Hands-On Machin Learning With Scikit-Learn, Keras & TensorFlow

PAPATA LABS

Index

규제가 있는 선형 모델

과대적합을 감소시키는 좋은 방법은 모델을 규제하는 것이다.

규제란 모델의 자유도를 줄여 데이터를 과하게 학습하지 못하게 하는 것

일반적으로
다항 회귀 모델의 경우 다항식의 차수를 감소하는 경우가 많고
선형 회귀 모델에서는 가중치를 제한함으로써 규제를 가한다+

01 규제가 있는 선형 모델

릿지 회귀
(Lidge Regression)

- 선형 회귀에서 규제항을 비용함수에 추가하여 연산
- 모델의 가중치가 가능한 한 작게 유지되도록 한다

- 모델의 훈련이 끝나면 모델의 성능을 규제가 없는 성능 지표로 평가한다

$$J(\theta) = \text{MSE}(\theta) + \alpha\frac{1}{2}\sum_{i=1}^{n}\theta_i^2$$

라쏘 회귀
(Lasso Regression)

- 선형 회귀에서 규제항을 비용함수에 추가하여 연산
- 덜 중요한 특성의 가중치를 제거하기위해 사용한다

- 절대값의 특성상 θ가 0인 경우 미분이 불가능하기 때문에 그러한 경우 서브그레이디언트 벡터를 사용한다

$$J(\theta) = \text{MSE}(\theta) + \alpha\sum_{i=1}^{n}|\theta_i|$$

엘라스틱넷
(elastic net)

- 릿지 회귀와 라쏘 회귀를 절충한 모델
- 혼합 비율 r을 사용해 조절

$$J(\theta) = \text{MSE}(\theta) + r\alpha\sum_{i=1}^{n}|\theta_i| + \frac{1-r}{2}\alpha\sum_{i=1}^{n}\theta_i^2$$

릿지 회귀

- 일반적인 비용 함수에 규제항 $\alpha \sum_{i=1}^{n} {\theta_i}^2$ 이 추가된 형태
- 학습 알고리즘을 데이터에 맞추는 것 뿐만 아니라 모델의 가중치를 가능한 한 작게 유지되도록 하여 규제를 가한다
- 규제항은 훈련하는 동안에만 비용 함수에 추가되고, 훈련이 끝난 후 성능을 평가할 때에는 규제항 없는 성능 지표로 평가한다

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

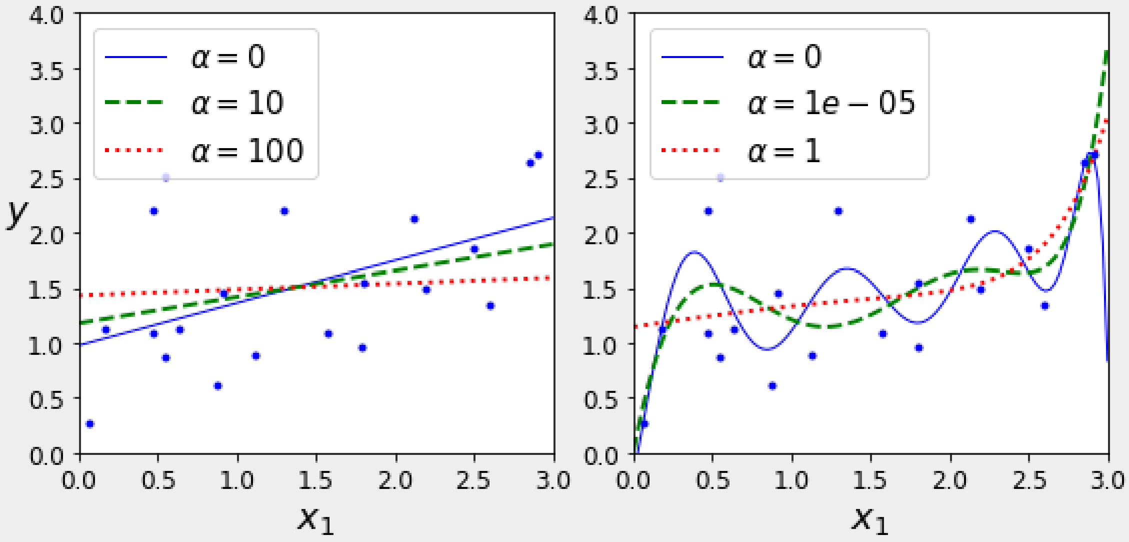
01 규제가 있는 선형 모델

릿지 회귀

```
[ ] 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()),
                        ("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"$\alpha = {}$".format(alpha))
        plt.plot(X, y, "b.", linewidth=3)
        plt.legend(loc="upper left", fontsize=15)
        plt.xlabel("$x_1$", 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("$y$", 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()
```

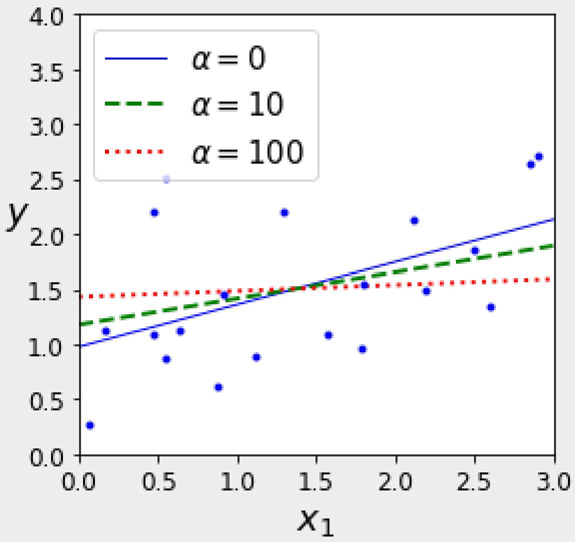
01 규제가 있는 선형 모델

릿지 회귀



01 규제가 있는 선형 모델

릿지 회귀

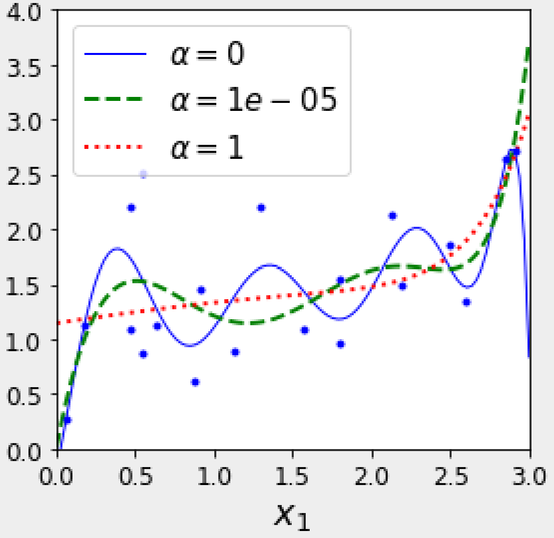



```
[72] for i in [0, 0.1, 1]:
       model = Ridge(i, random_state=42)

       model.fit(X,y)
       y_test2 = model.predict(X_new)

       print("alpha : " + str(i) + "₩ncoef_ : " + str(model.coef_) + "₩n")
```

```
alpha : 0
coef_ : [[0.3852145]]

alpha : 0.1
coef_ : [[0.3828496]]

alpha : 1
coef_ : [[0.36280369]]
```

01 규제가 있는 선형 모델

릿지 회귀




```
[70] for i in [0, 10**-7, 1]:
       model = Ridge(i, random_state=42)
       model = Pipeline([
                           ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
                           ("std_scaler", StandardScaler()),
                           ("regul_reg", model),
                       ])

       model.fit(X,y)
       y_test = model.predict(X_new)

       print("alpha : " + str(i) + "₩ncoef_ : " + str( model.named_steps["regul_reg"].coef_) + "₩n")

     alpha : 0
     coef_ : [[ 6.44765722e+00  2.54287968e+01 -2.71890613e+02 -1.92326286e+03
        2.12237849e+04 -7.56345999e+04  1.38131957e+05 -1.39417240e+05
        7.40451434e+04 -1.61852297e+04]]

     alpha : 1e-07
     coef_ : [[  15.48307621 -121.89219824  367.47125404 -445.09958471   20.20836729
        314.70823129    4.97016909 -222.60748709   -0.69815307   67.997637  ]]

     alpha : 1
     coef_ : [[ 0.19242496 -0.05219979 -0.05187216 -0.02809326 -0.00486859  0.01898063
        0.04493381  0.07311     0.10293252  0.13360102]]
```

01 규제가 있는 선형 모델

릿지 회귀

$$MSE(\theta) = \frac{1}{n}\sum_{i=1}^{n}(\theta x - y)^2$$

$$= y^T y - 2\theta^T \cdot x^T \cdot y + \theta^T X^T X \theta$$

$$Lidge = MSE + \alpha \sum_{i=1}^{n} \theta^2$$

$$+ \alpha \theta^2$$

$$\frac{\partial}{\partial \theta} Lidge = \frac{\partial}{\partial \theta} MSE + \frac{\partial}{\partial \theta} \alpha \theta^2$$

$$\frac{\partial}{\partial \theta} Lidge = \frac{\partial}{\partial \theta} MSE + \frac{\partial}{\partial \theta} \alpha \theta^2 = 0$$

$$= -2X^T y + 2X^T X\theta + 2\alpha A \theta$$

$$\Rightarrow 2(X^T X + \alpha A)\theta = 2X^T y$$

$$\theta = (X^T X + \alpha A)^{-1} X^T y$$

01 규제가 있는 선형 모델

릿지 회귀 - 정규방정식

$$\theta = (X^T X + \alpha A)^{-1} X^T y$$

```
[ ]  from sklearn.linear_model import Ridge
     ridge_reg = Ridge(alpha=1, solver="cholesky", random_state=42)
     ridge_reg.fit(X, y)
     ridge_reg.predict([[1.5]])
```

```
array([[1.55071465]])
```

01 규제가 있는 선형 모델

릿지 회귀 - 정규 방정식

```
[ ] sgd_reg = SGDRegressor(penalty="l2", max_iter=1000, tol=1e-3, random_state=42)
    sgd_reg.fit(X, y.ravel())
    sgd_reg.predict([[1.5]])

    array([1.47012588])
```

라쏘 회귀

- 일반적인 비용 함수에 규제항 $\alpha \sum_{i=1}^{n} |\theta_i|$ 이 추가된 형태
- 덜 중요한 특성의 가중치를 제거하기 위해 사용
- 다시 말해 라쏘 회귀는 자동으로 특성 선택을 하고 희소 모델을 만든다.

$$J(\theta) = \mathrm{MSE}(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

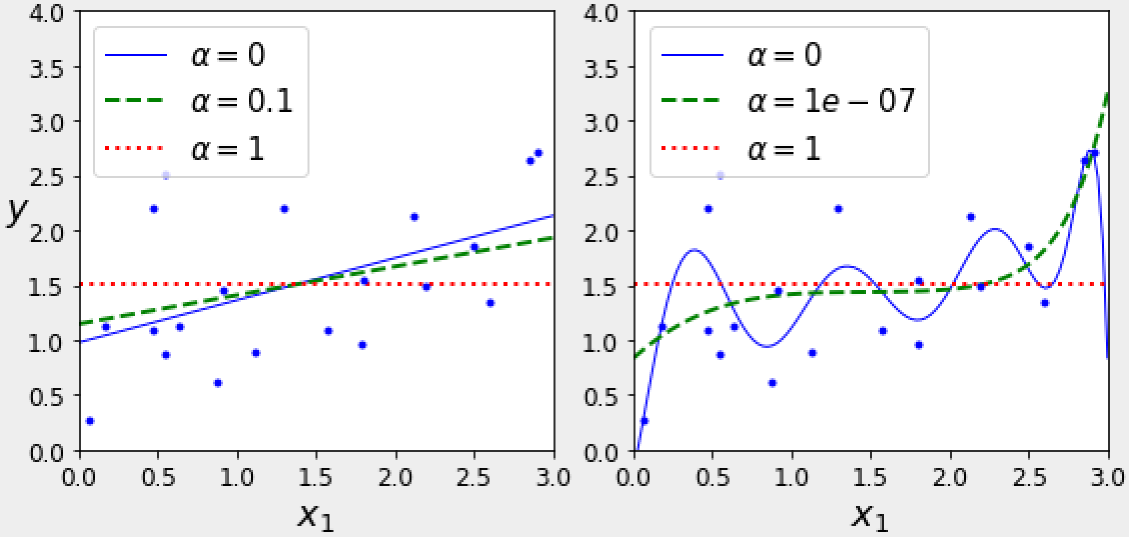
01 규제가 있는 선형 모델

라쏘 회귀

```
[38] from sklearn.linear_model import Lasso
     from sklearn.linear_model import LinearRegression

     plt.figure(figsize=(8,4))
     plt.subplot(121)
     plot_model(Lasso, polynomial=False, alphas=(0, 0.1, 1), random_state=42)
     plt.ylabel("$y$", rotation=0, fontsize=18)
     plt.subplot(122)
     plot_model(Lasso, polynomial=True, alphas=(0, 10**-7, 1), random_state=42)

     plt.show()
```

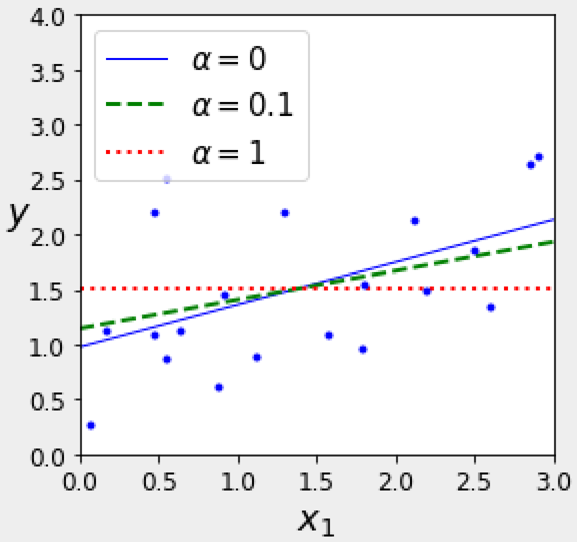
01 규제가 있는 선형 모델

라쏘 회귀



01 규제가 있는 선형 모델

라쏘 회귀

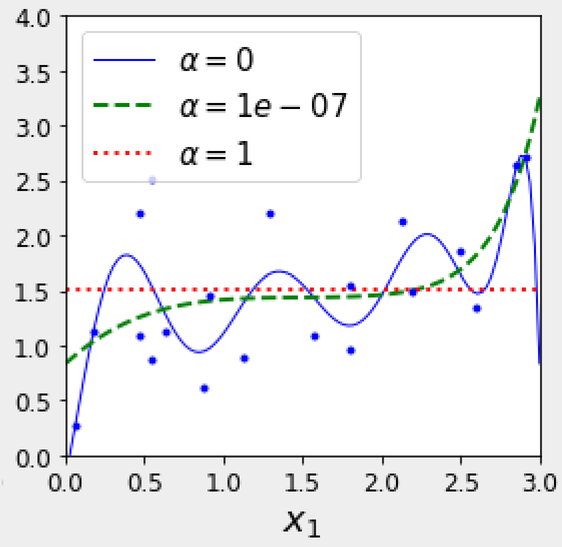



```
[65] for i in [0, 0.1, 1]:
       model = Lasso(i, random_state=42)

       model.fit(X,y)
       y_test2 = model.predict(X_new)

       print("alpha : " + str(i) + "₩ncoef_ : " + str(model.coef_) + "₩n")
```

```
alpha : 0
coef_ : [0.3852145]

alpha : 0.1
coef_ : [0.26167212]

alpha : 1
coef_ : [0.]
```

01 규제가 있는 선형 모델

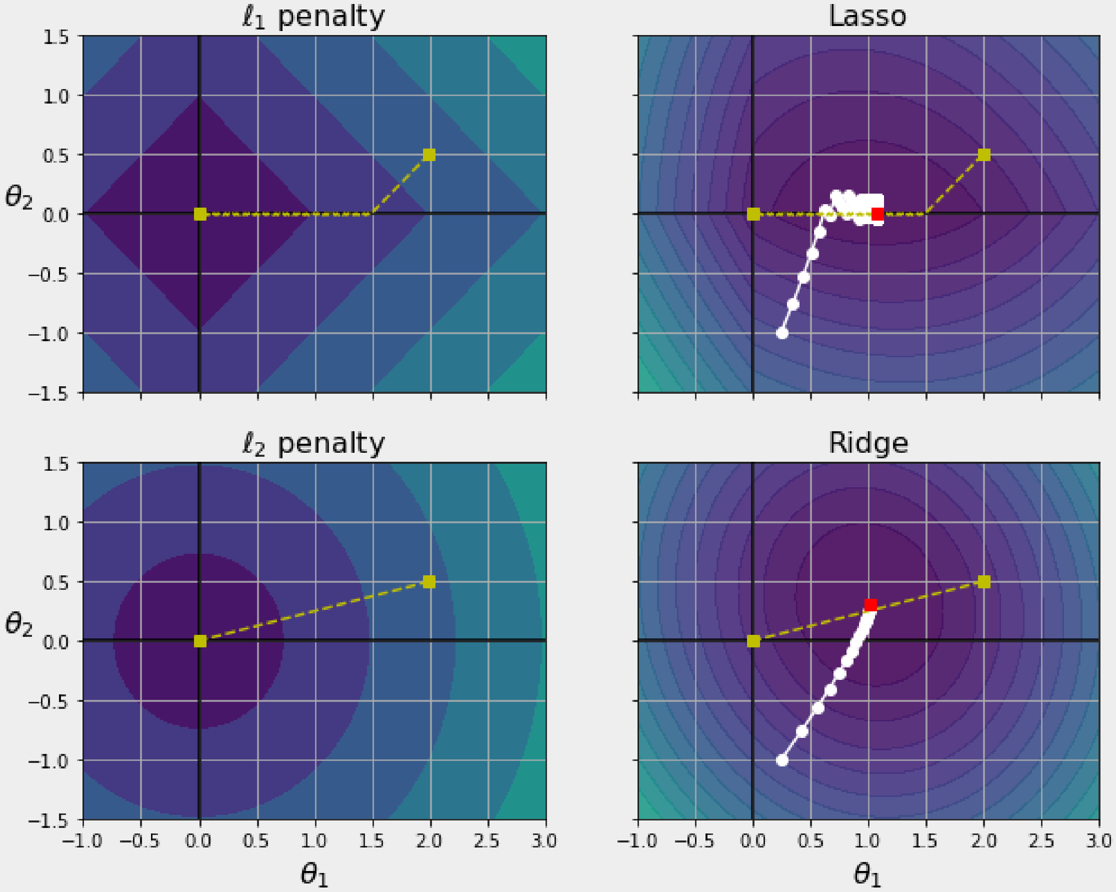
라쏘 회귀




```
[64] for i in [0, 10**-7, 1]:
       model = Lasso(i, random_state=42)
       model = Pipeline([
                           ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
                           ("std_scaler", StandardScaler()),
                           ("regul_reg", model),
                       ])

       model.fit(X,y)
       y_test = model.predict(X_new)

       print("alpha : " + str(i) + "₩ncoef_ : " + str( model.named_steps["regul_reg"].coef_) + "₩n")

     alpha : 0
     coef_ : [ 1.17874780e+00 -2.71604828e+00  1.88203196e+00  4.73570759e-01
      -3.35653044e-01 -4.49644855e-01 -2.64347184e-01 -3.94330591e-04
       2.48958583e-01  4.50299975e-01]

     alpha : 1e-07
     coef_ : [ 1.17872866e+00 -2.71592491e+00  1.88182074e+00  4.73568160e-01
      -3.35485451e-01 -4.49622249e-01 -2.64346131e-01 -5.14028619e-04
       2.48977337e-01  4.50318743e-01]

     alpha : 1
     coef_ : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

01 규제가 있는 선형 모델

라쏘 회귀



엘라스틱 넷

- 릿지 회귀와 라쏘 회귀를 절충한 모델
- 일반적인 비용 함수에 규제항 $\mathrm{r}\alpha\sum_{i=1}^{n}\theta_i{}^2 + \frac{1-r}{2}r\alpha\sum_{i=1}^{n}|\theta_i|$ 이 추가된 형태
- 혼합 정도는 혼합 비율 r을 이용하여 조절
- 일반적으로 릿지 회기를 사용하지만

  쓰이는 특성이 몇 개 뿐이라고 의심되면 라쏘 회귀나 엘라스틱 넷을 사용한다.

$$J(\theta) = \mathrm{MSE}(\theta) + r\alpha\sum_{i=1}^{n}|\theta_i| + \frac{1-r}{2}\alpha\sum_{i=1}^{n}\theta_i^2$$

01 규제가 있는 선형 모델

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]])

     array([1.54333232])
```

로지스틱 회귀 Logistic Regression

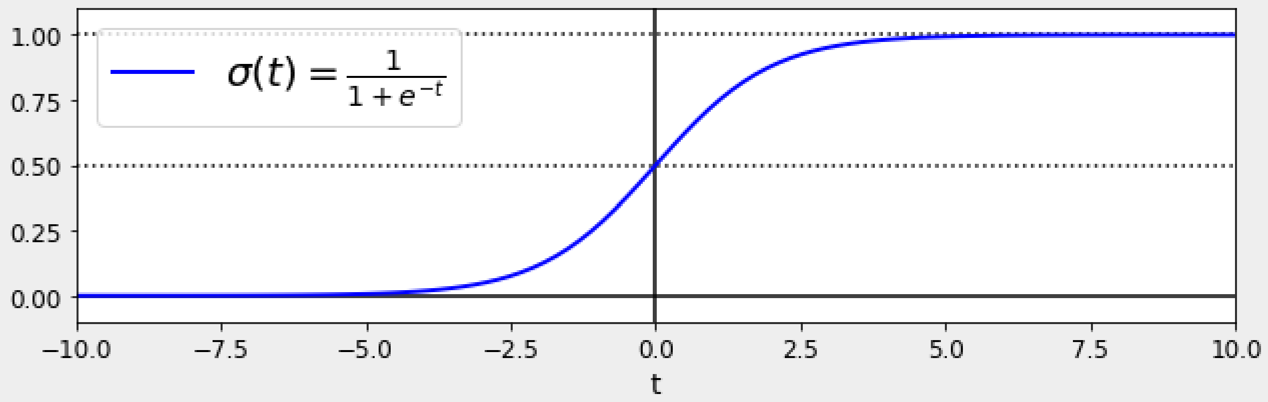
- 샘플이 특정 클래스에 속할 확률을 추정하는 데 사용됨
- 추정 확률이 50%가 넘으면 모델은
  그 샘플이 해당 클래스에 속한다고 예측,
  아니면 클래스에 속하지 않는다고 예측하는 이진 분류기
- 로지스틱 회귀는 가중치 합을 바로 출력하지 않고 그 로지스틱 확률을 출력함

02 로지스틱 회귀

로지스틱 회귀 모델의 확률 추정

$$\hat{p} = h_\theta(x) = \sigma(\theta^T x)$$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



하나의 훈련 샘플에 대한 비용 함수

$$c(\theta) = \begin{cases} -\log(\hat{p}) & if\ y = 1, \\ -\log(1-\hat{p}) & if\ y = 0. \end{cases}$$

로지스틱 회귀 비용 함수

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

로지스틱 회귀 비용 함수의 편도함수

$$\frac{\partial}{\partial \theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sigma(\theta^T x^i) - y^i)\, x_j{}^i$$

02 로지스틱 회귀

붓꽃

```
[ ]  from sklearn import datasets
     iris = datasets.load_iris()
     list(iris.keys())

     ['data',
      'target',
      'frame',
      'target_names',
      'DESCR',
      'feature_names',
      'filename',
      'data_module']

[ ]  print(iris.DESCR)

     .. _iris_dataset:

     Iris plants dataset
     --------------------

     **Data Set Characteristics:**

         :Number of Instances: 150 (50 in each of three classes)
         :Number of Attributes: 4 numeric, predictive attributes and the class
         :Attribute Information:
             - sepal length in cm
             - sepal width in cm
             - petal length in cm
             - petal width in cm
             - class:
                     - Iris-Setosa
                     - Iris-Versicolour
                     - Iris-Virginica
```

02 로지스틱 회귀

붓꽃

02 로지스틱 회귀

붓꽃


```
[ ] from sklearn import datasets
    iris = datasets.load_iris()
    list(iris.keys())

    ['data',
     'target',
     'frame',
     'target_names',
     'DESCR',
     'feature_names',
     'filename',
     'data_module']

[ ] print(iris.DESCR)

    .. _iris_dataset:

    Iris plants dataset
    --------------------

    **Data Set Characteristics:**

        :Number of Instances: 150 (50 in each of three classes)
        :Number of Attributes: 4 numeric, predictive attributes and the class
        :Attribute Information:
            - sepal length in cm
            - sepal width in cm
            - petal length in cm
            - petal width in cm
            - class:
                    - Iris-Setosa
                    - Iris-Versicolour
                    - Iris-Virginica
```

02 로지스틱 회귀

붓꽃

```
[ ] X = iris["data"][:, 3:]
    y = (iris["target"] == 2).astype(int)
```

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

```
LogisticRegression(random_state=42)
```

```
Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
```

02 로지스틱 회귀

붓꽃

```
[ ] X = iris["data"][:, 3:]
    y = (iris["target"] == 2).astype(int)
```

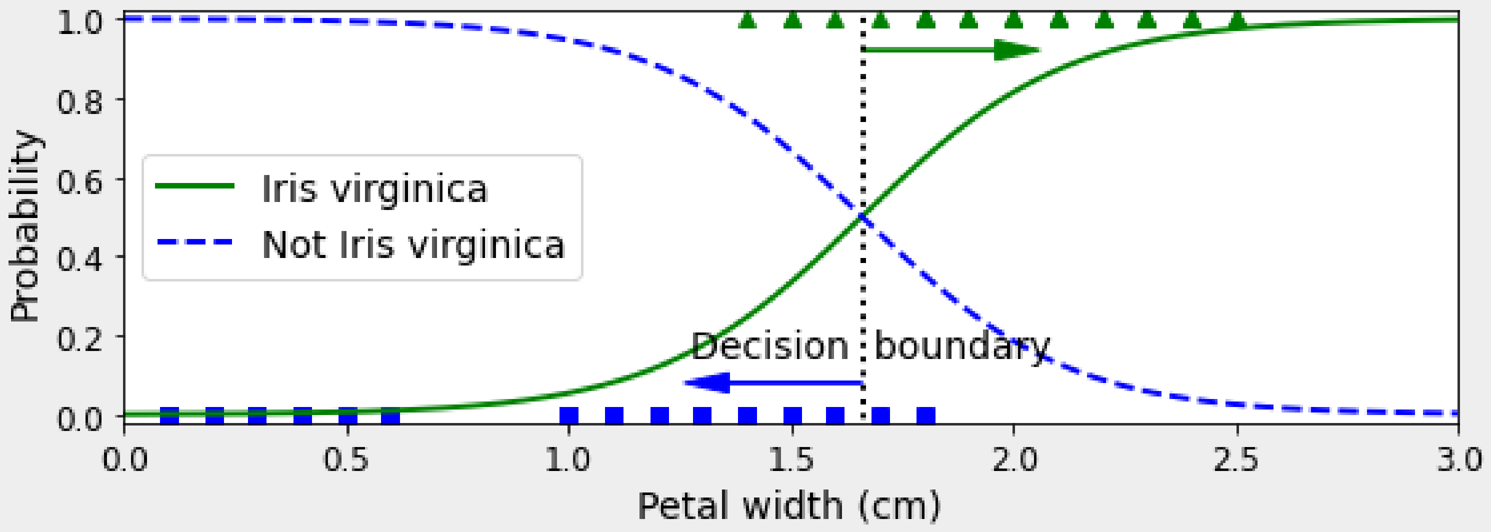
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="lbfgs", random_state=42)
log_reg.fit(X, y)
```

```
LogisticRegression(random_state=42)
```

```
[ ] X_new = np.linspace(0, 3, 1000).reshape(-1, 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")
```

02 로지스틱 회귀

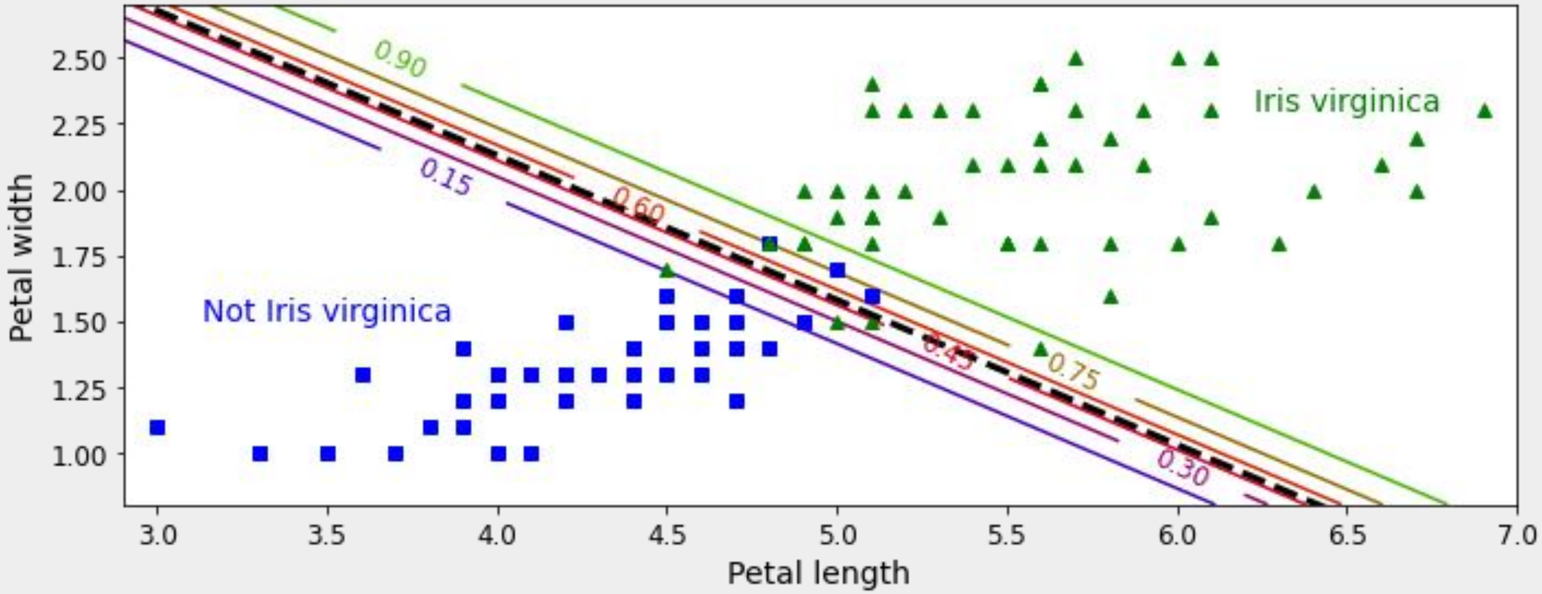
붓꽃



1.0
0.8
0.6
0.4
0.2
0.0
Probability
Iris virginica
Not Iris virginica
Decision boundary
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Petal width (cm)

02 로지스틱 회귀

붓꽃



Iris virginica
Not Iris virginica
0.90
0.75
0.60
0.45
0.30
0.15
Petal width
Petal length
2.50
2.25
2.00
1.75
1.50
1.25
1.00
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0

소프트맥스 회귀 SoftMax Regression

- 다중 분류기는 여러 개의 이진 분류기를 합칠 수도 있지만
  기존의 이중 분류기를 직접 다중 클래스를 지원하도록 일반화할 수 있다

- 주어진 샘플 x에 대해 각 클래스 k에 대한 점수를 계산하고,
  그 점수에 소프트맥스 함수를 적용하여 각 클래스의 확률을 추정

02 로지스틱 회귀

소프트맥스 회귀

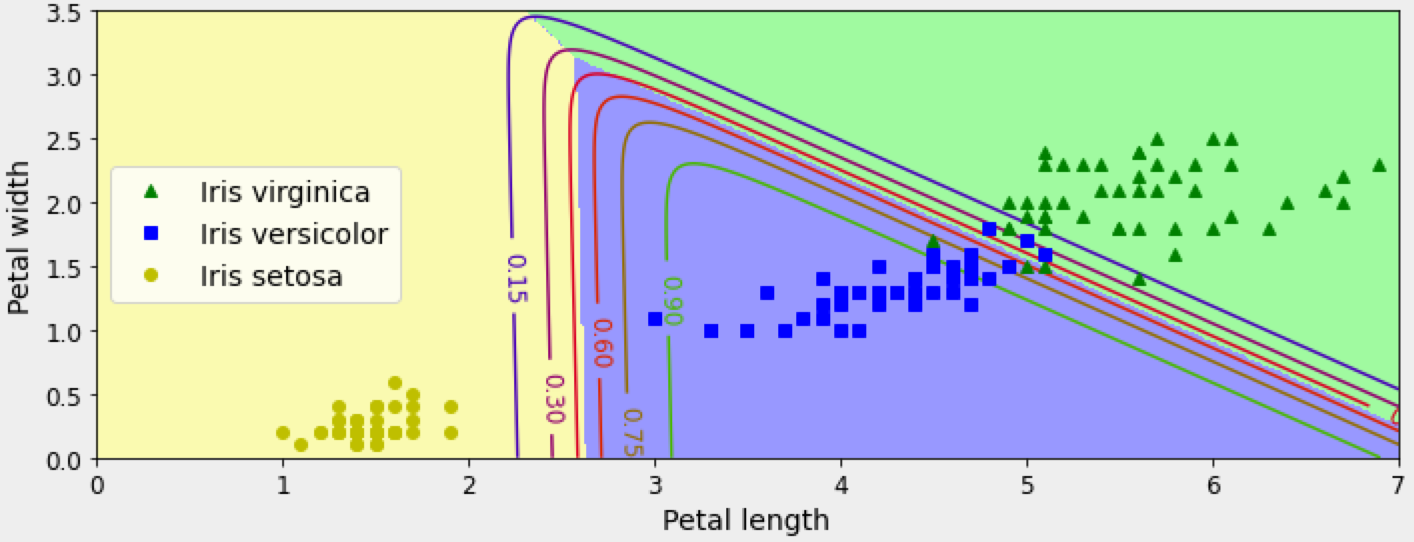
$$softmax(z) = [\frac{e^{z1}}{\sum_{j=1}^{3} e^{zj}} \ \frac{e^{z2}}{\sum_{j=1}^{3} e^{zj}} \ \frac{e^{z3}}{\sum_{j=1}^{3} e^{zj}}] = [p_1, p_2, p_3] = \hat{y} = \text{예측값}$$

$$softmax(z) = [\frac{e^{z1}}{\sum_{j=1}^{3} e^{zj}} \; \frac{e^{z2}}{\sum_{j=1}^{3} e^{zj}} \; \frac{e^{z3}}{\sum_{j=1}^{3} e^{zj}}] = [p_1, p_2, p_3] = \hat{y} = \text{예측값}$$

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$$

02 로지스틱 회귀

소프트맥스 회귀



END OF DOCUMENT

THANK YOU

PAPATA LABS