기계학습 - 230926

> 학습내용
>
> **[5가지 알고리즘]**
> - 정규방정식 (numpy)
> - SVD (사이킷런)
> - 배치 경사 하강법 (사이킷런)
> - 확률적 경사 하강법 (사이킷런)
> - 미니배치 경사 하강법 (사이킷런)
>
> **[다항회귀]**

1.선형 회귀

- 입력 특성의 가중치 합과 편향이라는 상수를 더해 예측함.

식 4-1 선형 회귀 모델의 예측

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

→ 가중치값의 최적값 찾기

- 선형 회귀 모델의 MSE 비용 함수
  - 선형 회귀 모델을 훈련시키려면 MSE를 최소화하는 θ를 찾아야 함.

$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=0}^{m-1}(\theta^T (\mathbf{x}_b^{(i)})^T - y^{(i)})^2$$

→ 비용함수 기울기가 수렴(안기울어짐)일때까지

선형회귀 코드 설명하기

X: (100, 1) 행렬로 2를 곱하여 0~2 사이의 값을 가짐

Y: X의 값에 대한 선형관계(4+3*X)를 통해 값을 생성하고, 가우시안노이즈(np.random.randn(100, 1))를 추가하여 데이터에 무작위성을 부여함

```
import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

```
linreg_data = np.c_[X, y]
linreg_data[:5]
```

2.1경사 하강법

- 임의의 값으로 시작해서 조금씩 비용 함수가 감소하는 방향으로 진행.
- 알고리즘이 최솟값에 수렴할 때 까지 점진적으로 향상시킴

경사 하강법 중요 파라미터 = 학습률 하이퍼파라미터 = 스텝의 크기

경사 하강법 단점: 전역 최솟값 보다 덜 좋은 지역 최솟값에 수렴

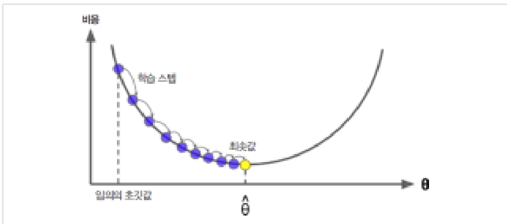
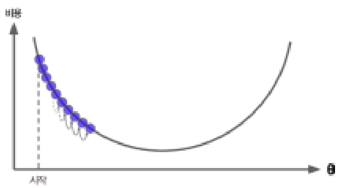


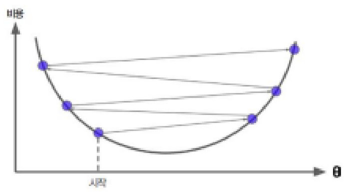
그림 4-3 이 경사 하강법 그림에서 모델 파라미터가 무작위하게 초기화된 후 반복적으로 수정되어 비용 함수를 최소화합니다. 학습 스텝 크기는 비용 함수의 기울기에 비례합니다. 따라서 파라미터가 최솟값에 가까워질수록 스텝 크기가 점진적으로 줄어듭니다.

→ 임의의 초깃값이 매우 중요(잘못잡으면 학습이 안됨)

- 학습률이 너무 작을 때 – 시간이 오래 걸림.



- 학습률이 너무 클때 – 발산



→ 학습률도 중요

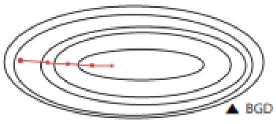
2.4 배치 경사 하강

훈련전체데이터로 그레디언트 계산

비용적음= 내적이180도

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \ \text{MSE}(\theta)$$

▲ 경사 하강법 스텝



▲ BGD 학습의 시각화

→ 위의 수식을 eta라고 함(학습률)

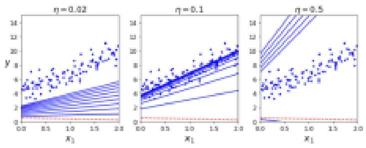
+a) 에폭(epoch) :

특징)

1오래걸림

2메모리 많이 필요

3안정적인 수렴



▲ 여러 가지 학습률에 대한 경사 하강법

eta값이 적으면 시간오래걸림 너무크면 발산

역전파, 미분, 그레디언트 의 연관성 다시한번 보기

```
다빈: 미분은 함수의 변화량을 나타내고, 그레디언트는 다변수 함수에서 각 변수에 대한 미분값을 벡터로 표현한 것
역전파는 인공 신경망에서 그레디언트를 효율적으로 계산하는 알고리즘이야.

한결: 정리하자면 딥 러닝 모델을 학습시킬 때, 미분은 손실 함수의 변화량을 계산하는 데 사용되며,
그레디언트는 이러한 변화량을 모든 변수에 대해 구하는 데 사용되네. 그렇게 해서 역전파는 그레디언트를
효율적으로 계산하는 알고리즘인거야.
```

2.5 확률적 경사 하강

학습 스케줄

▼ 확률적 경사 하강법 함께 코드 짜기

직접 수정하는 방법

```
n_epochs = 50
t0, t1 = 5, 50  #학습 스케줄 하이퍼파라미터

def learning_schedule(t):
    return t0 / (t + t1)
```

```
theta = np.random.randn(2, 1)  #무작위 초기화

# 확률적 경사 하강법 시작
for epoch in range(n_epochs):
    for i in range(m):  #각 에포크에서 데이터셋의 모든 샘플에 대해 반복
        random_index = np.random.randint(m)  #하나의 샘플을 무작위로 선택
        xi = X_b[random_index:random_index + 1]
        yi = y[random_index:random_index + 1]

        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)  #선택된 샘플에 대한 그래디언트 계산
        eta = learning_schedule(epoch * m + i)  #학습률 계산
        theta = theta - eta * gradients  #계산된 그래디언트와 학습률을 사용하여 매개변수 업데이트
```

sklearn 라이브러리를 사용하는 방법

→ 사이킷런 SGDRegressor() 메서드는 손실 함수로 MSE를 사용하여 경사하강법을 진행

```
from sklearn.linear_model import SGDRegressor
sr = SGDRegressor(max_iter=1000, eta0=1e-4, random_state=0, verbose=1)
sr.fit(X_train, y_train)
```

▼ 정답코드

```
n_epochs = 50   # 에포크 수, 총 50번의 에포크 기간동안 훈련 진행
t0, t1 = 5, 50  # 학습 스케줄을 위한 하이퍼파라미터 역할 수행

def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)    # 파라미터 랜덤 초기화

for epoch in range(n_epochs): #에폭만큼 돌린다

    # 매 샘플에 대해 그레이디언트 계산 후 파라미터 업데이트
    for i in range(m):

        # 처음 20번 선형 모델(직선) 그리기
        if epoch == 0 and i < 20:
            y_predict = X_new_b.dot(theta)
            style = "b-" if i > 0 else "r--"
            plt.plot(X_new, y_predict, style)

        # 파라미터 업데이트
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]

        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)  # 하나의 샘플에 대한 그레이디언트 계산
        eta = learning_schedule(epoch * m + i)        # 학습 스케쥴을 이용한 학습률 조정
        theta = theta - eta * gradients
        theta_path_sgd.append(theta)

plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 2, 0, 15])
save_fig("sgd_plot")
plt.show()
```

2.6 미니배치
- 미니배치라 부르는 임의의 작은 샘플 세트에 대해 그레이디언트를 계산.
- BGD와 SGD의 절충안.
- SGD 비해 미니배티 경사 하강법의 장단점

- 행렬 연산에 최적화된 하드웨어, GPU 구조 때문에 연산이 빨라짐.
- SGD보다 덜 불규칙
- SGD보다 전역 최솟값에 더 가까이 도달하게 됨. 그러나 지역 최솟값은 빠져 나오기 더 힘들수 있음.

▼ 에폭을 이해하자!

Q)

가중치를 몇번 업데이트 할 수 있는가

에폭=100, DataSet = 1000, 미니배치=50

```
한결조사: 1에폭은 각 데이터사이즈가 50인 배치가 들어간 20개의 iteration으로 나누어진다
따라서 데이터세트 / 배치사이즈 = 미니배치

다빈계산: 1000 / 배치사이즈 = 50 → 배치사이즈 = 20 이므로 50개의 미니배치로 나누어 학습하는 경우
배치사이즈는 20이되고, 확률적 경사 하강법(SGD)를 50번 반복하면 모든 훈련 데이터를 소진하게 된다.
이때 SGD회가 1에폭이 된다
```

- 총 데이터셋의 크기: m = 1000
- 미니배치의 크기: $minibatch$ = 50
- 에포크 수: n_epochs = 100

한 번의 에포크에서 수행되는 미니배치의 수는 전체 데이터셋 크기를 미니배치 크기로 나눈 것이므로:

$$iterations\_per\_epoch = \frac{m}{minibatch} = \frac{1000}{50} = 20$$

따라서 전체 반복 횟수(전체 미니배치의 수)는:

$$total\_iterations = iterations\_per\_epoch \times n\_epochs = 20 \times 100 = 2000$$

이유:

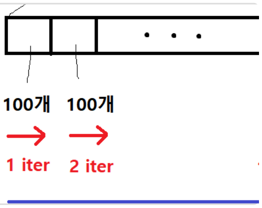
1. 미니배치 경사하강법은 각 에포크에서 전체 데이터셋을 미니배치 크기로 나누어 여러 미니배치를 형성합니다.
2. 각 미니배치에 대해 그래디언트를 계산하고 매개변수를 업데이트합니다.
3. 따라서 한 번의 에포크에서 $m/minibatch$만큼의 업데이트(반복)가 발생합니다.
4. 이를 모든 에포크에 대해 수행하므로 총 반복 횟수는 $iterations_per_epoch \times n_epochs$가 됩니다.

▼ 참조

에폭(epoch), 배치 사이즈(batch size), 미니 배치(mini batch), 이터레이션(iteration)

에폭(epoch)이란? 배치 사이즈(batch size)란? 에폭(epoch): 하나의 단위. 1에폭은 학습에서 훈련 데이터를 모두 소진했을 때의 횟수에 해당함. 미니 배치(mini batch): 전체 데이터 셋을 몇 개의 데이터 셋으로 나누었을 때, 그 작은 데이터 셋 뭉치 배치 사이즈(batch size): 하나의 미니

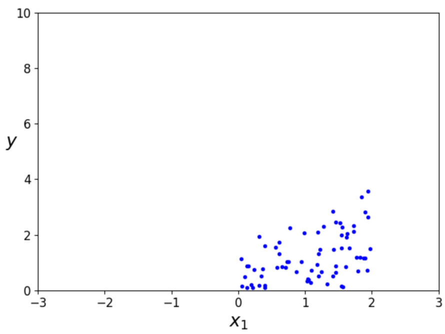
https://mole-starseeker.tistory.com/59



3.다항회귀

- 비선형 데이터를 학습하기 위해 선형 모델을 사용하는 기법
- 훈련 세트에 있는 각 특성을 제곱하여 새로운 특성을 추가 -> 확장된 훈련 데이터에 선형회귀 적용

```
m = 100
x = 6 * np.random.rand(m,1) -3
y = 0.5 * X**2 + np.random.randn(m, 1)


from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=3, include_bias=False)
X_poly = poly_features.fit_transform(X)
X[0]

lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
lin_reg.intercept_, lin_reg.coef_

plt.plot(X, y, "b.")
plt.xlabel("
", fontsize=18)
plt.ylabel("
", rotation=0, fontsize=18)
plt.axis([-3, 3, 0, 10])
save_fig("quadratic_data_plot")
plt.show()
```



4. 학습곡선

- 훈련 세트와 검증 세트의 모델 성능을 훈련 세트 크기(또는 훈련 반복)의 함수로 나타냄

```
from sklearn.metrics import mean_squared_error           # MSE 수동 계산
from sklearn.model_selection import train_test_split     # 무작위 샘플링

def plot_learning_curves(model, X, y):
    # 8:2 로 분류
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=10)
    train_errors, val_errors = [], []                    # MSE 추적 장치

    for m in range(1, len(X_train)):                     # m 개의 훈련 샘플을 대상으로 훈련
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        # MSE 기록
        train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))

    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
    plt.legend(loc="upper right", fontsize=14)
    plt.xlabel("Training set size", fontsize=14)
    plt.ylabel("RMSE", fontsize=14)
```

과소적합 학습곡선

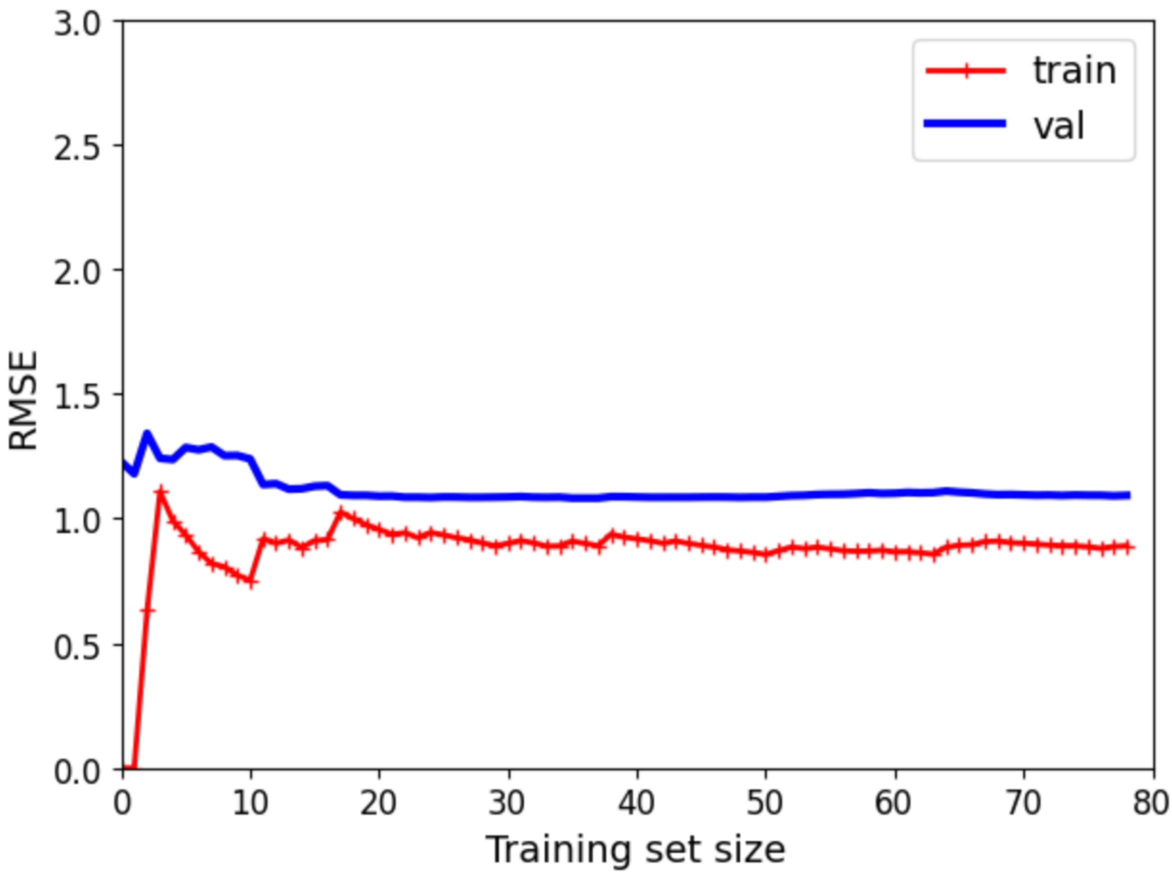
훈련 데이터(빨강)에 대한 성능

- 훈련 세트가 커지면서 RMSE(평균 제곱근 오차)가 커짐

훈련 세트가 어느 정도 커지면 더 이상 RMSE가 변하지 않음

검증 데이터(파랑)에 대한 성능

- 검증 세트에 대한 성능이 훈련 세트에 대한 성능과 거의 비슷해짐



과대적합 학습곡선

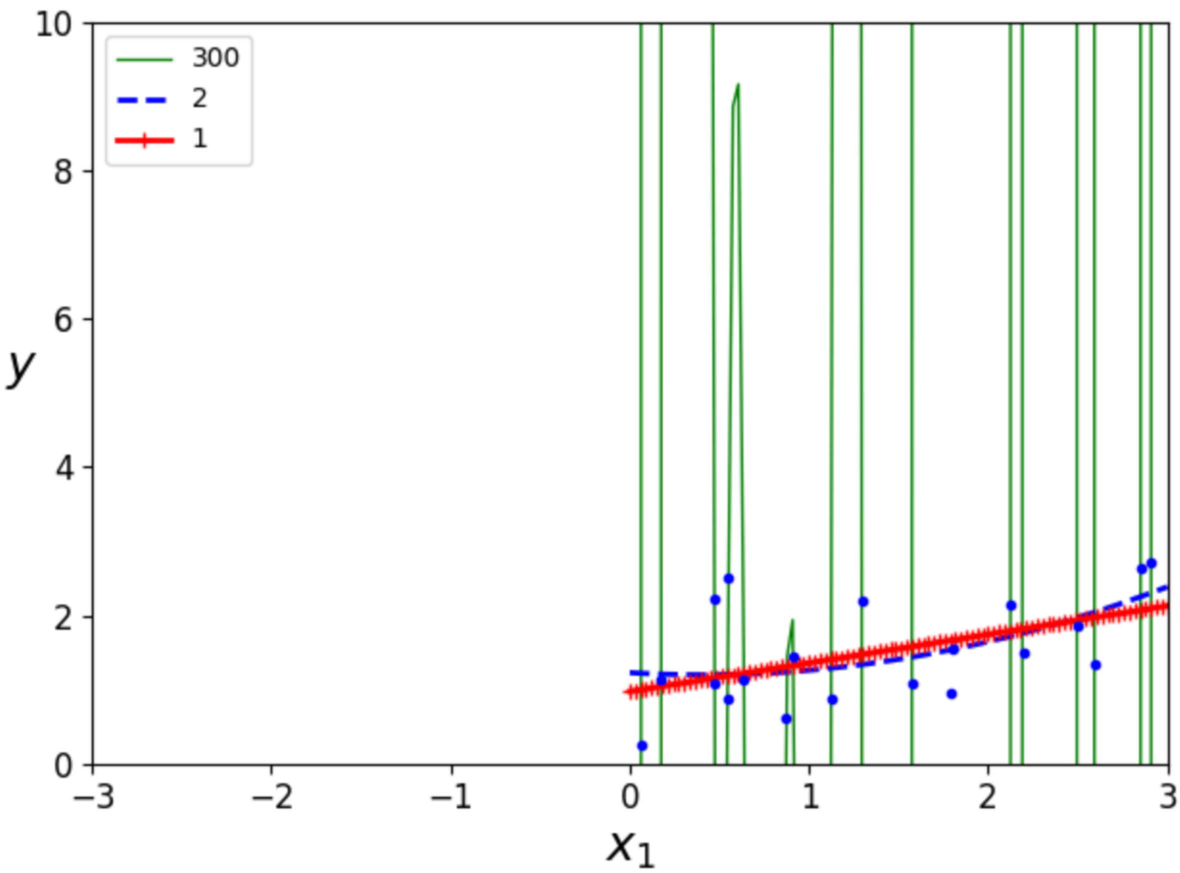
2차 다항식으로 생성된 데이터셋에 대해 10차 다항 회귀를 적용한 선형 회귀 모델의 학습 곡선은 다음과 같으며, 전형적인 과대 적합의 양태를 잘 보여준다.

훈련 데이터(빨강)에 대한 성능: 훈련 데이터에 대한 평균 제곱근 오차가 매우 낮음.

검증 데이터(파랑)에 대한 성능: 훈련 데이터에 대한 성능과 차이가 크게 벌어짐. 과대적합 모델 개선법: 훈련 데이터 추가

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# 세 개의 다항 회귀 모델 지정: 차례 대로 300차 다항 회귀, 2차 다항 회귀, 1차 선형 회귀 모델의 예측값 그래프 그리기
for style, width, degree in (("g-", 1, 300), ("b--", 2, 2), ("r-+", 2, 1)):

    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)  # 다항 특성 변환기
    std_scaler = StandardScaler()                                             # 표준화 축척 조정
    lin_reg = LinearRegression()                                              # 선형 회귀 모델

    polynomial_regression = Pipeline([                              # 파이프라인: 전처리 + 선형 회귀 모델
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ])

    polynomial_regression.fit(X, y)                                            # 훈련
    y_newbig = polynomial_regression.predict(X_new)                            # 예측

    plt.plot(X_new, y_newbig, style, label=str(degree), linewidth=width)       # 그래프 그리기
```

```
plt.plot(X, y, "b.", linewidth=3)                                          # 원 데이터 산점도
plt.legend(loc="upper left")
plt.xlabel("
", fontsize=18)
plt.ylabel("
", rotation=0, fontsize=18)
plt.axis([-3, 3, 0, 10])
save_fig("high_degree_polynomials_plot")
plt.show()
```



5. 규제가 있는 선형 모델

- > 규제를 통해 과대적합을 방지함
- 릿지회귀
- 라쏘회귀
- 엘라스틱회귀

릿지회귀

```
np.random.seed(42)

m = 20
```

```
X = 3 * np.random.rand(m, 1)
y = 1 + 0.5 * X + np.random.randn(m, 1) / 1.5     # 1차 선형회귀 모델을 따르도록 함. 단, 잡음 추가됨.
X_new = np.linspace(0, 3, 100).reshape(100, 1)    # 0~3 구간에서 균등하게 100개의 검증 데이터 선택


from sklearn.linear_model import Ridge

def plot_model(model_class, polynomial, alphas, **model_kargs):
    for alpha, style in zip(alphas, ("b-", "g--", "r:")):
        model = model_class(alpha, **model_kargs) if alpha > 0 else LinearRegression()
        if polynomial:
            model = Pipeline([
                    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
                    ("std_scaler", StandardScaler()),           # 표준화 축척 조정
                    ("regul_reg", model),
                ])
        model.fit(X, y)
        y_new_regul = model.predict(X_new)
        lw = 2 if alpha > 0 else 1
        plt.plot(X_new, y_new_regul, style, linewidth=lw, label=r"
".format(alpha))
    plt.plot(X, y, "b.", linewidth=3)
    plt.legend(loc="upper left", fontsize=15)
    plt.xlabel("
", fontsize=18)
    plt.axis([0, 3, 0, 4])

plt.figure(figsize=(8,4))
plt.subplot(121)
plot_model(Ridge, polynomial=False, alphas=(0, 10, 100), random_state=42)
plt.ylabel("
", rotation=0, fontsize=18)
plt.subplot(122)
plot_model(Ridge, polynomial=True, alphas=(0, 10**-5, 1), random_state=42)

save_fig("ridge_regression_plot")
plt.show()
```

라쏘회귀

```
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X, y)
lasso_reg.predict([[1.5]])
```

엘라스틱 넷

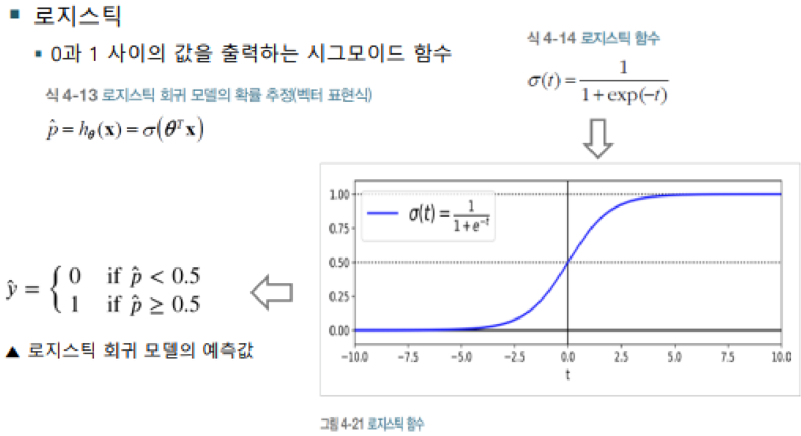
```
from sklearn.linear_model import ElasticNet
elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=42)
elastic_net.fit(X, y)
elastic_net.predict([[1.5]])
```

6.로지스틱 회귀

다른점 : 선형회귀처럼 바로 결과를 출력하지 않고 결괏값의 로지스틱을 출력(S자 형태의 시그모이드 함수)

- 확률 모델로서 독립변수의 선형 결합을 이용하여 사건의 발생 가능성을 예측하는데 사용되는 통계 기법
- 0.5를 기준으로 1과 0을 결정

6.1확률 추정



시그모이드 함수는 왼쪽위 함수의 t값을 오른쪽 위 함수의 t값에 넣은 모양

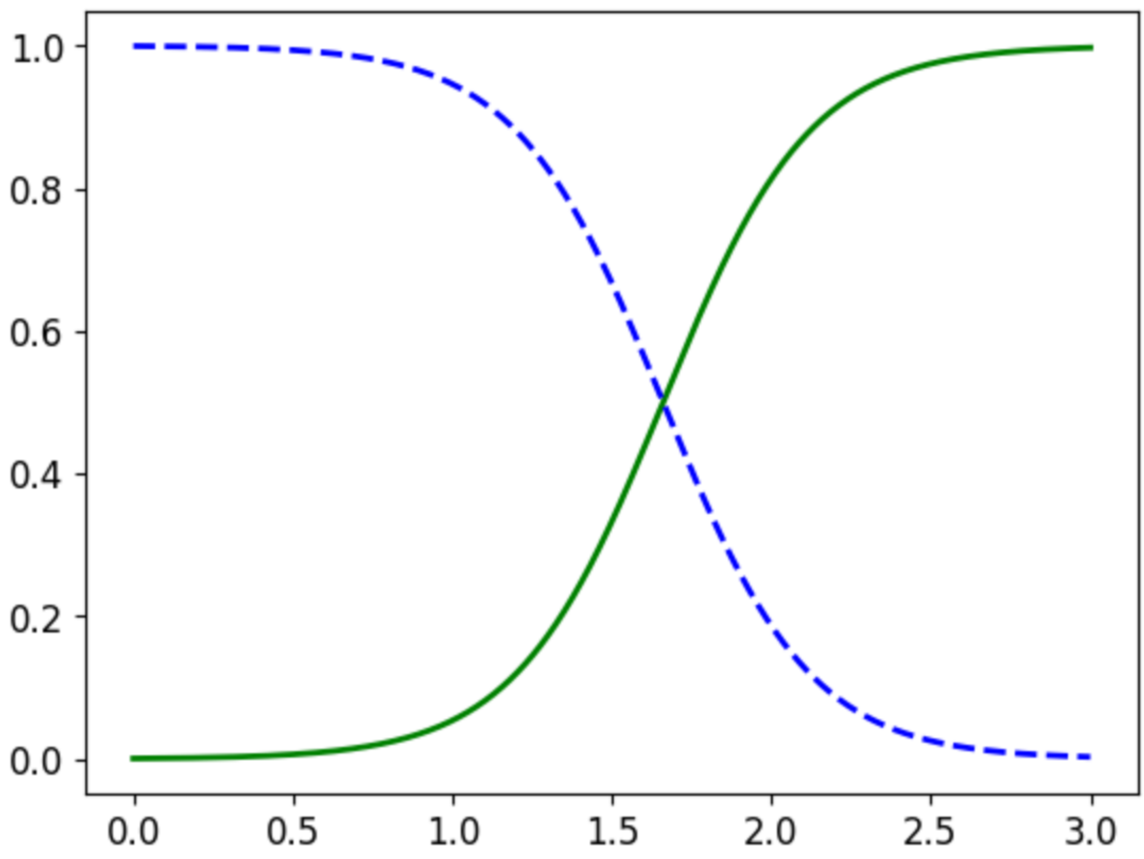
6.2훈련과 비용 함수

-비용함수

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{p}^{(i)}\right)\right]$$

6.3결정 경계(붓꽃 데이터 예제)

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

X = iris["data"][:, 3:]                  # 1개의 특성(꽃잎 너비)만 사용
y = (iris["target"] == 2).astype(np.int)  # 버지니카(Virginica) 품종일 때 1(양성)

from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="lbfgs", random_state=42)
log_reg.fit(X, y)

X_new = np.linspace(0, 3, 1000).reshape(1000, 1)
y_proba = log_reg.predict_proba(X_new)

plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2, label="Not Iris virginica")
```



소프트맥스 회귀

- 다중 클래스 분류를 지원하도록 한 회귀 모델
- 다항 로지스틱 회귀라고도 불림

```
X = iris["data"][:, (2, 3)]  # 꽃잎 길이, 꽃잎 너비
y = iris["target"]

softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10, random_state=42)
softmax_reg.fit(X, y)
```

```
x0, x1 = np.meshgrid(
        np.linspace(0, 8, 500).reshape(-1, 1),
        np.linspace(0, 3.5, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]


y_proba = softmax_reg.predict_proba(X_new)
y_predict = softmax_reg.predict(X_new)

zz1 = y_proba[:, 1].reshape(x0.shape)
zz = y_predict.reshape(x0.shape)

plt.figure(figsize=(10, 4))
```

```
plt.plot(X[y==2, 0], X[y==2, 1], "g^", label="Iris virginica")
plt.plot(X[y==1, 0], X[y==1, 1], "bs", label="Iris versicolor")
plt.plot(X[y==0, 0], X[y==0, 1], "yo", label="Iris setosa")

from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])

plt.contourf(x0, x1, zz, cmap=custom_cmap)
contour = plt.contour(x0, x1, zz1, cmap=plt.cm.brg)
plt.clabel(contour, inline=1, fontsize=12)
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.legend(loc="center left", fontsize=14)
plt.axis([0, 7, 0, 3.5])
save_fig("softmax_regression_contour_plot")
plt.show()
```