

# 선형 회귀

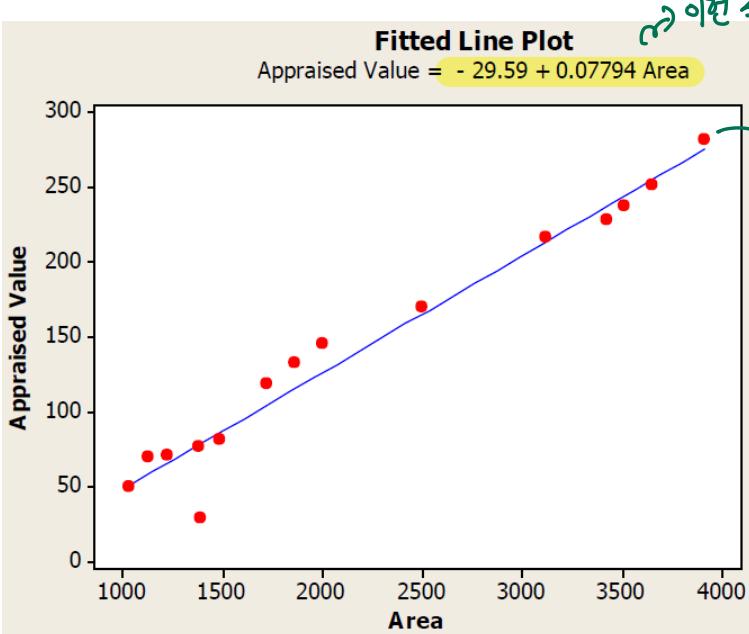
---

Multiple Linear Regression

# 다중선형회귀(Multiple Linear Regression)

## ■ 다중선형회귀란?

- 수치형 설명변수  $X$  와 연속형 숫자로 이루어진 종속변수  $Y$  간의 관계를 선형으로 가정하고 이를 가장 잘 표현할 수 있는 회귀계수를 데이터로부터 추정하는 모델
- (예시) 1개의 설명변수(주택 크기: $x_1$ )와 연속형 종속변수(주택 가격: $y$ )의 관계를 나타내는 직선을 찾는 문제 + 주택의 위치, 인프라 등등 많이 있을 것이다.



예) 차전거 X

$X_1$  바퀴의 수

$X_2$  바퀴의 사이즈

$X_3$  안장의 높이



이것 범주형이 X

연속형이다.

OK!

\* 차전거 A

→ 평속 : 17km/h

\* 차전거 B

17.5 km/h

방법론  
실데이터

$$\leftarrow y = \beta_0 + \beta_1 x_1$$

설명변수가 1개인 경우

# 다중선형회귀(Multiple Linear Regression)

라벨  
 $\hat{y} = X \cdot \vec{\beta}$   
회귀 계수  
설명

## ■ 다중선형회귀 모델 방정식

- N개의 데이터, K개의 설명변수( $x_{11}, \dots, x_{nk}$ ), K개의 회귀계수( $\beta_0, \dots, \beta_k$ )

★  
다중선형회귀  
방정식 이렇게  
설명으로 표현할 수  
있다.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1k} \\ 1 & x_{21} & \dots & x_{2k} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n1} & \dots & x_{nk} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

$\hat{y} = X \vec{\beta} + \vec{\varepsilon}$ ,  $\vec{\varepsilon} \sim N(E(\vec{\varepsilon}), V(\vec{\varepsilon}))$

$$E(\vec{\varepsilon}) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad V(\vec{\varepsilon}) = \sigma^2 I$$

$$y_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12}$$

$$y_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22}$$

:

N개

# 다중선형회귀(Multiple Linear Regression)

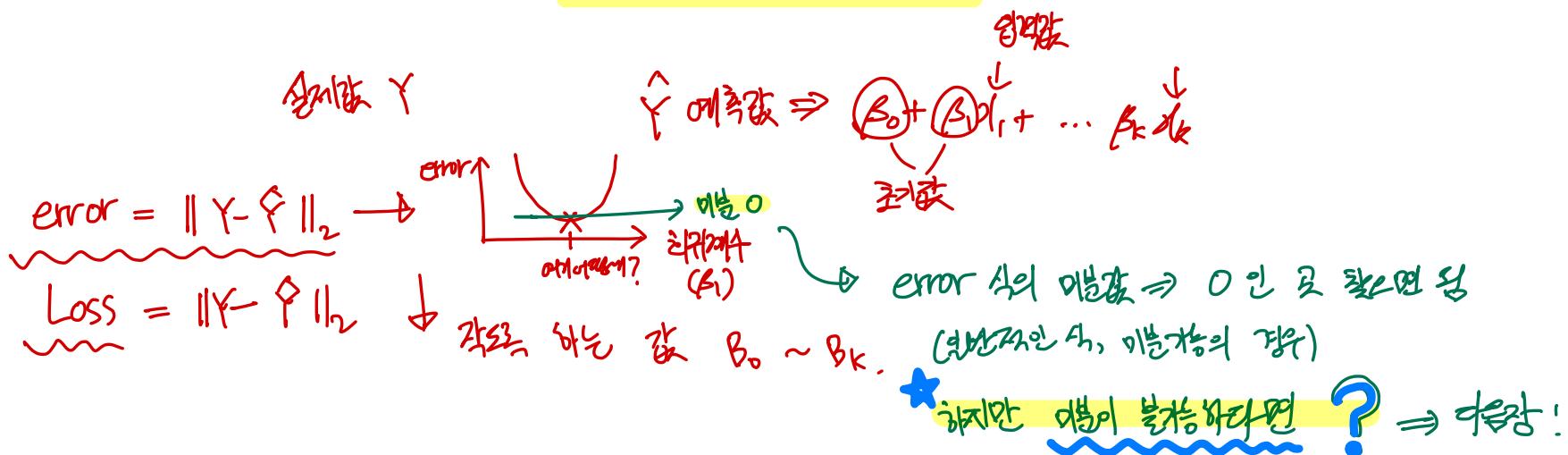
그러면 회귀계수  $\beta$  어떻게 구할 수 있을까?

$(\beta_0, \beta_1, \beta_2, \dots, \beta_k)$

## 회귀 계수 결정법 : Direct Solution

- 선형회귀의 계수들은 실제값(Y)과 모델 예측값( $\hat{Y}$ )의 차이, 오차제곱합(error sum of squares)을 최소로 하는 값을 회귀 계수로 선정
- 최적의 계수들은 회귀 계수에 대해 미분한 식을 0으로 놓고 풀면 명시적인 해를 구할 수 있음. 즉, X와 Y데이터만으로 회귀 계수를 구할 수 있음

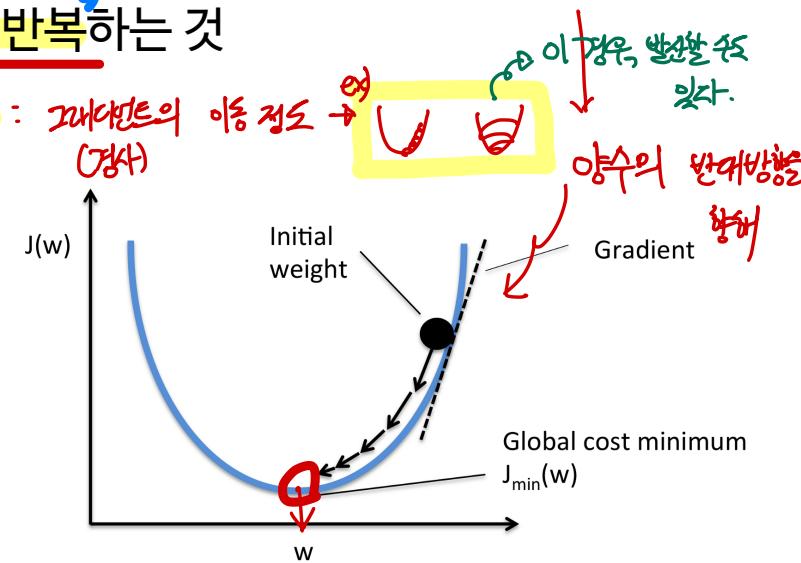
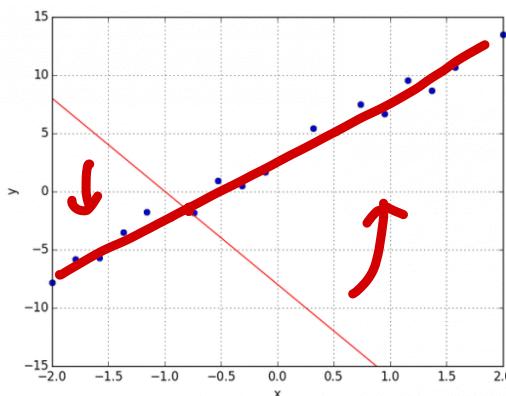
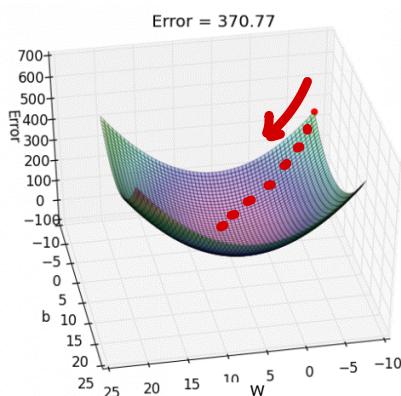
$$\vec{\beta} = (X^T X)^{-1} X^T \vec{y}$$



# 다중선형회귀(Multiple Linear Regression)

## ▪ 회귀 계수 결정법: Numerical Search

- 경사하강법(gradient descent) 같은 반복적인 방식으로 선형회귀 계수를 구할 수 있음
- 경사하강법이란 어떤 함수 값(목적 함수, 비용 함수, 에러 값)을 최소화하기 위해 임의의 시작점을 잡은 후 해당 지점에서의 그래디언트(경사)를 구하고, 그래디언트의 반대 방향으로 조금씩 이동하는 과정을 여러번 반복하는 것



☆ 최소화 하자 하는 것

☆ Learning rate : 그래디언트의 이동 정도 → 경사

이 경우, 빨리 할 수 있다.

양수의 반대방향을 향해

Initial weight

Global cost minimum  
 $J_{\min}(w)$

Gradient

# 다중선형회귀(Multiple Linear Regression)

## Numerical Search

### 경사하강법의 종류 ~~~ 매우 다양하자.

#### Batch Gradient Descent (GD)

$$y = f_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

- 파라미터를 업데이트 할 때마다 모든 학습 데이터를 사용하여 cost function의 gradient를 계산

Vanilla Gradient Descent 라 불림

매우 낮은 학습 효율을 보일 수 있음

N 개

학습 효율이 ↓

10만, 100만 ...

비대소  
Large Scale ...

#### Stochastic Gradient Descent (SGD)

- 파라미터를 업데이트 할 때, 무작위로 샘플링된 학습 데이터를 하나씩만 이용하여 cost function의 gradient를 계산

온전히 선정 N

모델을 자주 업데이트 하며, 성능 개선 정도를 빠르게 확인 가능

Local minima에 빠질 가능성을 줄일 수 있음

단점(최소 cost에 수렴했는지의 판단이 상대적으로 어려움) ~~~ '수행 봤다'라고 결론 내리기가 어려운 것!

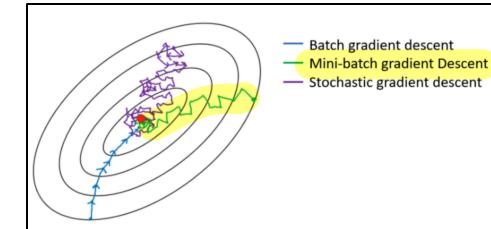
#### Mini Batch Gradient Descent ① 단점 보완

- 파라미터를 업데이트 할 때마다 일정량의 일부 데이터를 무작위로 뽑아 cost function의 gradient를 계산

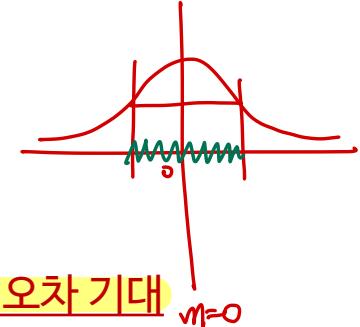
Batch Gradient Descent와 Stochastic Gradient Descent 개념의 혼합

SGD의 노이즈를 줄이면서, GD의 전체 배치보다 효율적

널리 사용되는 기법

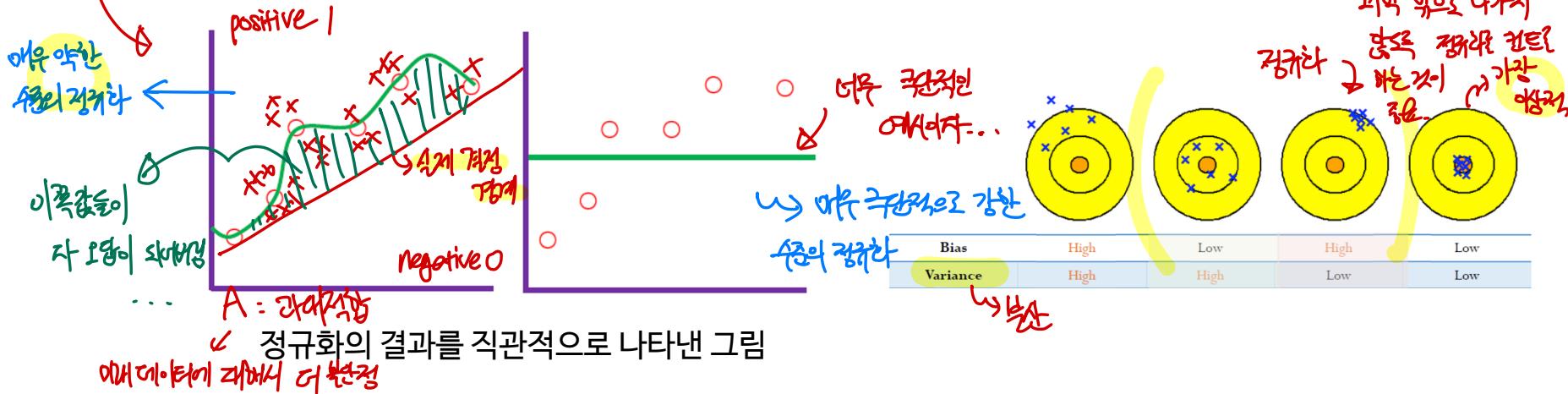


# 정규화



## ▪ 정규화(regularization)란?

- 회귀계수가 가질 수 있는 값에 제약조건을 부여하여 미래 데이터에 대한 오차 기대  $m=0$
- 미래데이터에 대한 오차의 기대 값은 모델의 Bias와 variance로 분해 가능.
- 정규화는 variance를 감소시켜 일반화 성능을 높이는 기법  $\rightarrow$  보간
  - 단, 이 과정에서 bias가 증가할 수 있음
- 왼쪽 그림은 학습데이터를 정말 잘 맞추고 있지만, 미래 데이터가 조금만 바뀌어도 예측 값이 들쭉날쭉할 수 있음
- 우측 그림은 가장 강한 수준의 정규화를 수행한 결과로 학습데이터에 대한 설명력을 다소 포기하는 대신 미래 데이터 변화에 상대적으로 안정적인 결과를 나타냄



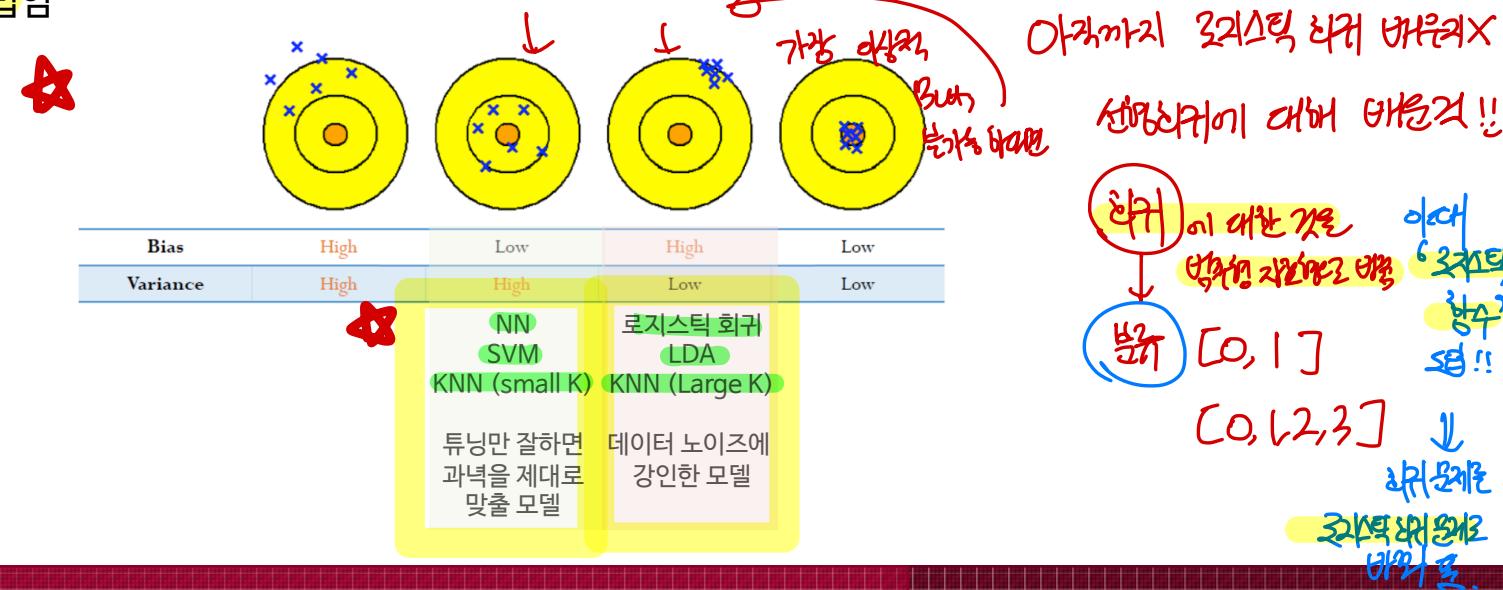
# Bias-Variance Decomposition

## ▪ Bias-Variance Decomposition란?

- 일반화(generalization) 성능을 높이는 정규화(Regularization), 앙상블(ensemble) 기법의 이론적 배경
- 학습에 쓰지 않은 미래데이터에 대한 오차의 기대값을 모델의 Bias와 Variance로 분해하자는 내용

## ▪ Bias-Variance의 직관적인 이해

- 첫번째 그림을 보면 예측값(파란색 엑스표)의 평균이 과녁(Truth)과 멀리 떨어져 있어 Bias가 크고, 예측값들이 서로 멀리 떨어져 있어 Variance 또한 큼
- 네번째 그림의 경우 Bias, Variance 모두 작음. 제일 이상적임
- 부스팅(Boosting)은 Bias를 줄여 성능을 높이고, 라쏘회귀(Lasso regression)는 Variance를 줄여 성능을 높이는 기법임



# 로지스틱 회귀

---

Logistic regression

# 다중선형회귀(Multiple Linear Regression)

- **다중선형회귀(Multiple Linear Regression)란?**
  - **수치형 설명변수(X) 와 연속형 숫자로 이뤄진 종속변수(Y) 간의 관계를 선형으로 가정하고 이를 가장 잘 표현할 수 있는 회귀 계수( $\beta$ )를 데이터로부터 추정하는 것**
  - 다중선형회귀모델 (오차항:  $\epsilon$ , 회귀계수:  $\beta_p$ )

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \epsilon$$

$$Y = \vec{\beta} \cdot X + \epsilon$$

- **회귀 계수 결정법**
  - 회귀 계수들은 모델의 예측 값( $\bar{Y}$ )과 실제 값( $Y$ )의 차이, 즉 **오차 제곱 합(error sum of squares)**을 최소로 하는 값으로 결정 가능함  
~~> 가능한한 경우
  - 즉, 회귀 계수에 대해 미분한 식을 0으로 놓고 풀어 명시적인 해를 구할 수 있음

- GD
- SGD
- Mini Batch GD

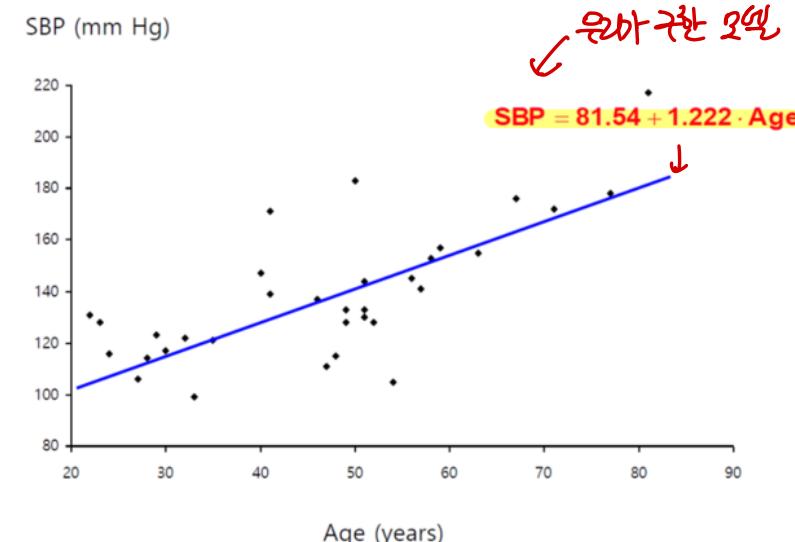
# 다중선형회귀(Multiple Linear Regression)

## [다중선형회귀 예시 #1]

- 33명의 성인 여성에 대한 나이와 혈압 데이터
- 오차제곱합을 최소로 하는 회귀 계수 계산 결과 및 분석
  - $SBP = 81.54 + 1.222AGE$
  - 나이라는 변수에 대응하는 계수는 1.222로 나타났는데, [이는 나이를 한 살 더 먹으면 혈압이 1.222mm/Hg만큼 증가한다는 결과를 보여줌]

선형 회귀에서 블록고 영어가는 방향선 →

Age	SBP	Age	SBP	Age	SBP
22	131	41	139	52	128
23	128	41	171	54	105
24	116	46	137	56	145
27	106	47	111	57	141
28	114	48	115	58	153
29	123	49	133	59	157
30	117	49	128	63	155
32	122	50	183	67	176
33	99	51	130	71	172
35	121	51	133	77	178
40	147	51	144	81	217



# 다중선형회귀(Multiple Linear Regression)

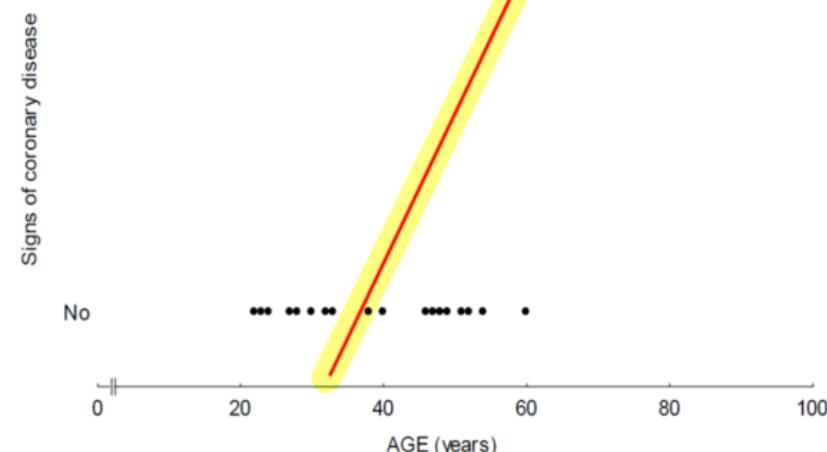
## [다중선형회귀 예시 #2]

- 33명의 성인 여성에 대한 나이(Age)와 암 발병(CD) 데이터
- 오차제곱합을 최소로 하는 회귀 계수 계산 결과 및 분석
- 다중선형회귀 모델 적용 불가
  - 범주형 숫자(암 발병 여부)는 연속형 숫자(혈압)와 달리 의미를 지니지 않음
  - 즉, 0(정상)과 1(발병)을 서로 바꾸어도 상관 없음

→ 범주형 숫자는 로지스틱 회귀 모델 적용 가능

★point!  
선형 회귀  
분  
Linear regression 으로 풀 수

Age	CD	Age	CD	Age	CD
22	0	40	0	54	0
23	0	41	1	55	1
24	0	46	0	58	1
27	0	47	0	60	1
28	0	48	0	60	0
30	0	49	1	62	1
30	0	49	0	65	1
32	0	50	1	67	1
33	0	51	0	71	1
35	1	51	1	77	1
38	0	52	0	81	1



# 로지스틱 함수 (Logistic function)

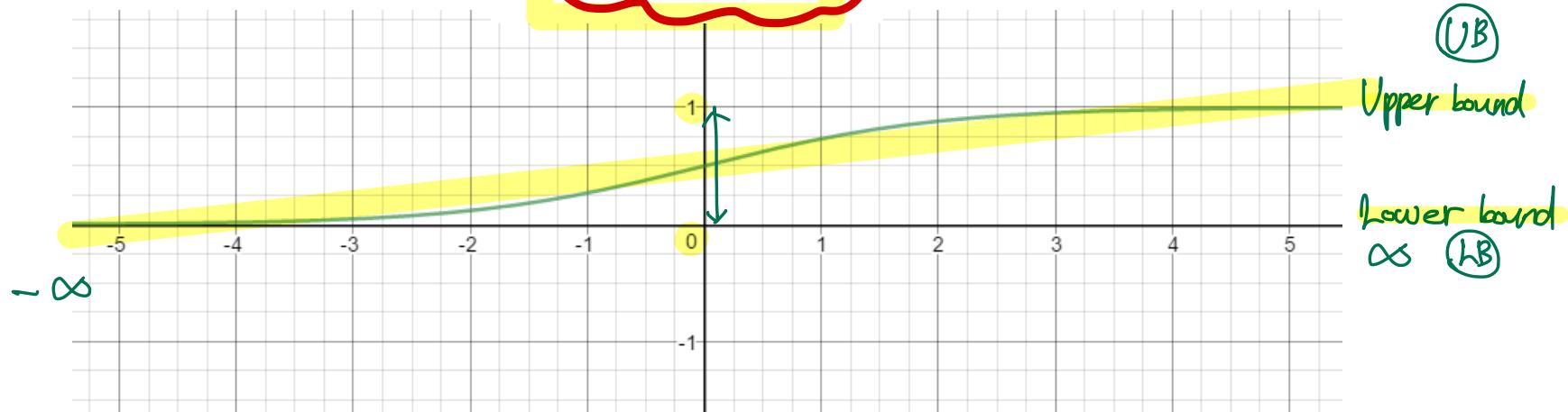
## ■ 로지스틱 함수란?

- 아래 그림과 같이 S-커브 함수를 나타냄
  - 실제 많은 자연, 사회현상에서는 특정 변수에 대한 확률 값이 선형이 아닌 S-커브 형태를 따르는 경우가 많음
- x값으로 어떤 값이든 받을 수가 있지만 출력 결과(y)는 항상 0에서 1사이 값이 됨
  - 확률밀도함수(probability density function) 요건을 충족
- 시그모이드 함수라고 명명하기도 함

로지스틱 함수

로지스틱 함수

$$y = \frac{1}{1 + e^{-x}}$$



# 승산 (Odds)

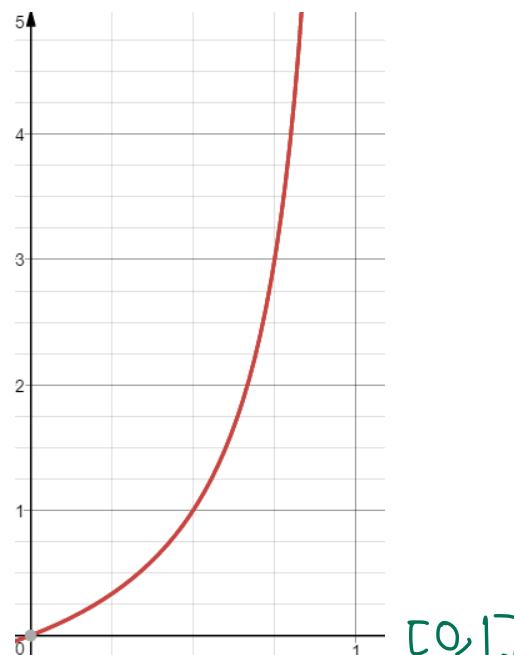
- 승산 (Odds)이란?

- 임의의 사건 A가 발생하지 않을 확률 대비 일어날 확률의 비율

$$odds = \frac{P(A)}{P(A^c)} = \frac{P(A)}{1 - P(A)}$$

- $P(A)$ 가 1에 가까울 수록 승산은 커지고, 반대로  $P(A)$ 가 0이라면 승산은 0

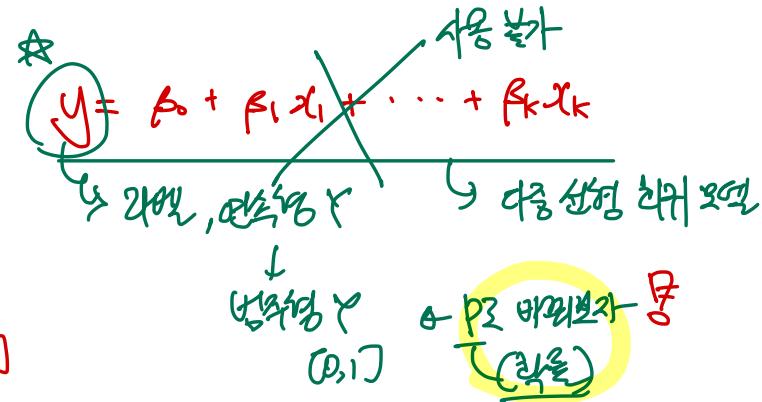
★  $[0, \infty]$



# 이항 로지스틱 회귀

범주가 2개인 경우

- Y가 범주형 일 경우, 회귀 모델을 적용할 수 없음,
- Y를 확률식으로 바꿔보면



$[0,1]$

$[-\infty, \infty]$

$$P(Y = 1 | X = \vec{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p \\ = \vec{\beta}^T \vec{x}$$

- Y를 승산으로 바꿔보면

$[0, \infty]$

$[-\infty, \infty]$

$$\frac{P(Y = 1 | X = \vec{x})}{1 - P(Y = 1 | X = \vec{x})} = \vec{\beta}^T \vec{x}$$

이렇게 범위를 맞춰준 것.

- Y 승산에 로그를 취하면

$[-\infty, \infty]$

$[-\infty, \infty]$

$$\log \left( \frac{P(Y = 1 | X = \vec{x})}{1 - P(Y = 1 | X = \vec{x})} \right) = \vec{\beta}^T \vec{x}$$

# 이항 로지스틱 회귀

- $x$ 가 주어졌을 때 범주1일 확률을  $p(x)$ , 위 식 우변을  $a$ 로 치환해 정리하면

$$\log_e \left( \frac{p(x)}{1-p(x)} \right) \Rightarrow \frac{p(x)}{1-p(x)} = e^a$$

$$\begin{aligned} &= \vec{\beta}^T \vec{x} \\ &\quad p(x) = e^a \{1 - p(x)\} \\ &\quad = e^a - e^a p(x) \end{aligned}$$

$$p(x)(1 + e^a) = e^a$$

$$\underline{p(x)} = \frac{e^a}{1 + e^a} = \frac{1}{1 + e^{-a}}$$

$$\therefore P(Y=1|X=\vec{x}) = \frac{1}{1 + e^{-\vec{\beta}^T \vec{x}}} \quad \rightarrow \text{로지스틱 함수}$$

# 이항 로지스틱 회귀의 결정 경계

- 이항 로지스틱 모델은 범주 정보를 모르는 입력 벡터  $x$ 를 넣으면 범주 1에 속할 확률을 반환하며, 범주 1로 분류하는 판단 기준은 아래와 같음

$$P(Y = 1|X = \vec{x}) > P(Y = 0|X = \vec{x})$$

- 범주가 두 개 뿐이므로, 위 식 좌변을  $p(x)$ 로 치환하면,

$$p(x) > 1 - p(x)$$

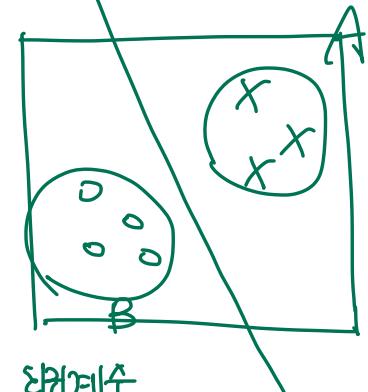
$$\frac{p(x)}{1 - p(x)} > 1$$

$$\log \frac{p(x)}{1 - p(x)} > 0$$

$$\therefore \vec{\beta}^T \vec{x} > 0 \quad \textcircled{1}$$

집값(학점)

$$\vec{\beta}$$



학점계수

$$> 0$$

$$\Rightarrow \textcircled{1}$$

Given

- 마찬가지로  $\vec{\beta}^T \vec{x} < 0$  이면 데이터 범주를 0으로 분류하게 되며, 로지스틱 결정 경계 (decision boundary)는  $\vec{\beta}^T \vec{x} = 0$  인 하이퍼플레인 (hyperplane)입니다.

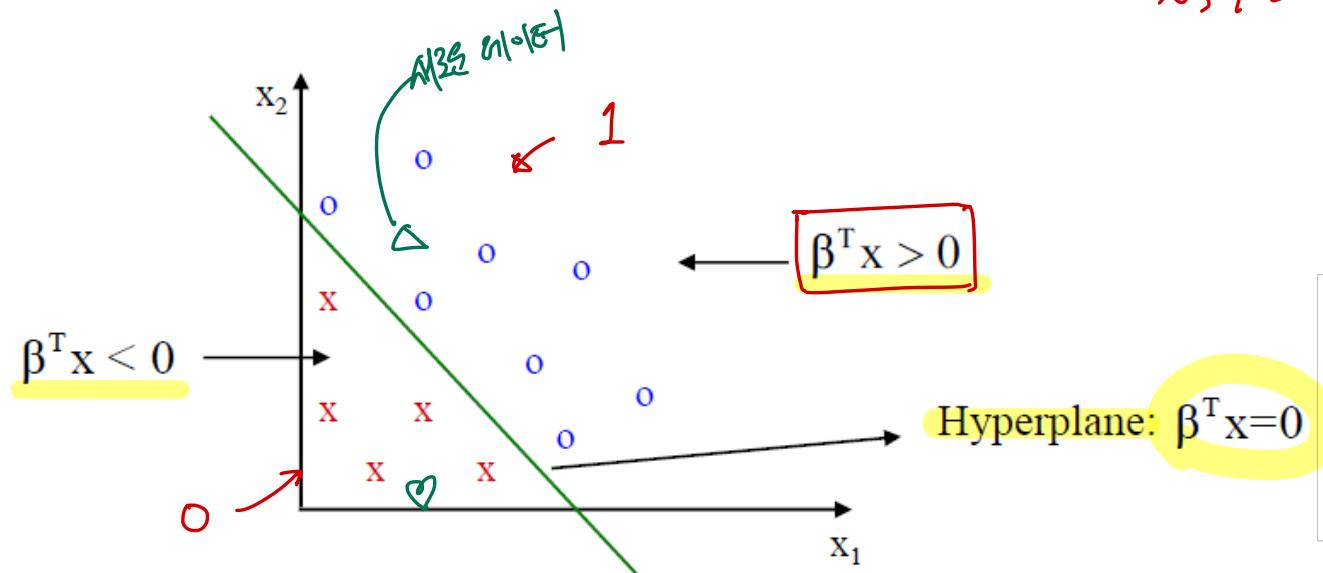
데이터의 학점계수의 공안 보고

국제사를 분류할 수 있는 것

# 이항 로지스틱 회귀의 결정 경계

로지스틱 회귀

↳ 확률,  $\beta$ 를 구하는 상황에서.



## Classifier

$$y = \frac{1}{(1 + \exp(-\beta^T x))}$$

$$\begin{cases} y \rightarrow 1 & \text{if } \beta^T x \rightarrow \infty \\ y = \frac{1}{2} & \text{if } \beta^T x = 0 \\ y \rightarrow 0 & \text{if } \beta^T x \rightarrow -\infty \end{cases}$$

# 다항 로지스틱 회귀

- 이항 로지스틱 회귀 모델을 통한 다항 로지스틱 회귀 문제 풀기

$$\log \frac{P(Y=1|X=\vec{x})}{P(Y=3|X=\vec{x})} = \beta_1^T \vec{x}$$

$$\log \frac{P(Y=2|X=\vec{x})}{P(Y=3|X=\vec{x})} = \beta_2^T \vec{x}$$

↑ 이와 같이 훈련할  
수 있게 된다.

- 세 번째 범주에 속할 확률 = 1 - 첫 번째 범주에 속할 확률 - 두 번째 범주에 속할 확률

$$P(Y=1|X=\vec{x}) = \frac{e^{\beta_1^T \vec{x}}}{1 + e^{\beta_1^T \vec{x}} + e^{\beta_2^T \vec{x}}}$$

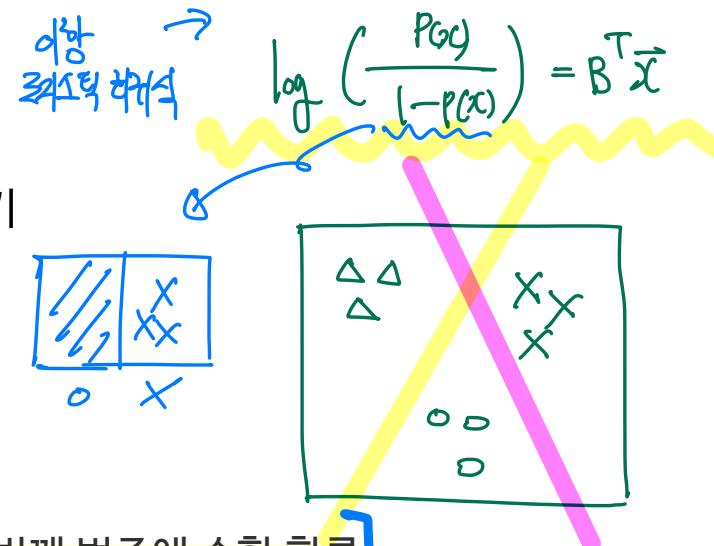
$$P(Y=2|X=\vec{x}) = \frac{e^{\beta_2^T \vec{x}}}{1 + e^{\beta_1^T \vec{x}} + e^{\beta_2^T \vec{x}}}$$

$$P(Y=3|X=\vec{x}) = \frac{1}{1 + e^{\beta_1^T \vec{x}} + e^{\beta_2^T \vec{x}}}$$

- K개 범주를 분류하는 다항로지스틱 회귀 모델의 입력 벡터  $x$ 가 각 클래스로 분류될 확률

$$P(Y=k|X=\vec{x}) = \frac{e^{\beta_k^T \vec{x}}}{1 + \sum_{i=1}^{K-1} e^{\beta_i^T \vec{x}}} \quad (k=0, 1, \dots, K-1)$$

$$P(Y=K|X=\vec{x}) = \frac{1}{1 + \sum_{i=1}^{K-1} e^{\beta_i^T \vec{x}}}$$



# 다항 로지스틱 회귀와 소프트맥스

- ‘로그승산’으로 된 좌변을 ‘로그확률’로 변경

$$\log P(Y = 1|X = \vec{x}) = \log \frac{e^{\beta_1^T \vec{x}}}{1 + e^{\beta_1^T \vec{x}} + e^{\beta_2^T \vec{x}}}$$

$$P(Y = 2|X = \vec{x}) = \frac{e^{\beta_2^T \vec{x}}}{1 + e^{\beta_1^T \vec{x}} + e^{\beta_2^T \vec{x}}}$$

$$P(Y = 3|X = \vec{x}) = \frac{1}{1 + e^{\beta_1^T \vec{x}} + e^{\beta_2^T \vec{x}}}$$

$$\log P(Y = 1|X = \vec{x}) = \beta_1^T \vec{x} - \log Z$$

$$\log P(Y = 2|X = \vec{x}) = \beta_2^T \vec{x} - \log Z$$

...

$$\log P(Y = K|X = \vec{x}) = \beta_K^T \vec{x} - \log Z$$

- 로그 성질을 활용해 c번째 범주에 속할 확률을 기준으로 식을 정리

$$\log P(Y = c) + \log Z = \beta_c^T \vec{x}$$

$$\log \{P(Y = c) \times Z\} = \beta_c^T \vec{x}$$

$$P(Y = c) \times Z = e^{\beta_c^T \vec{x}}$$

$$P(Y = c) = \frac{1}{Z} e^{\beta_c^T \vec{x}}$$

$$P(Y = c) = \frac{e^{\beta_c^T \vec{x}}}{\sum_{k=1}^K e^{\beta_k^T \vec{x}}}$$

소프트맥스 !!

- 전체 확률 합은 1

$$1 = \sum_{k=1}^K P(Y = k) = \sum_{k=1}^K \frac{1}{Z} e^{\beta_k^T \vec{x}} = \frac{1}{Z} \sum_{k=1}^K e^{\beta_k^T \vec{x}}$$

$$\therefore Z = \sum_{k=1}^K e^{\beta_k^T \vec{x}}$$

## ★ 앞 내용 정리 ★

↳ ① 자동 선별 허가로 풀 수 없는 법주행 문제를  
로지스틱 함수를 이용해 풀 수 있게 되었다.

② 이방로지스틱 회귀 / 다항로지스틱 회귀

↓  
시그모이드 함수  
(로지스틱 함수)  
↓  
소프트맥스 함수.

## 로지스틱 회귀 적용

↳ <sup>6↑</sup> <sub>7↑</sub> 분류 문제 풀어보자!

# 로지스틱 회귀 실험

- 실습 코드: [링크](#)
- 데이터셋: Iris 데이터
- 학습/시험 데이터: x\_train/x\_test
- 학습/시험 데이터 라벨: y\_train/y\_test → (0,1,2)  
*String* ↴
- 데이터 로더, 범주형 데이터 변환, 데이터 분할
  - KNN 실습과 동일
- 로지스틱 회귀 API
  - LogisticRegression 클래스 호출  $C=1/\lambda$ 로 규제화의 정도를 조절
    - C값이 클수록 규제화의 강도가 줄어듬

```
[5] 1 # Logistic regression
2 from sklearn.linear_model import LogisticRegression
3 Logit = LogisticRegression(C=200, random_state=11) # C = 1/λ. 디폴트: L2, Auto.
4 l_1=Logit.fit(X_train_std, y_train)
5 y_train_pred = Logit.predict(X_train_std)
6 y_test_pred = Logit.predict(X_test_std)
7
```

↳ 다른 파라미터는 default

# 로지스틱 회귀 실험

## ▪ 로지스틱 회귀 평가

- 학습데이터 정밀도(accuracy) / 시험데이터 정밀도 / 정확도 행렬(confusion matrix)

```
[6] 1 # Accuracy score
2 from sklearn.metrics import accuracy_score
3 print(accuracy_score(y_train,y_train_pred)) ... 학습
4 print(accuracy_score(y_test,y_test_pred)) ... 테스트
5
[] 0.9809523809523809
1.0
```

수정한 부분: 정밀도 행렬을 계산하는 코드를 제거하고 정밀도만 계산하는 코드로 수정하였습니다.

설명: 정밀도 행렬을 계산하는 코드는 제거되었지만, 정밀도(accuracy)를 계산하는 코드는 남아 있습니다. 정밀도는 학습 데이터와 테스트 데이터에 대한 정밀도를 각각 출력합니다. 정밀도는 0.9809523809523809로 표시됩니다.

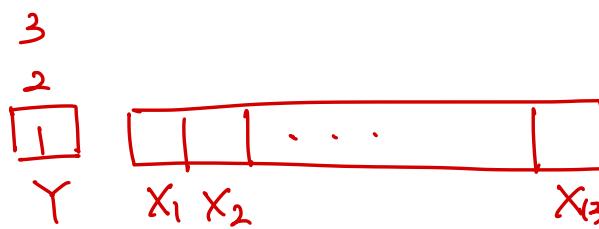
```
▶ 1 # Confusion matrix
2 from sklearn.metrics import confusion_matrix
3 print(confusion_matrix(y_test, y_test_pred)) # Confusion matrix
[] 1 [[15  0  0]
2 [ 0 15  0]
3 [ 0  0 15]]
```

수정한 부분: 정밀도 행렬을 계산하는 코드를 제거하고 정밀도 행렬을 계산하는 코드로 수정하였습니다.

설명: 정밀도 행렬을 계산하는 코드는 제거되었지만, 정밀도 행렬을 계산하는 코드는 남아 있습니다. 정밀도 행렬은 [[15, 0, 0], [0, 15, 0], [0, 0, 15]]로 표시됩니다. 행은 예측값, 열은 실제값입니다.

# 로지스틱 회귀 실험

- 실습코드: 링크
- 데이터셋: 와인 데이터
- 학습/시험 데이터:  $x_{train}/x_{test}$  (13개의 특성변수)
- 학습/시험 데이터 라벨:  $y_{train}/y_{test} \rightarrow (1, 2, 3)$ 로 이미 범주형으로 저장되어 있음



별도로 변환 X

- 다양한 규제강도에 따른 실험
  - $L_1, L_2$  규제화
  - $C=1/\lambda$ 로 규제화
  - 테스트 데이터의 라벨을 알 수 없을 경우, 학습데이터의 일부를 검증데이터(validation data)로 구성하여 테스트

→ '학습계수'가 어떻게 바뀌나?!
- 다양한 규제강도에 따른 초정계수 실험
  - 규제강도가 클수록 추정된 계수들의 절대값이 작아짐
  - $L_1$  규제화의 경우 규제강도가 클수록 계수에 0이 많아짐. 즉, 계수에 대응하는 특성변수를 제거하는 역할을 담당함

$L_2$ 의  $\beta$ 값은 거의 변화 X

$L_1$ 의  $\beta$ 값은 규제가 커지면 ~ 0 이 된다.

# 로지스틱 회귀 실험

## ■ 데이터 로더

```
[2] 1 # 데이터 불러오기. y값은 이미 범주형으로 되어있음.  
2 dat_wine=pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/'  
3                      'wine/wine.data',header=None)  
4 dat_wine.head()  
5 dat_wine.columns = ['class label', 'alchohol', 'malic acid', 'ash',  
6                      'alcalinity of ash', 'magnesium', 'total phenols',  
7                      'flavanoids', 'nonflavanoid phenols',  
8                      'proanthocyanins', 'color intensity', 'hue',  
9                      'OD208', 'proline'] # Column names  
10 print('class label:', np.unique(dat_wine['class label'])) # Class 출력  
11 dat_wine.head()
```

```
↳ class label: [1 2 3]
```

## ■ 학습/테스트 데이터 나누기

```
▶ 1 # 전체 data를 training set과 test set으로 split  
2 from sklearn.model_selection import train_test_split  
3 X, y = dat_wine.iloc[:,1:].values, dat_wine.iloc[:,0].values  
4 X_train, X_test, y_train,y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

# 로지스틱 회귀 실험

## ■ L1, L2 규제화에 따른 실험

```
1 # Logistic Regression with L2 or L1 Regularization
2 from sklearn.linear_model import LogisticRegression
3 lr2_10 = LogisticRegression(penalty='l2', C=10.0, solver='saga') # L2 with C(=1/λ)=10
4 lr2_1 = LogisticRegression(penalty='l2', C=1.0, solver='saga') # L2 with C(=1/λ)=1
5 lr2_0_1 = LogisticRegression(penalty='l2', C=0.1, solver='saga') # L2 with C(=1/λ)=0.1
6
7 lr1_10 = LogisticRegression(penalty='l1', C=10.0, solver='saga') # L1 with C(=1/λ)=10
8 lr1_1 = LogisticRegression(penalty='l1', C=1.0, solver='saga') # L1 with C(=1/λ)=1
9 lr1_0_1 = LogisticRegression(penalty='l1', C=0.1, solver='saga') # L1 with C(=1/λ)=0.1
```

## ■ L1, L2 규제화에 따른 정밀도

```
1 # 규제화 방법(L2 or L1)과 규제강도(λ)를 바꿔가며 accuracy score 계산
2 lr2_10.fit(X_train_std, y_train)
3 print('Training accuracy with L2 and λ=0.1:', lr2_10.score(X_train_std, y_train))
4 print('Test accuracy with L2 and λ=0.1:', lr2_10.score(X_test_std, y_test))
```

accuracy-score & 페인트 무방

```
↳ Training accuracy with L2 and λ=0.1: 1.0
Test accuracy with L2 and λ=0.1: 0.9814814814814815
```

# 로지스틱 회귀 실험

## ■ 규제화 강도에 따른 특성 변수 제거 관련

```
▶ 1 # L2 규제의 규제강도( $C=1/\lambda$ )를 바꿔가며 계수 추정치 관찰  
2 print(lr2_10.intercept_)  
3 print(lr2_1.intercept_)  
4 print(lr2_0_1.intercept_)  
5  
6 print(lr2_10.coef_)  
7 print(lr2_1.coef_)  
8 print(lr2_0_1.coef_)
```

```
▶ [ 0.32296362  0.60033615 -0.92329977]  
[ 0.28173282  0.6024029 -0.88413572]  
[ 0.0683881  0.45688086 -0.52526895]  
[[ 0.125664178  0.15899529  0.39926374 -1.53634982  0.08473759  0.37407767  
  0.83876311 -0.28751864  0.09268499  0.12775152  0.0722625  0.97188895  
  1.39116593]  
[-1.5372395 -0.43915291 -1.23984712  1.21218732 -0.32703465 -0.51670074  
  0.85895303  0.40866438  0.39442514 -1.36535608  1.14060554  0.02219384  
 -1.75612335]  
[ 0.28059772  0.28015763  0.84058338  0.32416249  0.24229707  0.14262307  
 -1.69771614 -0.12114574 -0.48711013  1.23760457 -1.21286804 -0.99408279  
  0.36495742]]  
[[ 0.75495729  0.06165881  0.2336697 -0.8925231  0.02649841  0.29464787  
  0.56064124 -0.2071806  0.13401678  0.12719203  0.10165733  0.61737663  
  0.90976587]  
[-0.98657135 -0.32327905 -0.65176965  0.66792906 -0.22933211 -0.20753188  
  0.43824097  0.19874428  0.24373934 -0.78043161  0.63697798  0.08558965  
 -1.03461549]  
[ 0.23161406  0.26162025  0.41809995  0.22459404  0.2028337 -0.08711598  
 -0.99888221  0.08843632 -0.37775612  0.65323959 -0.73863531 -0.70296628  
  0.12484963]]  
[[ 0.41030119 -0.03148562  0.13676704 -0.41134759  0.05383263  0.22360478  
  0.31670971 -0.15968597  0.11370031  0.07036472  0.1110665  0.30981116  
  0.51691919]  
[-0.5426495 -0.20155646 -0.25667025  0.28071006 -0.14835806 -0.0406011  
  0.12453008  0.0829087  0.10087435 -0.44571802  0.27319166  0.09645505  
 -0.51870207]  
[ 0.13234831  0.23304208  0.1199032  0.13063753  0.09452543 -0.18300368  
 -0.44123979  0.07677727 -0.21457466  0.3753533 -0.38425816 -0.40626621  
  0.00178288]]
```

$C=10$   
 $\downarrow$   
 $C=0.1$  규제↑

O의 개수가 ↑  
⇒ sparse

```
▶ 1 # L1 규제의 규제강도( $C=1/\lambda$ )를 바꿔가며 계수 추정치 관찰  
2 print(lr1_10.intercept_)  
3 print(lr1_1.intercept_)  
4 print(lr1_0_1.intercept_)  
5  
6 print(lr1_10.coef_)  
7 print(lr1_1.coef_)  
8 print(lr1_0_1.coef_)
```

```
▶ [ 0.39334184  0.5958625 -0.98920434]  
[ 0.28462985  0.54413385 -0.8287637 ]  
[ 0.05085504  0.30693808 -0.35779311]  
[[ 1.1999738e+00  0.0000000e+00  1.54580407e-01 -1.57710818e+00  
  0.0000000e+00  2.84110640e-01  8.25653391e-01  0.0000000e+00  
  0.0000000e+00  0.0000000e+00  0.0000000e+00  8.79401443e-01  
  1.62309433e+00]  
[-1.78064452e+00 -4.56713816e-01 -1.47383462e+00  1.05802427e+00  
 -2.92893382e-01 -4.06457188e-01  7.01037258e-01  4.65557053e-01  
  2.08045179e-01 -1.56217122e+00  1.10248967e+00  0.0000000e+00  
 -1.93798201e+00]  
[ 5.3939297e-02  1.17354860e-01  7.93799029e-01  0.0000000e+00  
  1.77579266e-01  0.0000000e+00 -2.03734189e+00 -6.11569248e-06  
 -3.41178760e-01  1.12646922e+00 -1.21297461e+00 -9.54584609e-01  
  0.0000000e+00]]  
[[ 0.0313239  0.          0.          -1.17721748  0.          0.  
  0.02148411  0.          0.          0.          0.          0.62105  
  0.97646011]  
[-1.58450657 -0.1446383 -0.77389205  0.04388529 -0.0731721  0.  
  0.          0.12895574  0.          -0.98296559  0.23622794  0.  
 -1.21587376]  
[ 0.          0.          0.          0.          0.          0.  
 -2.08083573  0.          -0.04410569  0.27113817 -0.80443491 -0.65929035  
  0.          ]]  
[[ 0.          0.          0.          -0.04184623  0.          0.  
  0.23299536  0.          0.          0.          0.          0.  
  0.84042285]  
[-0.8348383  0.          0.          0.          0.          0.  
  0.          0.          0.          -0.4233421  0.          0.  
 -0.20651752]  
[ 0.          0.          0.          0.          0.          0.  
 -0.60150089  0.          0.          0.10503944 -0.3520872 -0.521087  
  0.          ]]
```

★ 로지스틱 회귀 ⇒ 외연, ihs  
데이터적으로 풀어보기  
→ kNN으로 풀어보자!

KNN 문제 속으로  
적용 적용해보기  
중