 번째 원소만 True이고 나머지 원소는 모두 False인 배열을 전달하면 됩니다.

손코딩

```
char_arr = np.array(['A', 'B', 'C', 'D', 'E'])  
print(char_arr[[True, False, True, False, False]])
```

↳ ['A' 'C']

이와 같은 방식을 사용해 훈련 세트에서 도미(Bream)와 빙어(Smelt)의 행만 골라내겠습니다. 비교 연산자를 사용하면 도미와 빙어의 행을 모두 True로 만들 수 있습니다. 예를 들어 도미인 행을 골라내려면 `train_target == 'Bream'`과 같이 씁니다. 이 비교식은 `train_target` 배열에서 'Bream'인 것은 True이고 그 외는 모두 False인 배열을 반환합니다. 도미와 빙어에 대한 비교 결과를 비트 OR 연산자(!)를 사용해 합치면 도미와 빙어에 대한 행만 골라낼 수 있습니다.

 손코딩

```
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]
```

`bream_smelt_indexes` 배열은 도미와 빙어일 경우 True이고 그 외는 모두 False 값이 들어가 있습니다. 따라서 이 배열을 사용해 `train_scaled`와 `train_target` 배열에 불리언 인덱싱을 적용하면 손쉽게 도미와 빙어 데이터만 골라낼 수 있습니다.

손코딩

```
from sklearn.linear_model import LogisticRegression  
lr = LogisticRegression()  
lr.fit(train_bream_smelt, target_bream_smelt)
```

훈련한 모델을 사용해 train_bream_smelt에 있는 처음 5개 샘플을 예측해 보죠.

손코딩

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



```
['Bream' 'Smelt' 'Bream' 'Bream' 'Bream']
```

두 번째 샘플을 제외하고는 모두 도미로 예측했습니다. KNeighborsClassifier와 마찬가지로 예측 확률은 predict_proba() 메서드에서 제공합니다. train_bream_smelt에서 처음 5개 샘플의 예측 확률을 출력해 보겠습니다.

```
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]]
```

샘플마다 2개의 확률이 출력되었군요. 첫 번째 열이 음성 클래스(0)에 대한 확률이고 두 번째 열이 양성 클래스(1)에 대한 확률입니다. 그럼 Bream과 Smelt 중에 어떤 것이 양성 클래스일까요? 앞서 k-최근접 이웃 분류기에서 보았듯이 사이킷런은 타깃값을 알파벳순으로 정렬하여 사용합니다. `classes_` 속성에서 확인해 보죠.

```
print(lr.classes_)
```

```
['Bream' 'Smelt']
```

빙어(Smelt)가 양성 클래스군요. `predict_proba()` 메서드가 반환한 배열 값을 보면 두 번째 샘플만 양성 클래스인 빙어의 확률이 높습니다. 나머지는 모두 도미(Bream)로 예측하겠네요.

note 만약 도미(Bream)를 양성 클래스로 사용하려면 어떻게 해야 할까요? 2장에서 했던 것처럼 Bream인 타깃값을 1로만 두고 나머지 타깃값은 0으로 만들어 사용하면 됩니다.

로지스틱 회귀로 성공적인 이진 분류를 수행했군요! 그럼 선형 회귀에서처럼 로지스틱 회귀가 학습한 계수를 확인해 보죠.

 손코딩


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

```
[[ -0.4037798  -0.57620209 -0.66280298 -1.01290277 -0.73168947]] [-2.16155132]
```

따라서 이 로지스틱 회귀 모델이 학습한 방정식은 다음과 같습니다.

$$z = -0.404 \times (\text{Weight}) - 0.576 \times (\text{Length}) - 0.663 \times (\text{Diagonal}) - \\ 1.013 \times (\text{Height}) - 0.732 \times (\text{Width}) - 2.161$$

확실히 로지스틱 회귀는 선형 회귀와 매우 비슷하군요. 그럼 LogisticRegression 모델로 z 값을 계산해 볼 수 있을까요? 네, 가능합니다. LogisticRegression 클래스는 `decision_function()` 메서드로 z 값을 출력할 수 있습니다. `train_bream_smelt`의 처음 5개 샘플의 z 값을 출력해 보죠.

손코딩

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



```
[-6.02927744  3.57123907 -5.26568906 -4.24321775 -6.06071117 ]
```


손코딩

```
from scipy.special import expit  
print(expit(decisions))
```

```
[0.00240145 0.97264817 0.00513928 0.01415798 0.00232731]
```

출력된 값을 보면 `predict_proba()` 메서드 출력의 두 번째 열의 값과 동일합니다. 즉 `decision_function()` 메서드는 양성 클래스에 대한 z 값을 반환합니다.

좋습니다, 아주 훌륭하군요. 이진 분류를 위해 2개의 생선 샘플을 골라냈고 이를 사용해 로지스틱 회귀 모델을 훈련했습니다. 이진 분류일 경우 `predict_proba()` 메서드는 음성 클래스와 양성 클래스에 대한 확률을 출력합니다. 또 `decision_function()` 메서드는 양성 클래스에 대한 z 값을 계산합니다. 또 `coef_` 속성과 `intercept_` 속성에는 로지스틱 모델이 학습한 선형 방정식의 계수가 들어 있습니다.

이제 이진 분류의 경험을 바탕으로 7개의 생선을 분류하는 다중 분류 문제로 넘어가 보겠습니다.

로지스틱 회귀로 다중 분류 수행하기

앞에서 이진 분류를 위해 로지스틱 회귀 모델을 훈련시켜 보았습니다. 다중 분류도 크게 다르지 않습니다. 여기에서도 LogisticRegression 클래스를 사용해 7개의 생선을 분류해 보면서 이진 분류와의 차이점을 알아보겠습니다.

LogisticRegression 클래스는 기본적으로 반복적인 알고리즘을 사용합니다. max_iter 매개변수에서 반복 횟수를 지정하며 기본값은 100입니다. 여기에 준비한 데이터셋을 사용해 모델을 훈련하면 반복 횟수가 부족하다는 경고가 발생합니다. 충분하게 훈련시키기 위해 반복 횟수를 1,000으로 늘리겠습니다.

또 LogisticRegression은 기본적으로 릿지 회귀와 같이 계수의 제곱을 규제합니다. 이런 규제를 L2 규제라고도 부릅니다. 릿지 회귀에서는 alpha 매개변수로 규제의 양을 조절했습니다. alpha가 커지면 규제도 커집니다. LogisticRegression에서 규제를 제어하는 매개변수는 C입니다. 하지만 C는 alpha와 반대로 작을수록 규제가 커집니다. C의 기본값은 1입니다. 여기에서는 규제를 조금 완화하기 위해 20으로 늘리겠습니다.

손코딩

```
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
```

훈련 세트와 테스트 세트에 대한 점수가 높고 과대적합이나 과소적합으로 치우친 것 같지 않습니다. 좋네요. 그럼 테스트 세트의 처음 5개 샘플에 대한 예측을 출력해 보죠.


손코딩

```
print(lr.predict(test_scaled[:5]))
```



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

이번에는 테스트 세트의 처음 5개 샘플에 대한 예측 확률을 출력해 보겠습니다. 출력을 간소하게 하기 위해 소수점 네 번째 자리에서 반올림하겠습니다.

손코딩

```
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]]
```

와우, 많은 숫자가 출력되네요. 5개 샘플에 대한 예측이므로 5개의 행이 출력되었습니다. 또 7개 생선에 대한 확률을 계산했으므로 7개의 열이 출력되었습니다. 이진 분류일 경우 2개의 열만 있었다는 것을 기억하세요.

손코딩

```
print(lr.classes_)
```

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

네, 맞군요. 첫 번째 샘플은 Perch를 가장 높은 확률로 예측했습니다. 두 번째 샘플은 여섯 번째 열인 Smelt를 가장 높은 확률(94.6%)로 예측했습니다.

다중 분류도 어렵지 않네요. 이진 분류는 샘플마다 2개의 확률을 출력하고 다중 분류는 샘플마다 클래스 개수만큼 확률을 출력합니다. 여기에서는 7개입니다. 이 중에서 가장 높은 확률이 예측 클래스가 됩니다.

그럼 다중 분류일 경우 선형 방정식은 어떤 모습일까요? coef_와 intercept_의 크기를 출력해 보겠습니다.

손코딩

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

➦ (7, 5) (7,)

이 데이터는 5개의 특성을 사용하므로 coef_ 배열의 열은 5개입니다. 그런데 행이 7이군요. intercept_도 7개나 있습니다. 이 말은 이진 분류에서 보았던 z 를 7개나 계산한다는 의미입니다. 혹시 눈치채셨나요? 네, 맞습니다. 다중 분류는 클래스마다 z 값을 하나씩 계산합니다. 당연히 가장 높은 z 값을 출력하는 클래스가 예측 클래스가 됩니다. 그럼 확률은 어떻게 계산한 것일까요? 이진 분류에서는 시그모이드 함수를 사용해 z 를 0과 1 사이의 값으로 변환했습니다. 다중 분류는 이와 달리 **소프트맥스** softmax 함수를 사용하여 7개의 z 값을 확률로 변환합니다.

+ 여기서 잠깐

소프트맥스 함수가 뭔가요?

시그모이드 함수는 하나의 선형 방정식의 출력값을 0~1 사이로 압축합니다. 이와 달리 소프트맥스 함수는 여러 개의 선형 방정식의 출력값을 0~1 사이로 압축하고 전체 합이 1이 되도록 만듭니다. 이를 위해 지수 함수를 사용하기 때문에 **정규화된 지수 함수**라고도 부릅니다.

소프트맥스도 어렵지 않습니다. 차근차근 계산 방식을 짚어 보겠습니다. 먼저 7개의 z 값의 이름을 $z1$ 에서 $z7$ 이라고 붙이겠습니다. $z1 \sim z7$ 까지 값을 사용해 지수 함수 $e^{z1} \sim e^{z7}$ 을 계산해 모두 더합니다. 이를 e_sum 이라고 하겠습니다.

$$e_sum = e^{z1} + e^{z2} + e^{z3} + e^{z4} + e^{z5} + e^{z6} + e^{z7}$$

그다음 $e^{z1} \sim e^{z7}$ 을 각각 e_sum 으로 나누어 주면 됩니다.

$$s1 = \frac{e^{z1}}{e_sum}, s2 = \frac{e^{z2}}{e_sum}, \dots, s7 = \frac{e^{z7}}{e_sum}$$

$s1$ 에서 $s7$ 까지 모두 더하면 분자와 분모가 같아지므로 1이 됩니다. 7개 생선에 대한 확률의 합은 1이 되어야 하므로 잘 맞네요.

✚ 여기서 잠깐

시그모이드 함수와 소프트맥스 함수가 중요한가요?

시그모이드 함수와 소프트맥스 함수를 왜 이렇게 자세히 공부하는지 궁금할 수 있습니다. 사이킷런에서 자동으로 계산해 주지만 이 두 함수는 나중에 신경망을 배울 때 또다시 등장합니다. 여기에서 자세히 익혀두면 나중에 신경망을 배울 때 훨씬 잘 이해할 수 있습니다.

그럼 이전 분류에서처럼 `decision_function()` 메서드로 $z1 \sim z7$ 까지의 값을 구한 다음 소프트맥스 함수를 사용해 확률로 바꾸어 보겠습니다. 먼저 테스트 세트의 처음 5개 샘플에 대한 $z1 \sim z7$ 의 값을 구해 보죠.

손코딩

```
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]]
```


역시 사이파이는 소프트맥스 함수도 제공합니다. `scipy.special` 아래에 `softmax()` 함수를 임포트해 사용하겠습니다.

 손코딩

```
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]]
```

앞서 구한 `decision` 배열을 `softmax()` 함수에 전달했습니다. `softmax()`의 `axis` 매개변수는 소프트맥스를 계산할 축을 지정합니다. 여기에서는 `axis=1`로 지정하여 각 행, 즉 각 샘플에 대해 소프트맥스를 계산합니다. 만약 `axis` 매개변수를 지정하지 않으면 배열 전체에 대해 소프트맥스를 계산합니다.

로지스틱 회귀로 확률 예측

문제해결 과정

김 팀장은 럭키백에 담긴 생선이 어떤 생선인지 확률을 예측해달라고 혼공머신에게 요청했습니다. 분류 모델은 예측뿐만 아니라 예측의 근거가 되는 확률을 출력할 수 있습니다. 어찌 보면 이 확률은 분류 모델이 얼마나 예측을 확신하는지 나타냅니다. 확률이 높을수록 강하게 예측하는 셈이죠.

k-최근접 이웃 모델이 확률을 출력할 수 있지만 이웃한 샘플의 클래스 비율이므로 항상 정해진 확률만 출력합니다. 이는 마케팅 팀의 요구사항을 만족시킬 수 없을 것 같습니다. 고객이 어느 정도 생선을 예상할 수 있지만 상품마다 좀 더 그럴싸한 확률을 표시했으면 합니다.

이를 위해 가장 대표적인 분류 알고리즘 중 하나인 로지스틱 회귀를 사용했습니다. 로지스틱 회귀는 회귀 모델이 아닌 분류 모델입니다. 선형 회귀처럼 선형 방정식을 사용합니다. 하지만 선형 회귀처럼

계산한 값을 그대로 출력하는 것이 아니라 로지스틱 회귀는 이 값을 0~1 사이로 압축합니다. 우리는 이 값을 마치 0~100% 사이의 확률로 이해할 수 있습니다.

로지스틱 회귀는 이진 분류에서는 하나의 선형 방정식을 훈련합니다. 이 방정식의 출력값을 시그모이드 함수에 통과시켜 0~1 사이의 값을 만듭니다. 이 값이 양성 클래스에 대한 확률입니다. 음성 클래스의 확률은 1에서 양성 클래스의 확률을 빼면 됩니다.

다중 분류일 경우에는 클래스 개수만큼 방정식을 훈련합니다. 그다음 각 방정식의 출력값을 소프트맥스 함수를 통과시켜 전체 클래스에 대한 합이 항상 1이 되도록 만듭니다. 이 값을 각 클래스에 대한 확률로 이해할 수 있습니다.

다음 절에서는 인기가 높고 성능이 뛰어난 또 다른 머신러닝 알고리즘인 확률적 경사 하강법에 대해 배워 보겠습니다.