퍼셉트론

퍼셉트론(perceptron)은 가장 오래되고 단순한 형태의 판별함수기반 분류모형 중 하나이다.

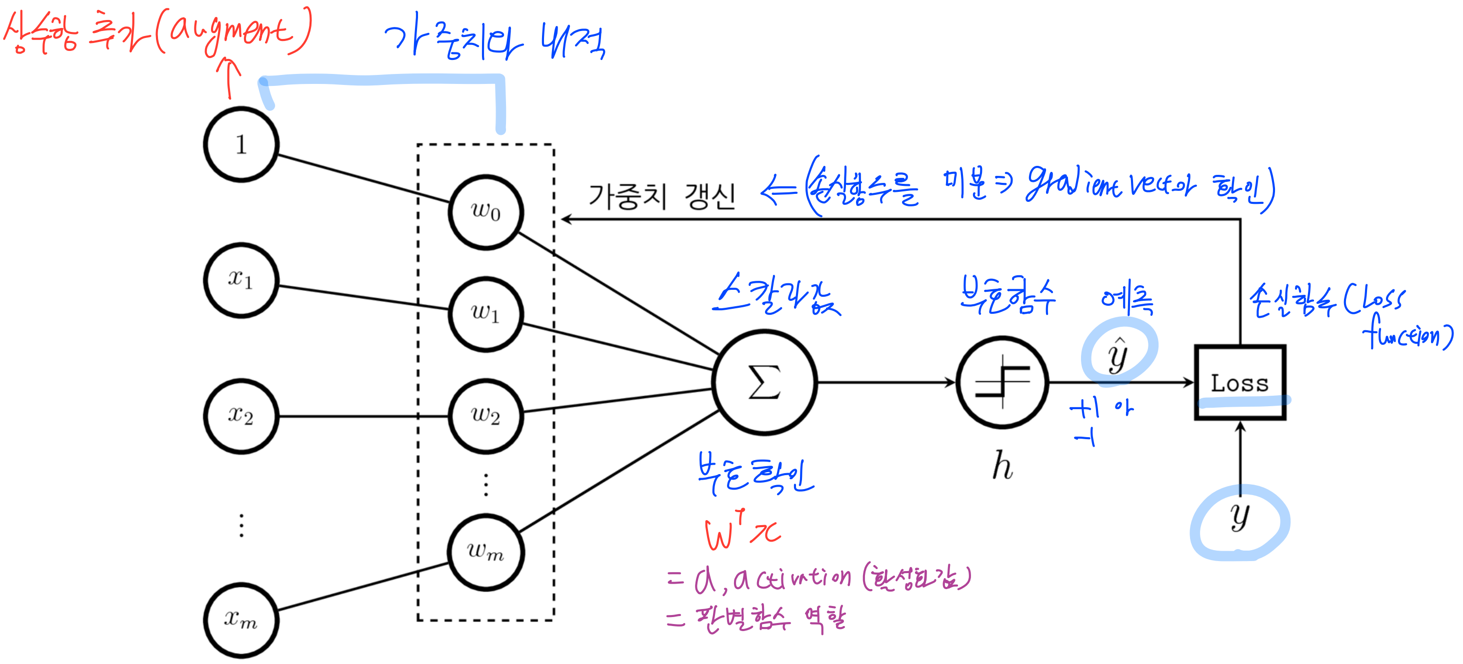


그림 45.1 : 퍼셉트론

상수항 augmented!

퍼셉트론은 입력 $x = (1, x_1, \cdots, x_m)$에 대해 1 또는 -1의 값을 가지는 y를 출력하는 비선형 함수이다. 1을 포함하는 입력 요소 x_i에 대해 가중치 w_i를 곱한 값 $a = w^T x$을 활성화값(activations)이라고 하며 이 값이 판별함수의 역할을 한다.

$$a = w^T x$$

판별 함수 값이 활성화함수(activation function) $h(a)$를 지나면 분류 결과를 나타내는 출력 $\hat{y}$가 생성된다.

$$\hat{y} = h(w^T x)$$

퍼셉트론의 활성화 함수는 부호함수(sign function) 또는 단위계단함수(Heaviside step function)라고 부르는 함수이다.

$$h(a) = \begin{cases} -1, & a < 0, \\ 1, & a \ge 0 \end{cases}$$

↳ Deep learning 에선, 로지스틱 함수, ReLU 함수등을 사용

퍼셉트론 손실함수

다음과 같이 N개의 학습용 데이터가 있다고 하자.

$$(x_1, y_1), (x_2, y_2), \ldots, (x_i, y_i), \ldots, (x_N, y_N)$$

↳ +1 or -1

퍼셉트론은 독립변수 x로부터 종속변수 y를 예측하는 예측 모형이므로 모든 학습 데이터에 대해 예측 오차를 최소화하는 가중치 w를 계산해야 한다. 가중치 w에 따라 달라지는 전체 예측 오차 L는 i번째 개별 데이터에 대한 손실함수 $L_i(\hat{y}_i, y_i)$의 합으로 표현할 수 있다.

1~N 개 데이터 각각의 손실함수값 누적 합.

$$L = \sum_{i=1}^{N} \underbrace{L_i(y_i, \hat{y}_i)}_{\text{개별 데이터 'i' 1개의 손실함수 값}}$$

회귀분석에선, RSS가 Loss function 역할

손실 $L_i(y_i, \hat{y}_i)$는 실제값 y와 예측값 $\hat{y}$의 차이를 나타내는 함수이다. 회귀 분석에서는 $L(\hat{y}, y) = -(y - \hat{y})^2$과 같은 손실함수를 많이 사용하였지만 퍼셉트론의 경우에는 다음과 같은 손실 함수를 사용한다. 이를 제로-원 **손실함수(zero-one loss function)**이라고 한다.

$$L_i(y_i, \hat{y}_i) = \max(0, -y_i \hat{y}_i)$$

$\hat{y} = \{ {1 \atop -1}$ $y = \{ {1 \atop -1}$

제로-원 손실함수 L_i은 $\hat{y}$과 y가 같으면 0이고 다르면 1이다. 다음처럼 서술할 수도 있다.

$$L_i(\hat{y}) = \begin{cases} \frac{1}{2}(\text{sgn}(-\hat{y}) + 1) & \text{if } y = 1 \\ \frac{1}{2}(\text{sgn}(\hat{y}) + 1) & \text{if } y = -1 \end{cases}$$

$y \backslash \hat{y}$	1	-1
1	0	1
-1	1	0

$L_i(y_i, \hat{y}_i) = \max(0, -y_i \hat{y}_i)$

∴ 맞추면 Loss '0'
틀리면 Loss '1'

전체 손실함수는

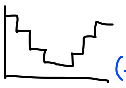
Loss 값은 틀린문제에만 존재

$$L = \sum_{i=1}^{N} \max(0, -y_i \hat{y}_i) = -\sum_{i \in M} y_i \hat{y}_i$$

가 된다. 이 식에서 M은 오분류(misclassification)된 데이터의 집합이다. y와 $\hat{y}$값이 다르면 오분류된 것이다.

$$\hat{y} = y \rightarrow \quad \text{right classficiation}$$
$$\hat{y} \neq y \rightarrow \quad \text{misclassification}$$

그런데 제로-원 손실함수를 쓰면 $\hat{y}(x)$가 x에 대한 계단형 함수이므로 대부분의 영역에서 기울기가 0이 되어 미분값으로부터 최소점의 위치를 구할 수 없다. 따라서 퍼셉트론에서는 $\hat{y}$ 대신 활성화값 $w^T x$를 손실함수로 사용한다.

$\Sigma L(\hat{y}_i) =$  (수치적 최적화 X)

$\hat{y} = h(w^T x)$
$\hat{y}_i = h(w_i^T x_i)$

$$L_P(w) = -\sum_{i \in M} y_i \cdot w^T x_i$$

$= \Sigma \max(0, -y_i w_i^T x_i)$

$\Sigma L_p(w_i)$ (수치적 최적화 O)

이를 **퍼셉트론 손실함수(perceptron loss function)** 또는 **0-힌지 손실함수(zero-hinge loss function)**라고 한다. 여기에서 손실값은 오분류된 표본에 대해서만 계산한다는 점에 주의하라. 이 때는 y와 $\text{sgn}(\hat{y})$값이 다르면 오분류된 것이다.

$$\text{sgn}\hat{y} = y \rightarrow \quad \text{right classficiation}$$
$$\text{sgn}\hat{y} \neq y \rightarrow \quad \text{misclassification}$$

퍼셉트론 손실함수는 다음처럼 표기할 수도 있다.

$$L_{P,i}(\hat{y}) = \begin{cases} -\frac{1}{2} w^T x \, (\text{sgn}(-\hat{y}) + 1) & \text{if } y = 1 \\ \frac{1}{2} w^T x \, (\text{sgn}(\hat{y}) + 1) & \text{if } y = -1 \end{cases}$$

가중치 계산

퍼셉트론 손실함수 $L_P(w)$를 최소화하는 w를 찾기 위해 $L_P(w)$를 w로 미분하여 그레디언트를 구하면 다음과 같다.

$$\frac{dL_P}{dw} = -\sum_{i \in M} x_i y_i$$

그레디언트 디센트(gradient descent) 방법을 사용하면 다음과 같이 w를 갱신할 수 있다.

$$w_{k+1} = w_k + \eta_k \sum_{i \in M} x_i y_i$$

$W_{k+1} = W_k + \eta_k \nabla L_P$

* gradient descent 방식

$w_{k+1} = w_k - \mu_k \nabla f_k$

여기에서 η는 스텝사이즈(step size) 또는 학습속도(learning rate)라고 한다.

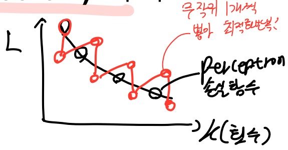
실제로는 계산량을 줄이기 위해 전체 오분류 데이터 집합 M 중에서 하나만 골라서 사용한다. 다음 식에서 m은 오분류된 데이터 중의 하나를 무작위로 고른 것이다. ($m \in M$)

$$w_{k+1} = w_k + \eta_k x_m y_m$$

또 $\hat{y}$이 1또는 -1의 값만 가질 수 있으므로 실제로는 다음과 같은 식이 된다.

$$w_{k+1} = \begin{cases} w_k + \eta_k x_m & \text{if } \hat{y}_m = 1 \neq y_m \\ w_k - \eta_k x_m, & \text{if } \hat{y}_m = -1 \neq y_m \end{cases}$$

그런데 퍼셉트론 손실함수는 원래의 손실함수와 정확하게 같지 않기 때문에 이러한 방식으로 학습을 했을 때 매 단계마다 반드시 원래의 손실함수가 감소한다는 보장은 없다. 다만 **퍼셉트론 수렴 정리(perceptron convergence theorem)**로부터 데이터가 선형분리(linearly separable)가능한 경우에는 완전분류모형으로 수렴한다는 것이 증명되어 있다.



↳ M 개 중 1개(m)으로만 Wk+1 구해도 반복하면 완전분류모형으로 수렴!

(전체 다써서 이동한 Wk+1 과 무작위로 1개만 써서 이동한 Wk는 (M개 모두 포괄) 결국엔 같다. but 무작위로 1개만 쓰면 수렴속도가 느린 것 뿐!)

Scikit-Learn 의 퍼셉트론 구현

Scikit-Learn에서 제공하는 퍼셉트론 모형인 `Perceptron` 클래스는 다음과 같은 입력 인수를 가진다.

- `max_iter` : 최적화를 위한 반복 횟수(iteration number)
- `eta0` : 학습속도 η
- `n_iter_no_change` : 이 설정값만큼 반복을 해도 성능이 나아지지 않으면 `max_iter` 설정값과 상관없이 멈춘다.

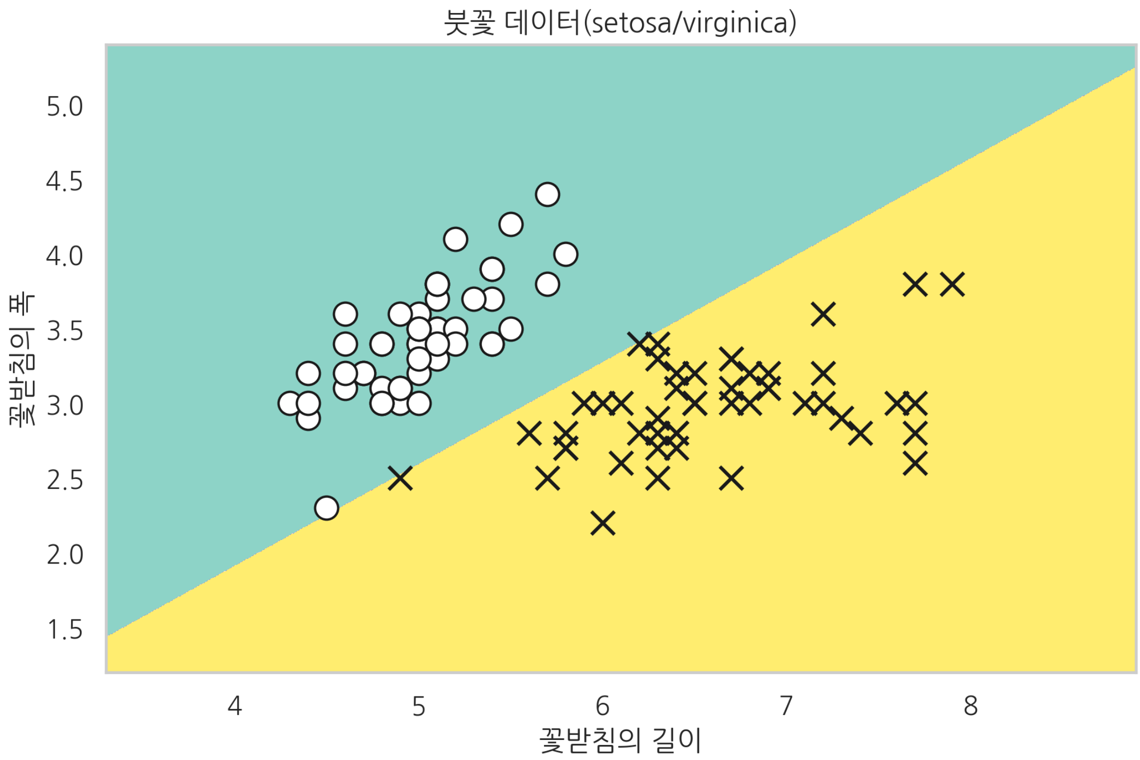
퍼셉트론 실행 예 (1번씩 무작위로 뽑은 x_m으로 손실함수 적용해 수치적 최적화!)

In [1]:

```
from sklearn.datasets import load_iris
iris = load_iris()

idx = np.in1d(iris.target, [0, 2])
X = iris.data[idx, :2]
y = (iris.target[idx] / 2).astype(np.int)

from sklearn.linear_model import Perceptron

def plot_perceptron(n):
    model = Perceptron(max_iter=300, shuffle=False, tol=0, n_iter_no_change=1e9)
.fit(X, y)
    XX_min = X[:, 0].min() - 1
    XX_max = X[:, 0].max() + 1
    YY_min = X[:, 1].min() - 1
    YY_max = X[:, 1].max() + 1
    XX, YY = np.meshgrid(np.linspace(XX_min, XX_max, 1000),
                         np.linspace(YY_min, YY_max, 1000))
    ZZ = model.predict(np.c_[XX.ravel(), YY.ravel()]).reshape(XX.shape)
    plt.contourf(XX, YY, ZZ, cmap=mpl.cm.Set3)
    plt.scatter(X[y==0, 0], X[y==0, 1], c='w', s=100, marker='o', edgecolor='k')
    plt.scatter(X[y==1, 0], X[y==1, 1], c='k', s=100, marker='x', edgecolor='k')
    plt.xlabel("꽃받침의 길이")
    plt.ylabel("꽃받침의 폭")
    plt.title("붓꽃 데이터(setosa/virginica)")
    plt.xlim(XX_min, XX_max)
    plt.ylim(YY_min, YY_max)
    plt.grid(False)
    plt.show()


from ipywidgets import widgets
widgets.interact(plot_perceptron, n=widgets.IntSlider(min=1, max=100, step=1, va
lue=1));
```

In [2]:

```
plot_perceptron(300)
```



In [3]:

```
from sklearn.metrics import confusion_matrix, classification_report

model = Perceptron(max_iter=400, shuffle=False, tol=0, n_iter_no_change=1e9).fit
(X, y)
confusion_matrix(y, model.predict(X))
```

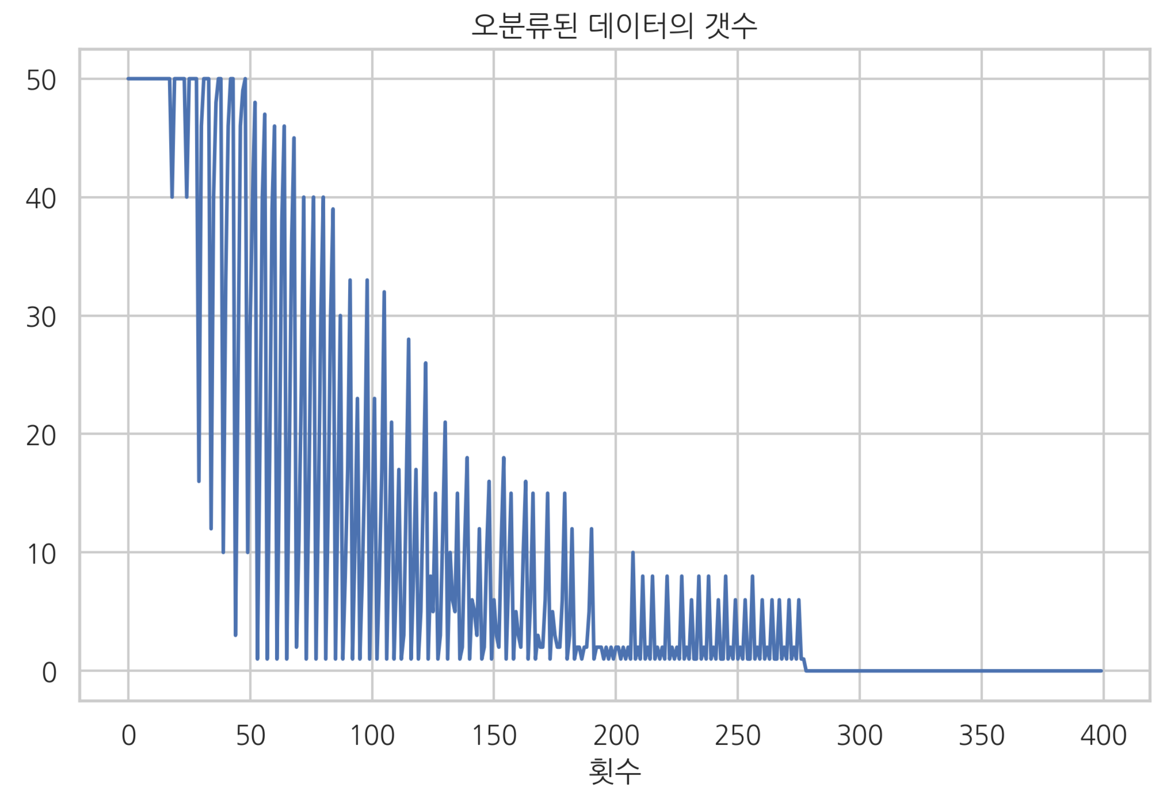
Out[3]:

```
array([[50,  0],
       [ 0, 50]])
```

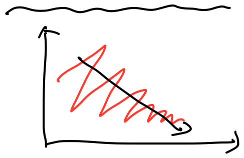
학습 성능 (1개씩 사용해서 반복한 SGD)

In [4]:

```
n = 400
loss = np.zeros(n)
model = Perceptron(warm_start=True, shuffle=False)
for i in range(n):
    model.partial_fit(X, y, classes=[0, 1])
    loss[i] = np.sum(y != model.predict(X))

plt.plot(loss)
plt.xlabel("횟수")
plt.title("오분류된 데이터의 갯수")
plt.show()
```



SGD Stochastic Gradient Descent



파동처럼 오르내리는 양태 = Stochastic

SGD(Stochastic Gradient Descent) 방법은 손실함수 자체가 아니라 손실함수의 기댓값을 최소하는 방법이다.

$$\arg\min \mathrm{E}[L]$$

전체 손실함수 L는 개별 데이터의 손실함수 $L_i(\hat{y}_i, y_i)$의 합이다.

$$E = \sum_i L_i(\hat{y}_i, y_i)$$

mini batch (일부 데이터)
⇓
표본평균 $\xrightarrow{\text{추정}}$ 기대값

그레디언트 디센트(gradient descent) 방법을 사용하면 다음과 같이 w를 갱신할 수 있다.

$$w_{k+1} = w_k + \eta_k \sum_{i \in M} x_i y_i$$

$w_{k+1} = w_k + \eta_k \nabla L_p$

* gradient descent 방식
$w_{k+1} = w_k - \mu_k \nabla f_k$

SGD 최적화 방법은 그레디언트가 아니라 그레디언트의 기댓값의 추정치를 이용한다.

SGD: 오분류된 데이터의 손실함수 기대값 E[L]
″ 그레디언트 기대값 E[∇L]

$$w_{k+1} = w_k + \mathrm{E}[\nabla L]$$

그레디언트의 기댓값의 추정치는 표본 평균이다. 즉 모든 학습용 데이터를 다 사용하여 그레디언트를 구하는 것이 아니라 미니배치(minibatch)라고 부르는 일부의 데이터만 사용하여 그레디언트 추정치를 구한다. 따라서 한번의 계산량이 많거나 학습 데이터가 많은 딥러닝(deep learning)에 사용된다. 퍼셉트론은 오분류된(mis-classified) 데이터만 이용하는 SGD의 일종이다.

SGD 방법이 기댓값이 최소화되도록 수렴한다는 것은 다양한 논문에서 증명이 되어 있다. 다만 손실함수 자체를 최적화하는 것이 아니라 손실함수의 기댓값의 추정치를 ~~최대화~~ 최적화하기 때문에 손실함수값이 전반적으로 감소하는 추세를 보이는 것 뿐이고 항상 절대적으로 감소한다는 보장은 없다.

SGD에서는 제로-원이나 퍼셉트론 손실함수 이외에도 손실함수가 볼록함수(convex function)이면 모두 개별 데이터 손실함수로 사용할 수 있다. 다음 그림에서는 $y = 1$인 경우 많이 사용되는 손실함수의 값을 나타내었다.

In [5]:

```
def modified_huber_loss(y_true, y_pred):
    z = y_pred * y_true
    loss = -4 * z
    loss[z >= -1] = (1 - z[z >= -1]) ** 2
    loss[z >= 1.] = 0
    return loss

xmin, xmax = -4, 4
xx = np.linspace(xmin, xmax, 100)
plt.plot([xmin, 0, 0, xmax], [1, 1, 0, 0], color='k', lw=2, ls=":", label="제로-원
손실함수")
plt.plot(xx, np.where(xx < 1, 1 - xx, 0), color='teal', lw=2, ls="-", label="힌지
손실함수")
plt.plot(xx, -np.minimum(xx, 0), color='yellowgreen', lw=2, ls="-.", label="퍼셉트
론 손실함수")
plt.plot(xx, np.log2(1 + np.exp(-xx)), color='cornflowerblue', lw=2, label="로그
 손실함수")
plt.plot(xx, np.where(xx < 1, 1 - xx, 0) ** 2, color='orange', lw=2, label="제곱
 힌지 손실함수")
plt.plot(xx, modified_huber_loss(xx, 1), color='darkorchid', lw=2, ls='--', labe
l="수정 휴버 손실함수")
plt.ylim((0, 8))
plt.legend(loc="upper right")
plt.xlabel(r"$w^Tx$")
plt.ylabel(r"$L(y=1, w^Tx)$")
plt.title("여러가지 손실함수의 형태 (y=1인 경우)")
plt.show()
```

다. 이를 제로-원 손실함수(zero-one

$\hat{y} = \{^{1}_{-1}$ $y = \{^{1}_{-1}$

있다.

$y \backslash \hat{y}$	1	-1
1	0	1
-1	1	0

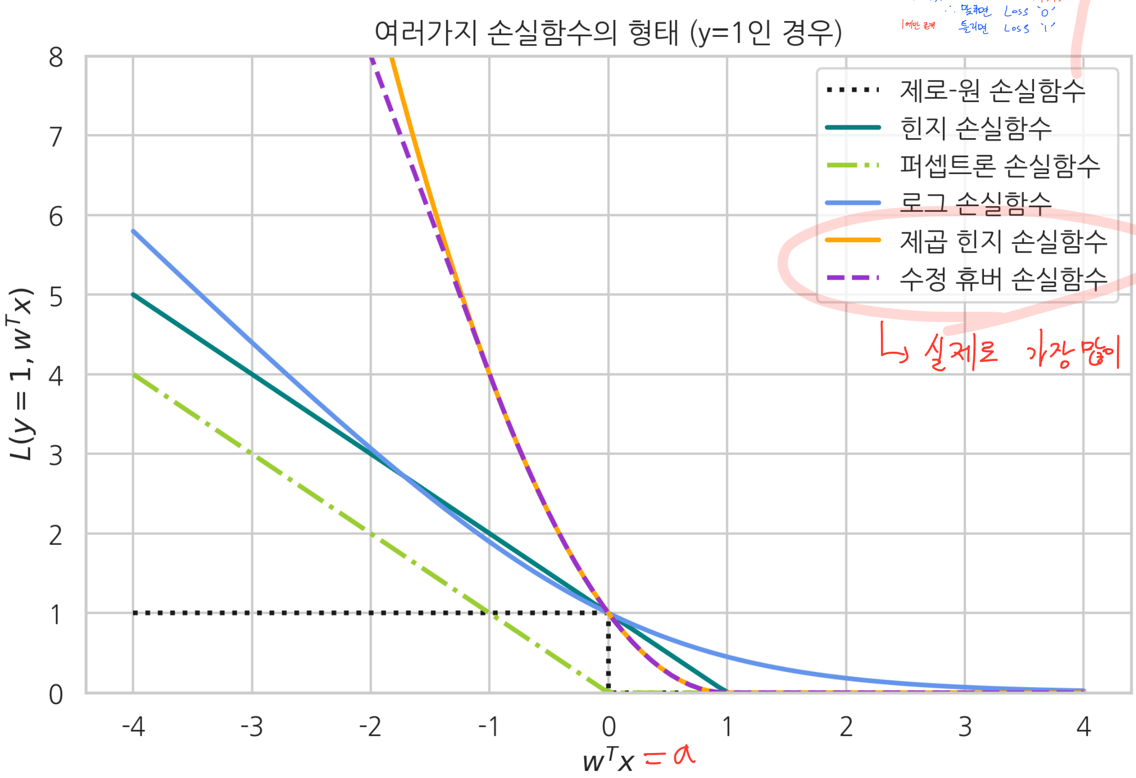
$L_i(y_i, \hat{y}_i) = max(0, -y_i\hat{y}_i)$

∴ 맞히면 Loss '0'

1에만 존재 틀리면 Loss '1'

$L_i = max(0, -\hat{y}_i y_i)$

$= 1$ or 0



↳ 실제로 가장 많이 활용!

$L = \sum_{i=1}^{N} \max(0, -y_i\hat{y}_i) = -\sum_{i \in M} y_i\hat{y}_i$

Loss 값은 틀린문제에만 존재

$\hat{y} = h(a)$ $(a = w^Tx)$ 〈제로원〉 $a>0 \Rightarrow \hat{y}=1$, $a<0 \Rightarrow \hat{y}=-1$ ¦ $y=1$ ¦ Zero one $\{^{0 \text{ (맞으면)}}_{1 \text{ (틀리면)}}$

$L_P(w) = -\sum_{i \in M} y_i \cdot w^T x_i$

$= \sum max(0, -y_i w_i^T x_i)$

$\hat{y}$ 대신 w^Tx를 Loss function에! 〈퍼셉트론〉 제로 힌지 $a>0 \Rightarrow \hat{y}=1$, $L=0$; $a<0 \Rightarrow \hat{y}=-1$, $L=-ay_i$ $(a<0)$

⇓

수치적 최적화 가능! (GD + 퍼셉트론 수렴정리, SGD)

Scikit-Learn 의 SGD 구현

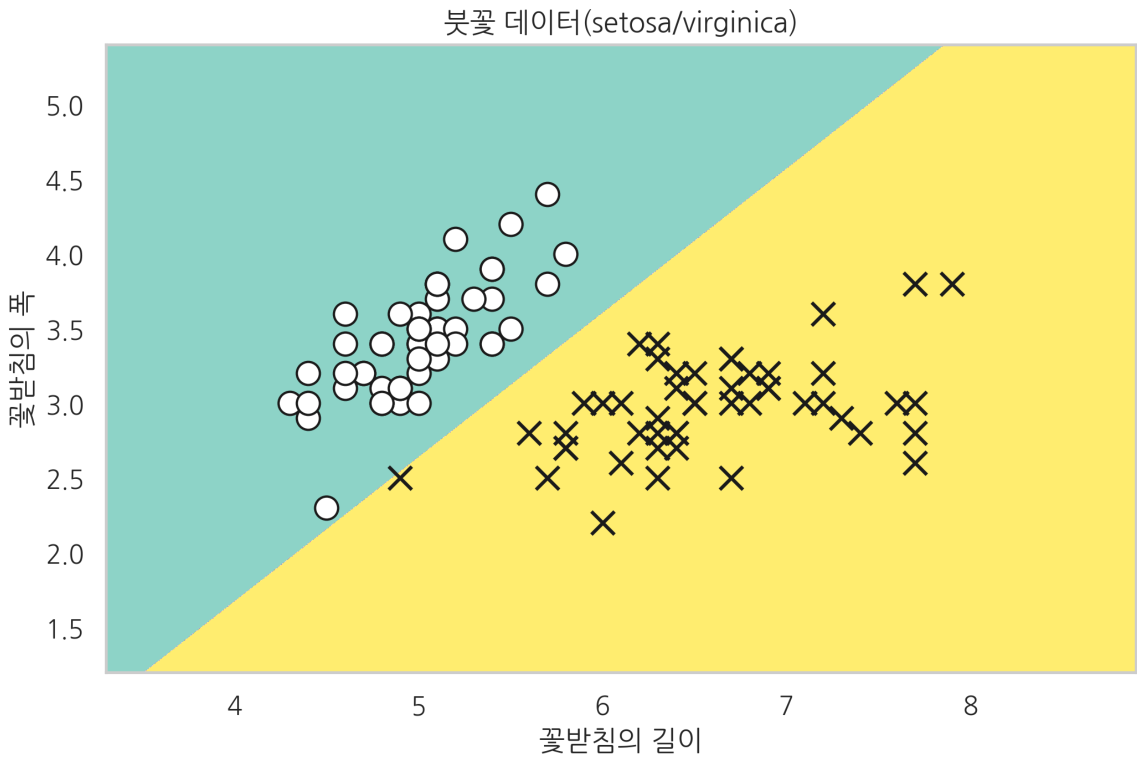
Scikit-Learn에서 제공하는 `SGDClassifier` 클래스는 `Perceptron` 클래스에의 입력 인수 이외에도 손실함수를 결정하는 `loss` 인수를 가진다. 가능합 값은 `hinge`, `perceptron`, `log`, `huber`, `modified_huber`, `squared_hinge` 등이다. 보통 `modified_huber` 를 사용한다.

In [6]:

```
from sklearn.linear_model import SGDClassifier

def plot_sgd(n):
    model = SGDClassifier(loss="modified_huber", max_iter=n, shuffle=False, n_it
er_no_change=1e9).fit(X, y)
    XX_min = X[:, 0].min() - 1
    XX_max = X[:, 0].max() + 1
    YY_min = X[:, 1].min() - 1
    YY_max = X[:, 1].max() + 1
    XX, YY = np.meshgrid(np.linspace(XX_min, XX_max, 1000),
                         np.linspace(YY_min, YY_max, 1000))
    ZZ = model.predict(np.c_[XX.ravel(), YY.ravel()]).reshape(XX.shape)
    cmap = mpl.colors.ListedColormap(sns.color_palette("Set2"))
    plt.contourf(XX, YY, ZZ, cmap=mpl.cm.Set3)
    plt.scatter(X[y==0, 0], X[y==0, 1], c='w', s=100, marker='o', edgecolor='k')
    plt.scatter(X[y==1, 0], X[y==1, 1], c='k', s=100, marker='x', edgecolor='k')
    plt.xlabel("꽃받침의 길이")
    plt.ylabel("꽃받침의 폭")
    plt.title("붓꽃 데이터(setosa/virginica)")
    plt.xlim(XX_min, XX_max)
    plt.ylim(YY_min, YY_max)
    plt.grid(False)
    plt.show()


from ipywidgets import widgets
widgets.interact(plot_sgd, n=widgets.IntSlider(min=1, max=100, step=1, value=1
));
```

In [7]:

```
plot_sgd(400)
```



In [8]:

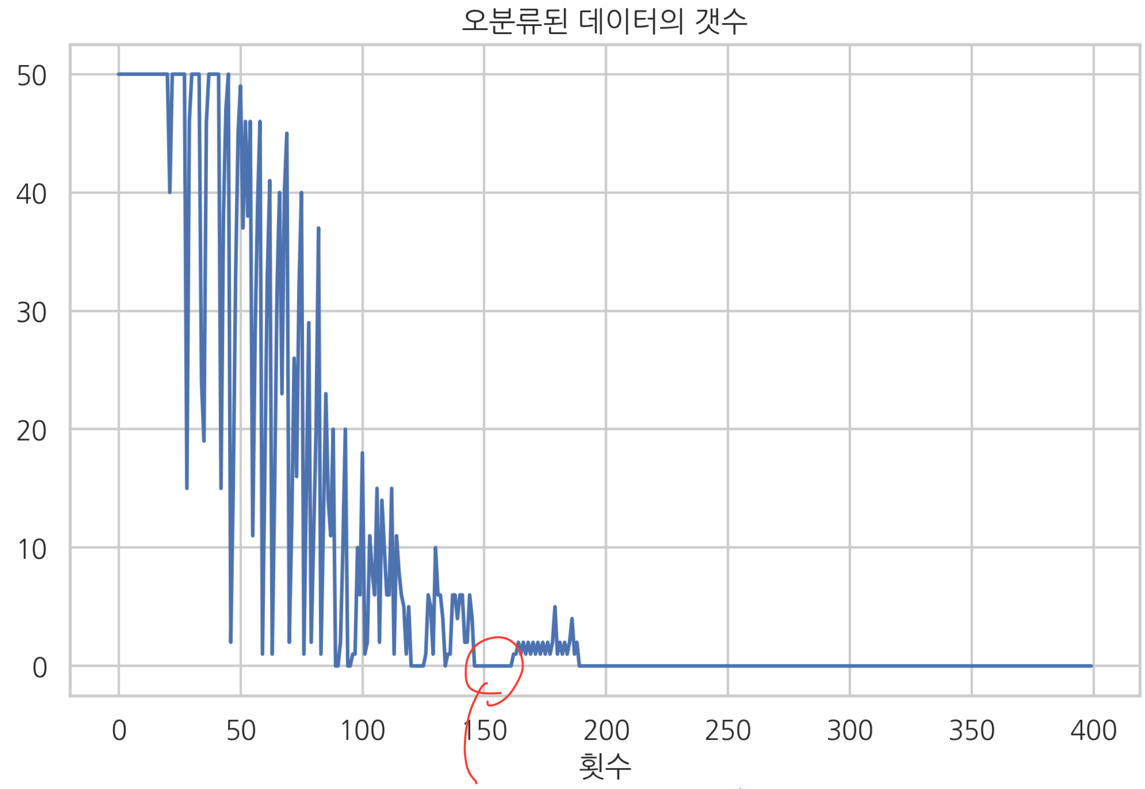
```
model = SGDClassifier(loss="modified_huber", max_iter=400, shuffle=False, n_iter
_no_change=1e9).fit(X, y)
confusion_matrix(y, model.predict(X))
```

Out[8]:

```
array([[50,  0],
       [ 0, 50]])
```

In [9]:

```
n = 400
loss = np.empty(n)
model = SGDClassifier(loss="modified_huber", shuffle=False)
for i in range(n):
    model.partial_fit(X, y, classes=[0, 1])
    loss[i] = np.sum(y != model.predict(X))

plt.plot(loss)
plt.xlabel("횟수")
plt.title("오분류된 데이터의 갯수")
plt.show()
```



'0'이 되어도 계속 시도!

∴ 직접 눈으로 이걸 보며 stop 시켜야 함