동국대학교
dongguk university

Logistic Regression

Machine Learning

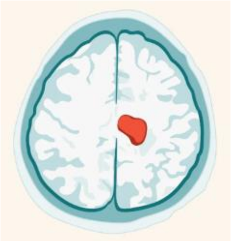
(PM chap 2, chap 3)

Classification

- From x → predict y (class)

x

Size of tumor is 0.8 cm



→

y

cancer ?

Yes / No

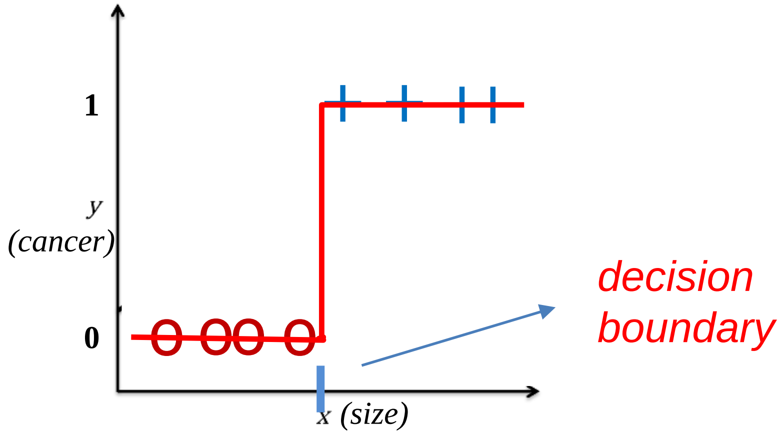
Classification

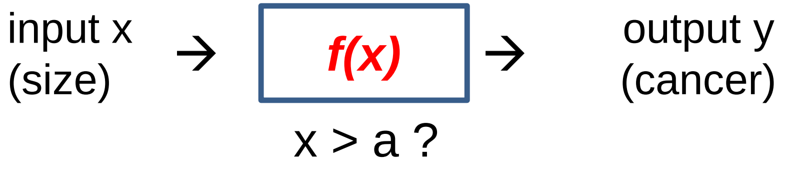
- Predicting class y
  - From x (input value) → predict y (class/category, discrete value)
  - *Logistic Regression*: constructing a model ***y = f(x)*** , using training data

		class label
x		y
size		cancer
0.75		1 (yes)
0.82		1 (yes)
0.26		0 (no)
0.54		0 (no)
…	…	…

0.43 → cancer ?

model

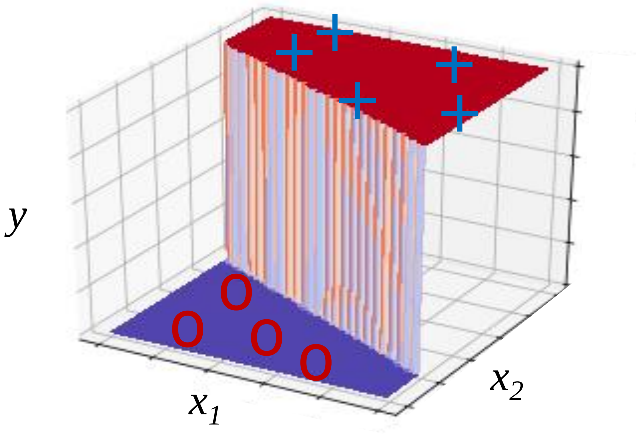



input x
(size)
→
f(x)
→
output y
(cancer)
x > a ?

Classification

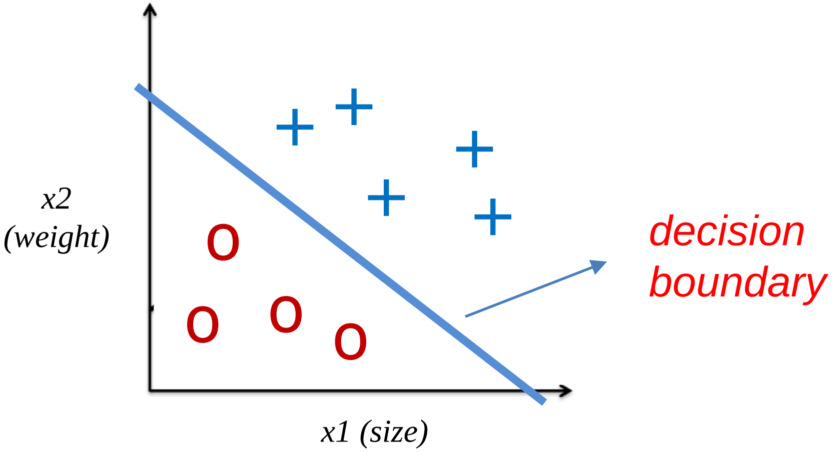
x		class label y
size	weight	cancer
0.75	0.9	1 (yes)
0.82	0.8	1 (yes)
0.26	0.4	0 (no)
0.54	0.7	0 (no)
…	…	…

model



input x_1, x_2 (size, weight) → $\boxed{f(x)}$ → output y (Yes/No)

$$a \cdot x_1 + b \cdot x_2 + c > 0 \text{ ?}$$

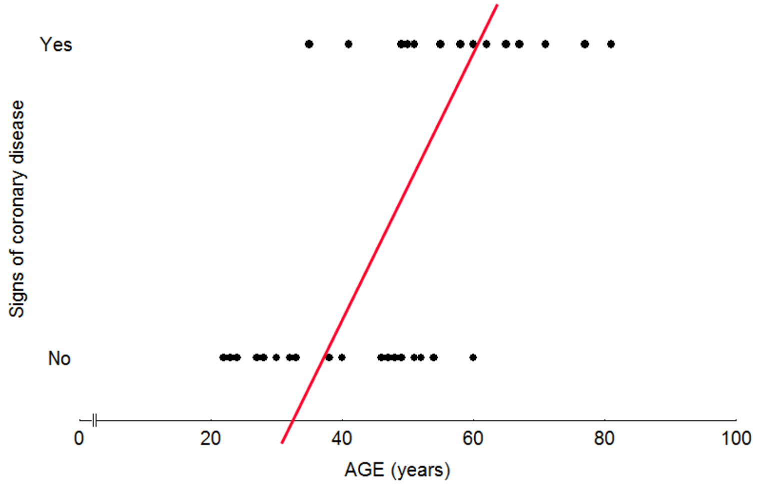


Logistic Regression

- Example
  - Relationship between Age and signs of coronary heart disease (CD)

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

  - linear regression is not appropriate

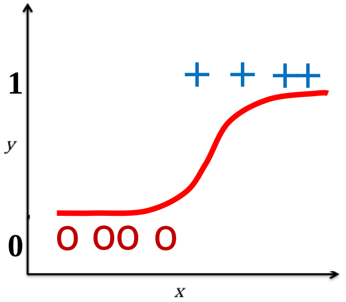


Logistic Regression

- The model
  - Build a model that predicts the *probability of y*
  - Logit(p) = f(x) is modeled as a linear function

$$logit(p) = f_{\theta}(x) = w\cdot x + b$$

parameters
(*w*, *b*)



x	y
0.75	1
0.82	1
0.26	0
0.54	0
…	…

- *w* and *b* are estimated using the training data $(x^{(i)}, y^{(i)})$
  - *w* : weight (coefficient)
  - *b* : bias (intercept)

  } θ : parameters

- Minimize prediction error → find *w* and *b*
  - Gradient descent

Logistic Regression

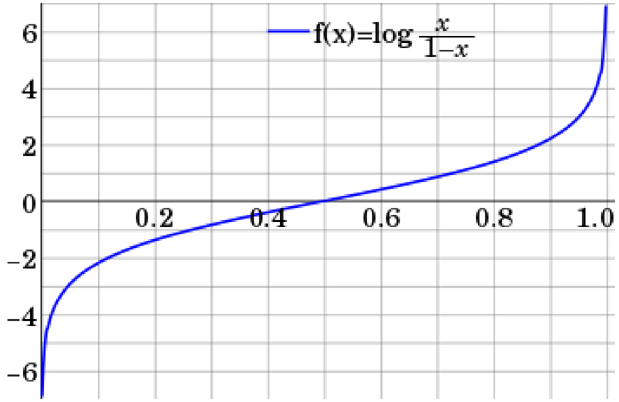
- Predicting probability of y
  - y is usually coded as 1 (true) or 0 (false)
  - Predict prob. of y = 1 **p** $[0, 1]$ **= w x + b** [-∞, ∞] → not appropriate
- Odds ratio and Logit
  - The ***odds*** : the ratio of the probability of success to the probability of failure

$$Odds = \frac{p(y=1|x)}{p(y=0|x)} \qquad [\mathbf{0}, \infty]$$

  - Logit : log odds

$$Logit(p) = \ln(Odds) = \ln\left(\frac{p}{1-p}\right)$$

$[-\infty, \infty]$



Logistic Regression

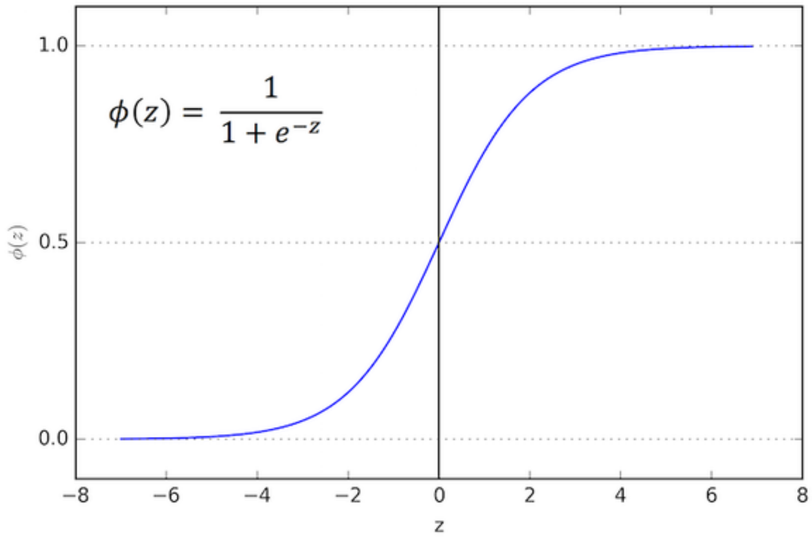
- Logistic function
  - Solving for P

$$\ln\left(\frac{p}{1-p}\right) = wx + b$$

$$\Rightarrow \quad \frac{p}{1-p} = e^{wx+b}$$

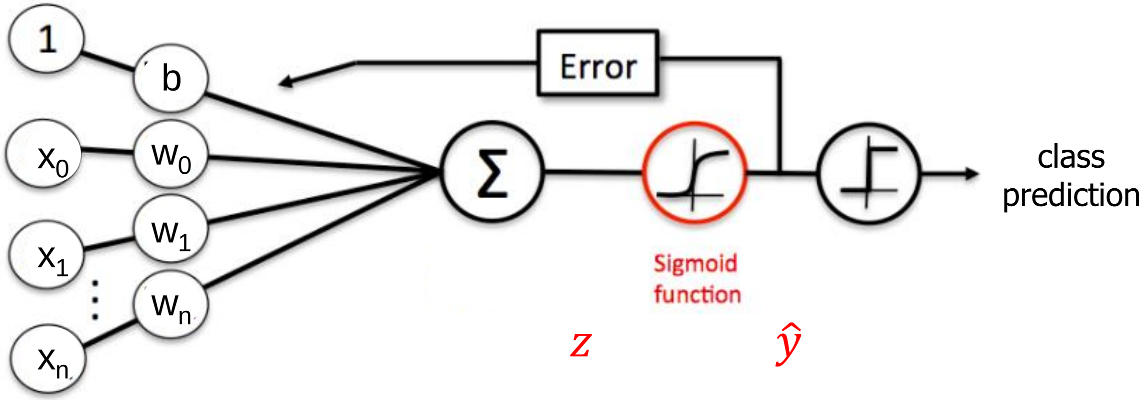
$$\Rightarrow \quad p = \frac{1}{1+e^{-(wx+b)}}$$

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



Logistic Regression

- The model
  - Output function to predict y : sigmoid



$$z = w_0 x_0 + w_1 x_1 + \cdots + b$$

$$\hat{y} = \phi(z) = \frac{1}{1 + e^{-z}} \qquad class = \begin{cases} 1 & if \quad \hat{y} \geq 0.5 \\ 0 & otherwise \end{cases}$$

Logistic Regression

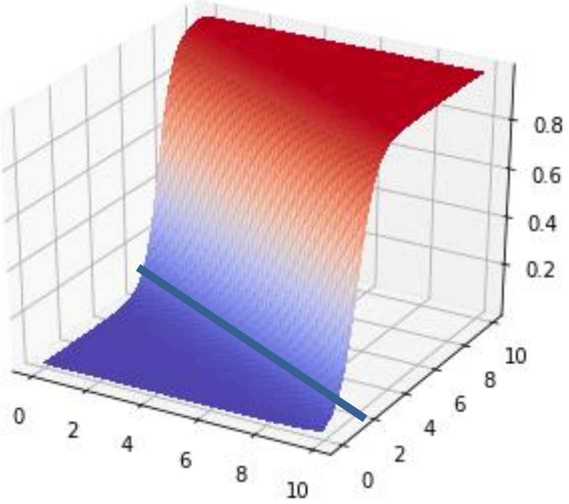
- Example

$$\mathbf{x} = (x_0, x_1) \qquad \mathbf{w} = (0.1, 0.2) \qquad b = -1.4$$

  - Decision boundary

$$class = 1 \quad if \quad \hat{y} = \frac{1}{1 + e^{-(0.1x_1 + 0.2x_2 - 1.4)}} \geq 0.5$$

$$class = 1 \quad if \quad z = (0.1x_1 + 0.2x_2 - 1.4) \geq 0$$



Estimating Parameters

- Maximum Likelihood
  - For observed data D, model parameter θ, the likelihood $L(\theta)$ = $p(D \mid \theta)$
  - Maximum likelihood estimation (MLE) :

$$\boxed{\hat{\theta} = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} p(D \mid \theta)}$$

  - For i.i.d (independent and identical distributed) data $D = \{d^{(1)}, d^{(2)}, \dots\}$

$$\hat{\theta} = \arg\max_{\theta} p(D \mid \theta) = \arg\max_{\theta} \prod p(d^{(i)} | \theta)$$

Estimating Parameters

- Maximum likelihood for logistic regression
  - The model : $\theta = (w_0, w_1, b)$

    $$z = w_0 x_0 + w_1 x_1 + b \qquad p = \hat{y} = \frac{1}{1+e^{-z}} \quad (P(y=1|x))$$

  - For *n* sample data

    $$p(D\,|\theta) = \prod \left(p^{(i)}\right)^{y^{(i)}} \left(1 - p^{(i)}\right)^{1-y^{(i)}} = \prod \left(\hat{y}^{(i)}\right)^{y^{(i)}} \left(1 - \hat{y}^{(i)}\right)^{1-y^{(i)}}$$

    Ex> **y** = (1, 0, 1, 1, 0) → p·(1-p)·p·p·(1-p)

  - Maximizing Likelihood = Minimizing [– log(Likelihood)]

    $$\therefore \arg\max_{\theta} p(D\,|\theta)$$

    $$= \arg\min_{\theta} \sum \left[-y^{(i)} \log(\hat{y}^{(i)}) - \left(1 - y^{(i)}\right) \log(1 - \hat{y}^{(i)})\right]$$

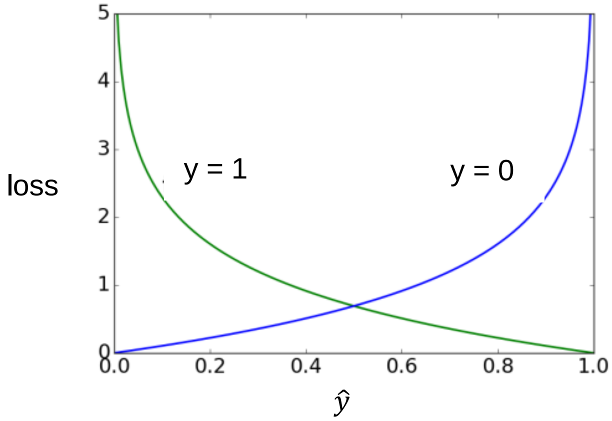
Cost Function

- Data $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \dots, \left(x^{(m)}, y^{(m)}\right)$
- Prediction model $\hat{y} = \dfrac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + \cdots + b)}}$
- Cost function (binary cross entropy)

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

  - If $y = 1 \Rightarrow -\log(\hat{y})$

    $y = 0 \Rightarrow -\log(1 - \hat{y})$



Gradient of BCE

- For 1 example data

$$J(\theta) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

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

$$z = w_0x_0 + w_1x_1 + \cdots + b$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{-(-e^{-z})}{(1+e^{-z})^2} = \frac{(1+e^{-z})-1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right)$$

$$= \hat{y}(1-\hat{y})$$

- Gradient of binary cross entropy

$$\frac{\partial J(\theta)}{\partial w_j} = \frac{\partial J(\theta)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_j}$$

$$= \left(-y\frac{1}{\hat{y}} + (1-y)\frac{1}{(1-\hat{y})}\right) \cdot \hat{y}(1-\hat{y}) \cdot x_j$$

$$= (\hat{y} - y) \cdot x_j$$

GD for Logistic Regression

- Data $\left(x_0{}^{(1)}, x_1{}^{(1)}, \dots, y^{(1)}\right), \left(x_0{}^{(2)}, x_1{}^{(2)}, \dots, y^{(2)}\right), \dots, \left(x_0{}^{(m)}, x_1{}^{(m)}, \dots, y^{(m)}\right)$
- Model $\hat{y} = \dfrac{1}{1+e^{-z}}$ $\quad z = w_0 x_0 + w_1 x_1 + \cdots + b$
- Cost $J(\theta) = \dfrac{1}{m}\displaystyle\sum_{i=1}^{m}\left[-y^{(i)}\log(\hat{y}^{(i)}) - \left(1-y^{(i)}\right)\log(1-\hat{y}^{(i)})\right]$
- Gradients

$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)x_0{}^{(i)}$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)x_1{}^{(i)}$$

…

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum\left(\hat{y}^{(i)} - y^{(i)}\right)$$

GD for Logistic Regression

- Vector form - Data (N features, M data)

$$\underset{\text{(M x N)}}{\mathbf{X} =} \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)} & \dots \\ {x_0}^{(2)} & {x_1}^{(2)} & \dots \\ {x_0}^{(3)} & {x_1}^{(3)} & \dots \\ & \dots & \end{bmatrix} \qquad \underset{\text{(M x 1)}}{\mathbf{y} =} \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \end{bmatrix} \qquad \underset{\text{(N x 1)}}{\mathbf{w} =} \begin{bmatrix} w_0 \\ w_1 \\ \dots \end{bmatrix}$$

- Model

$$z^{(1)} = w_0 {x_0}^{(1)} + w_1 {x_1}^{(1)} \dots + \mathrm{b}$$
$$z^{(2)} = w_0 {x_0}^{(2)} + w_1 {x_1}^{(2)} \dots + \mathrm{b}$$
$$z^{(3)} = w_0 {x_0}^{(3)} + w_1 {x_1}^{(3)} \dots + \mathrm{b}$$
$$\dots$$
$$\hat{y}^{(i)} = sigmoid\ (z^{(i)})$$

- Cost

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

$$\begin{bmatrix} z^{(1)} \\ z^{(2)} \\ z^{(3)} \\ \dots \end{bmatrix} = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)}, \dots \\ {x_0}^{(2)} & {x_1}^{(2)}, \dots \\ {x_0}^{(3)} & {x_1}^{(3)}, \dots \\ \dots & \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ w_1 \\ \dots \end{bmatrix} + \mathrm{b}$$

$$\mathbf{z} = \mathbf{X} \cdot \mathbf{w} + b$$
$$\hat{\mathbf{y}} = sigmoid(\mathbf{z})$$

$$cost = \frac{1}{m} sum(-\mathbf{y} \log(\hat{\mathbf{y}}) \\ -(1 - \mathbf{y}) \log(1 - \hat{\mathbf{y}}))$$

GD for Logistic Regression

- Gradient

$$\frac{\partial J}{\partial w_0} = \frac{1}{m}\left((\hat{y}^{(1)} - y^{(1)}){x_0}^{(1)} + (\hat{y}^{(2)} - y^{(2)}){x_0}^{(2)} + \cdots\right)$$

$$\frac{\partial J}{\partial w_1} = \frac{1}{m}\left((\hat{y}^{(1)} - y^{(1)}){x_0}^{(1)} + (\hat{y}^{(2)} - y^{(2)}){x_0}^{(2)} + \cdots\right)$$

…

$$\frac{\partial J}{\partial b} = \frac{1}{m}\left((\hat{y}^{(1)} - y^{(1)}) + (\hat{y}^{(2)} - y^{(2)}) + \cdots\right)$$

$$\begin{bmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \\ \ldots \end{bmatrix} = \frac{1}{m}\begin{bmatrix} {x_0}^{(1)}, {x_0}^{(2)}, {x_0}^{(3)}, \ldots \\ {x_1}^{(1)}, {x_1}^{(2)}, {x_1}^{(3)}, \ldots \\ \ldots \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \\ \ldots \end{bmatrix}$$

$$\nabla J(\mathbf{w}) = \frac{1}{m}\mathbf{X}^{\mathbf{T}}(\hat{\mathbf{y}} - \mathbf{y})$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}[1 \quad 1 \quad 1] \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \hat{y}^{(2)} - y^{(2)} \\ \hat{y}^{(3)} - y^{(3)} \\ \ldots \end{bmatrix}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} sum(\hat{\mathbf{y}} - \mathbf{y})$$

GD for Logistic Regression

- Learning (gradient descent)

Given

$$\mathbf{X} = \begin{bmatrix} {x_0}^{(1)} & {x_1}^{(1)} & \dots \\ {x_0}^{(2)} & {x_1}^{(2)} & \dots \\ {x_0}^{(3)} & {x_1}^{(3)} & \dots \\ & \dots & \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \end{bmatrix}$$

Initialize

$$\mathbf{w} = [w_1 \quad w_2 \quad \dots], \qquad b$$

repeat

$$\mathbf{z} = \mathbf{X} \cdot \mathbf{w} + b$$

$$\hat{\mathbf{y}} = sigmoid(\mathbf{z})$$

decision boundary

$$\boxed{\begin{aligned} \mathbf{w} &= \mathbf{w} - \alpha \frac{1}{m}\left(\mathbf{X}^{\mathrm{T}}(\hat{\mathbf{y}} - \mathbf{y})\right) \\ b &= b - \alpha \frac{1}{m}(sum(\hat{\mathbf{y}} - \mathbf{y})) \end{aligned}}$$

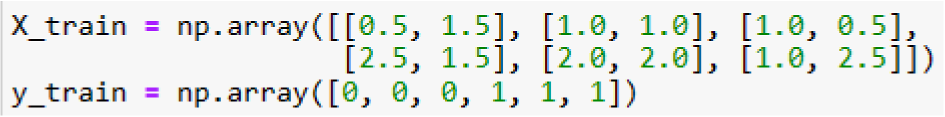
$$\mathbf{w} = \mathbf{w} - \alpha \nabla J(\mathbf{w})$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

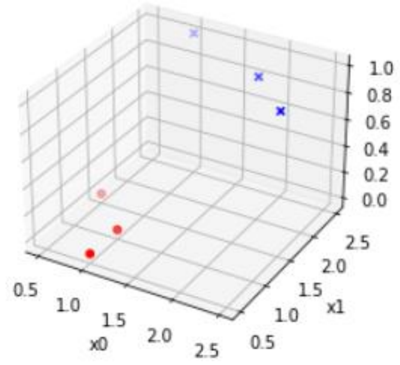
$$cost = \frac{1}{m} sum(-\mathbf{y}\log(\hat{\mathbf{y}}) - (1 - \mathbf{y})\log(1 - \hat{\mathbf{y}}))$$

Example

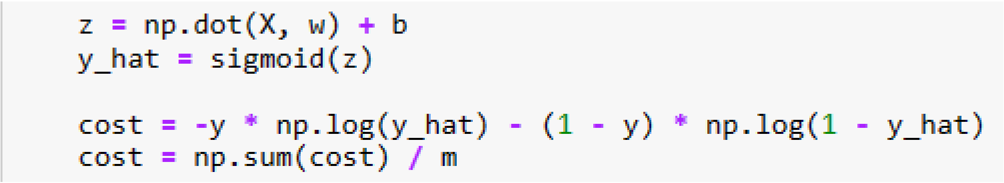
- Data



```
X_train = np.array([[0.5, 1.5], [1.0, 1.0], [1.0, 0.5],
                    [2.5, 1.5], [2.0, 2.0], [1.0, 2.5]])
y_train = np.array([0, 0, 0, 1, 1, 1])
```



- Compute $\hat{y}$ and cost



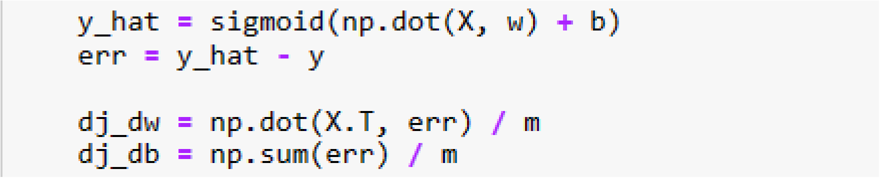
```
    z = np.dot(X, w) + b
    y_hat = sigmoid(z)

    cost = -y * np.log(y_hat) - (1 - y) * np.log(1 - y_hat)
    cost = np.sum(cost) / m
```

$$\mathbf{z} = \mathbf{X} \cdot \mathbf{w} + b$$

$$\hat{\mathbf{y}} = sigmoid(\mathbf{z})$$

$$cost = \frac{1}{m} sum(-\mathbf{y} \log(\hat{\mathbf{y}}) \\ -(1 - \mathbf{y}) \log(1 - \hat{\mathbf{y}}))$$

- Compute gradients



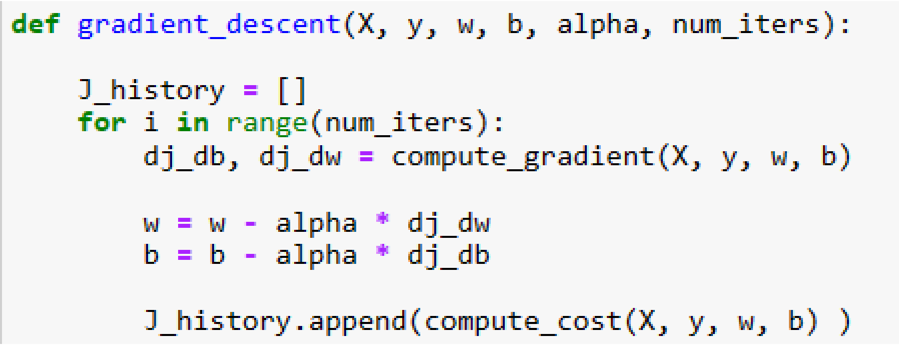
```
    y_hat = sigmoid(np.dot(X, w) + b)
    err = y_hat - y

    dj_dw = np.dot(X.T, err) / m
    dj_db = np.sum(err) / m
```

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{1}{m} \mathbf{X}^{\mathrm{T}} (\hat{\mathbf{y}} - \mathbf{y})$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} sum(\hat{\mathbf{y}} - \mathbf{y})$$

Example

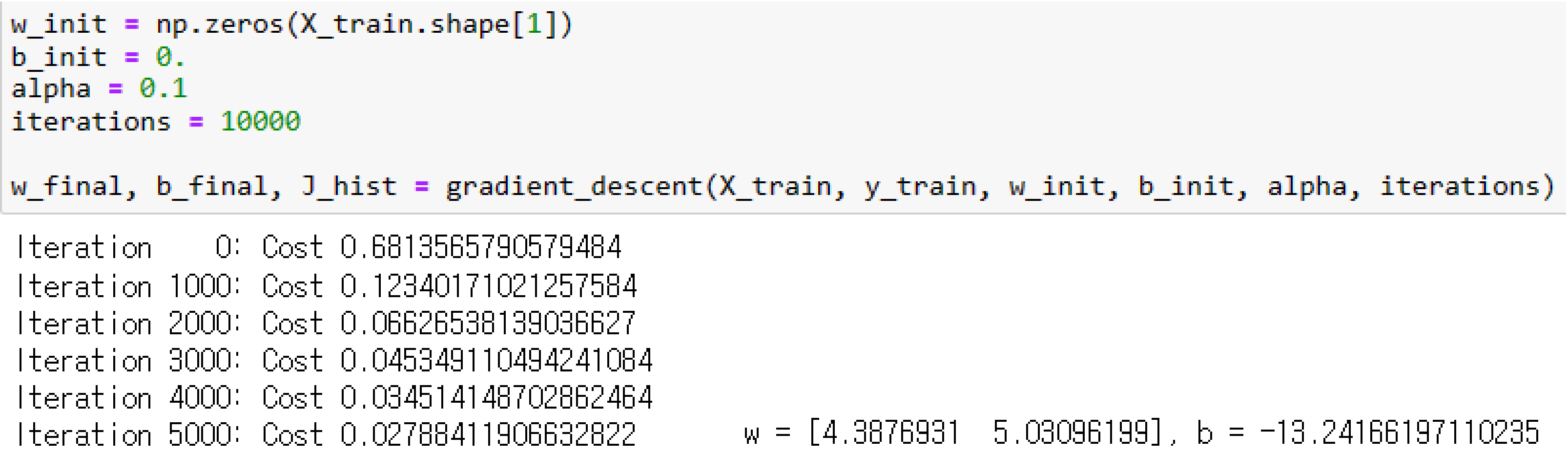
- Learning (gradient descent)



```python
def gradient_descent(X, y, w, b, alpha, num_iters):

    J_history = []
    for i in range(num_iters):
        dj_db, dj_dw = compute_gradient(X, y, w, b)

        w = w - alpha * dj_dw
        b = b - alpha * dj_db

        J_history.append(compute_cost(X, y, w, b) )
```

$$\mathbf{w} = \mathbf{w} - \alpha\,\nabla J(\mathbf{w})$$

$$b = b - \alpha\frac{\partial J}{\partial b}$$

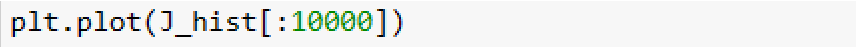


```python
w_init = np.zeros(X_train.shape[1])
b_init = 0.
alpha = 0.1
iterations = 10000

w_final, b_final, J_hist = gradient_descent(X_train, y_train, w_init, b_init, alpha, iterations)
```

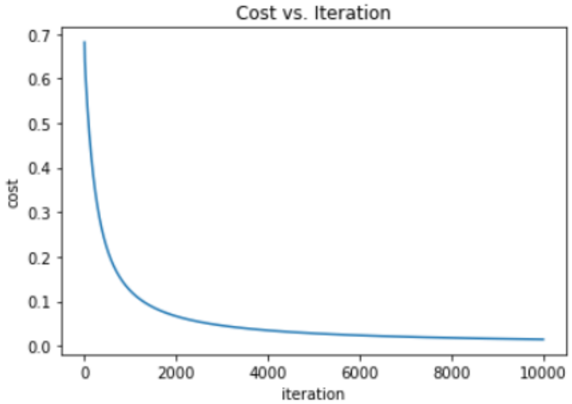
```
Iteration     0: Cost 0.6813565790579484
Iteration  1000: Cost 0.12340171021257584
Iteration  2000: Cost 0.06626538139036627
Iteration  3000: Cost 0.045349110494241084
Iteration  4000: Cost 0.034514148702862464
Iteration  5000: Cost 0.02788411906632822        w = [4.3876931   5.03096199], b = -13.24166197110235
```

Example

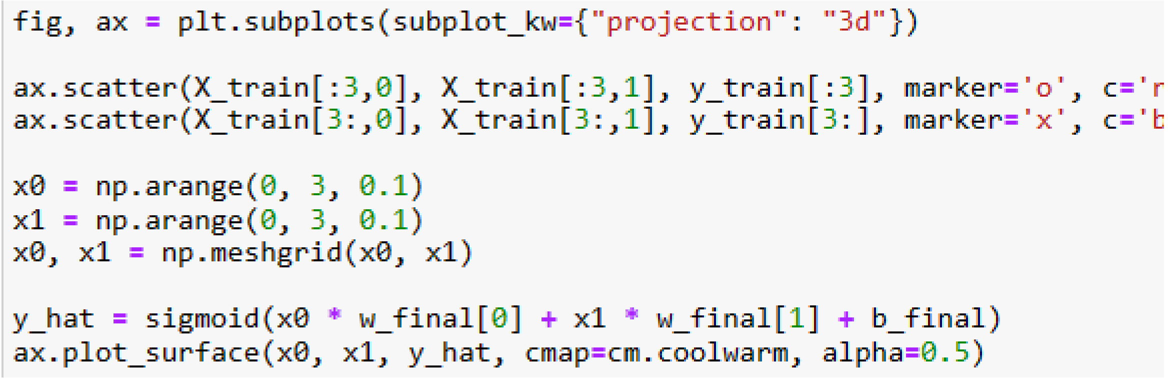
- Cost change



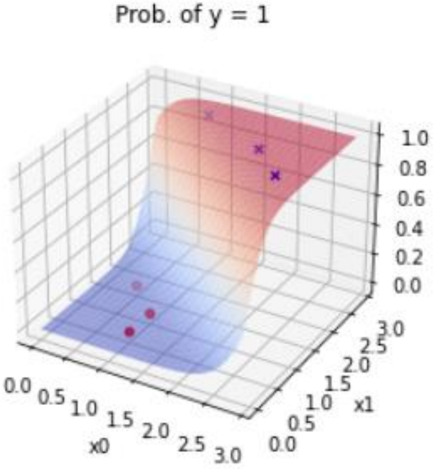
```
plt.plot(J_hist[:10000])
```



- Learned model

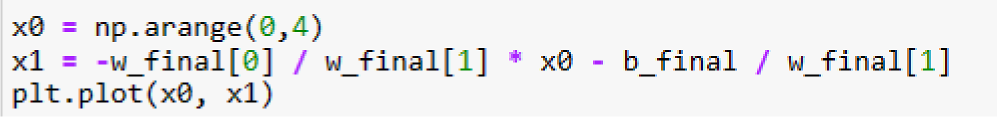


```
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

ax.scatter(X_train[:3,0], X_train[:3,1], y_train[:3], marker='o', c='r
ax.scatter(X_train[3:,0], X_train[3:,1], y_train[3:], marker='x', c='b

x0 = np.arange(0, 3, 0.1)
x1 = np.arange(0, 3, 0.1)
x0, x1 = np.meshgrid(x0, x1)

y_hat = sigmoid(x0 * w_final[0] + x1 * w_final[1] + b_final)
ax.plot_surface(x0, x1, y_hat, cmap=cm.coolwarm, alpha=0.5)
```



Example

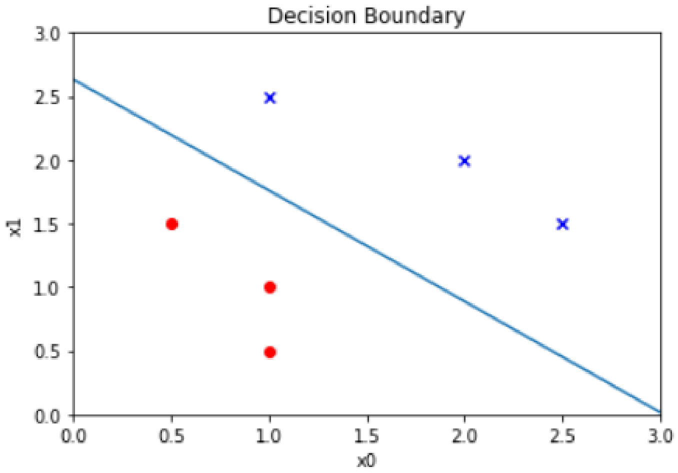
- Learned decision boundary



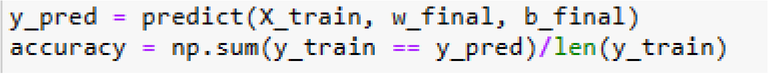
```
x0 = np.arange(0,4)
x1 = -w_final[0] / w_final[1] * x0 - b_final / w_final[1]
plt.plot(x0, x1)
```

$$\mathbf{z} = \mathbf{x} \cdot \mathbf{w} + b = 0$$

$$w_0 x_0 + w_1 x_1 + b = 0$$



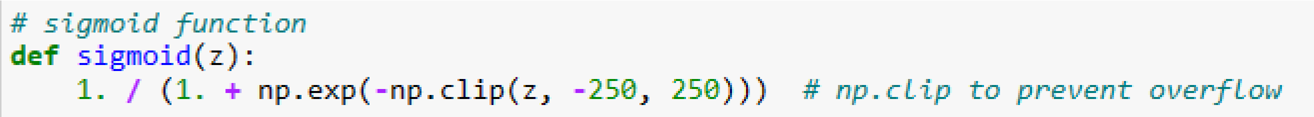
- Accuracy of the model



```
y_pred = predict(X_train, w_final, b_final)
accuracy = np.sum(y_train == y_pred)/len(y_train)
```

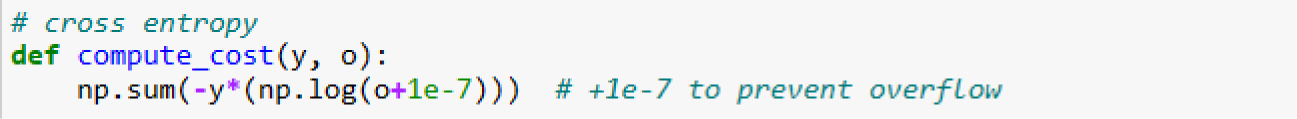
Example

- Preventing overflow
  - Prevent exp(∞) in sigmoid or softmax



```
# sigmoid function
def sigmoid(z):
    1. / (1. + np.exp(-np.clip(z, -250, 250)))  # np.clip to prevent overflow
```

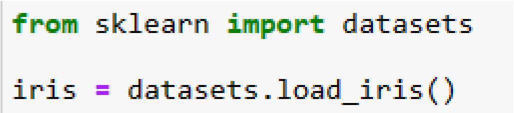
  - Prevent log(0) in computing cross entropy



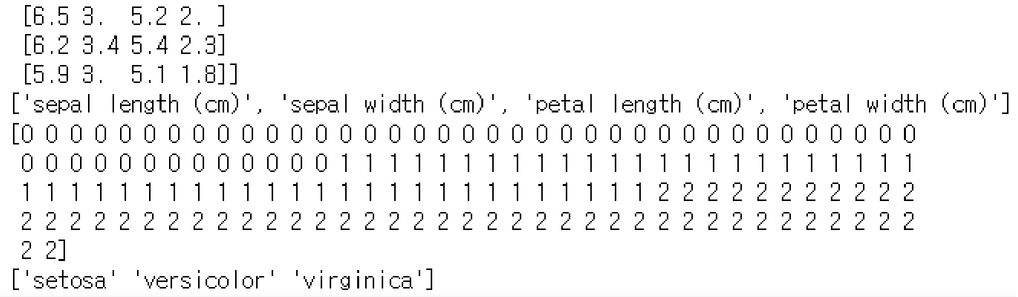
```
# cross entropy
def compute_cost(y, o):
    np.sum(-y*(np.log(o+1e-7)))  # +1e-7 to prevent overflow
```

Logistic Regression using Scikit Learn

- Load iris dataset from scikit learn → return dictionary with attributes:



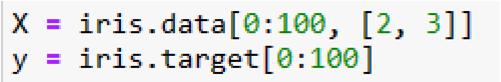
```python
from sklearn import datasets

iris = datasets.load_iris()
```



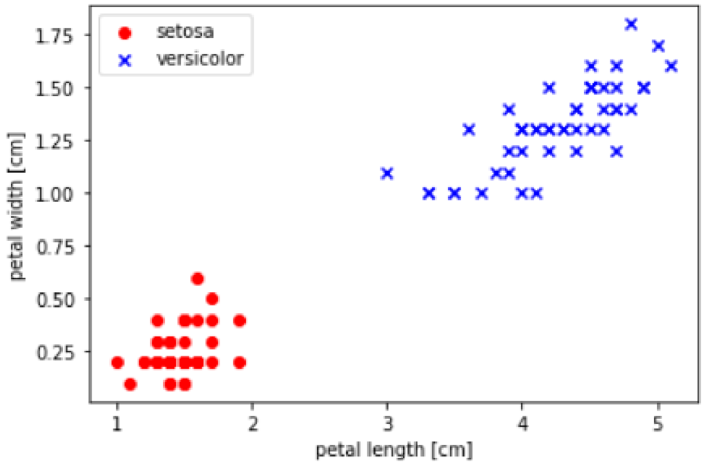
```
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```

  - data
  - feature_names
  - target
  - target_names

- Get X, y data
  - Select only features 2, 3
  - Select only class 0 and 1

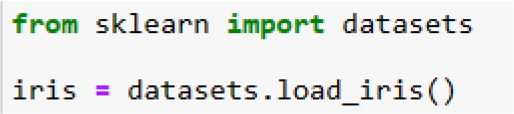


```python
X = iris.data[0:100, [2, 3]]
y = iris.target[0:100]
```



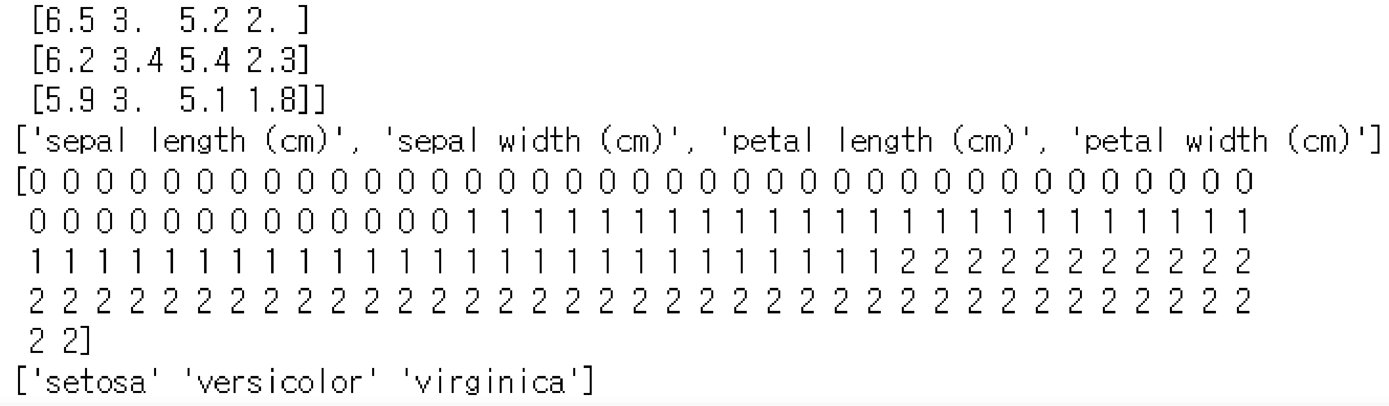
Logistic Regression using Scikit Learn

- Load iris dataset from scikit learn → return dictionary with attributes:



```
from sklearn import datasets

iris = datasets.load_iris()
```

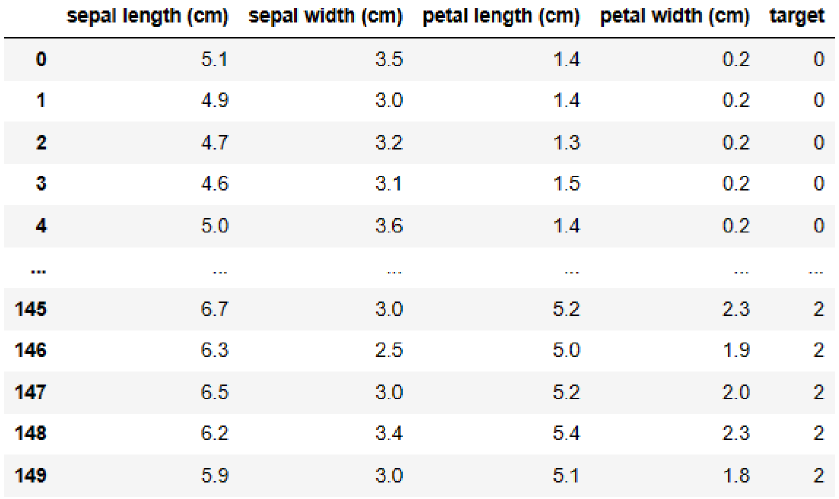
  - data
  - feature_names
  - target
  - target_names



```
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```

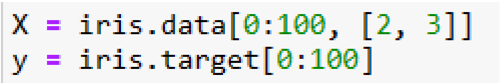
Logistic Regression using Scikit Learn

- Read into a DataFrame

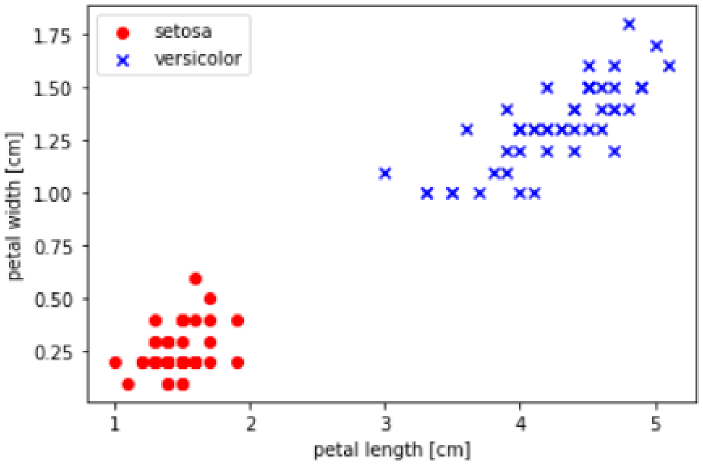


	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

- Get X, y array
  - Select only features 2, 3
  - Select only class 0 and 1

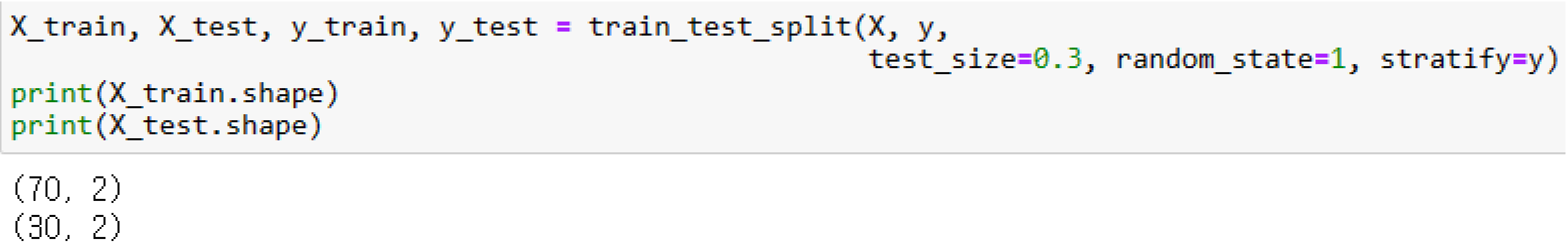


```
X = iris.data[0:100, [2, 3]]
y = iris.target[0:100]
```



Logistic Regression using Scikit Learn

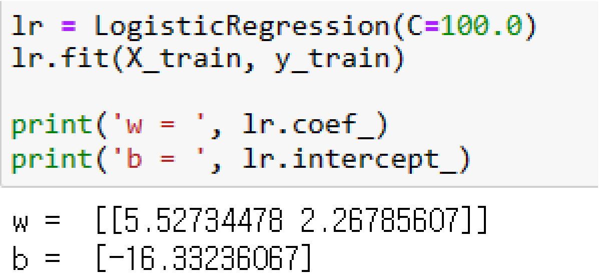
- Split data into training / test dataset
  - sklearn.model_selection.train_test_split



```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3, random_state=1, stratify=y)
print(X_train.shape)
print(X_test.shape)
```

```
(70, 2)
(30, 2)
```

- Use LogisticRegression. Training the model - .fit()

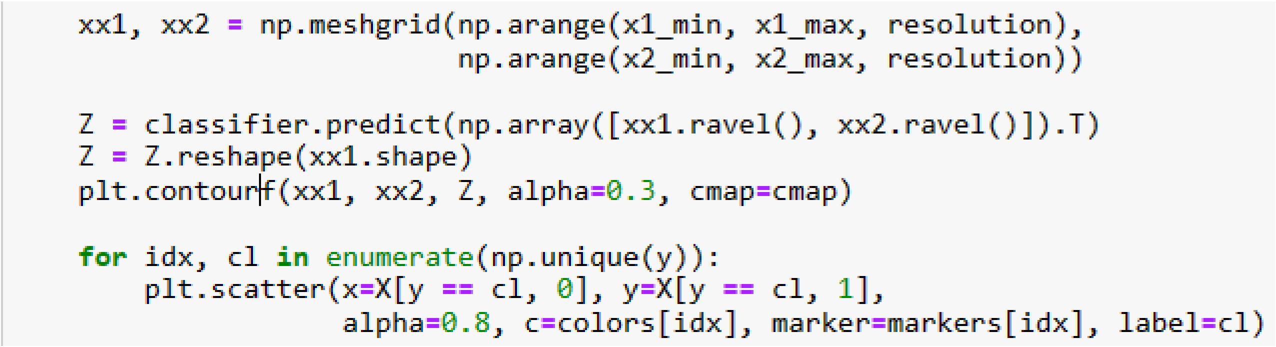


```
lr = LogisticRegression(C=100.0)
lr.fit(X_train, y_train)

print('w = ', lr.coef_)
print('b = ', lr.intercept_)
```

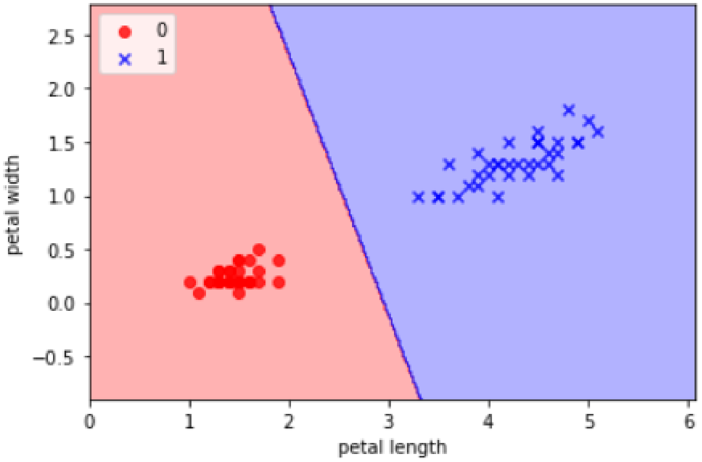
```
w =  [[5.52734478 2.26785607]]
b =  [-16.33236067]
```

Logistic Regression using Scikit Learn

- Plotting the decision boundary
  - Use np.meshgrid, plt.contour

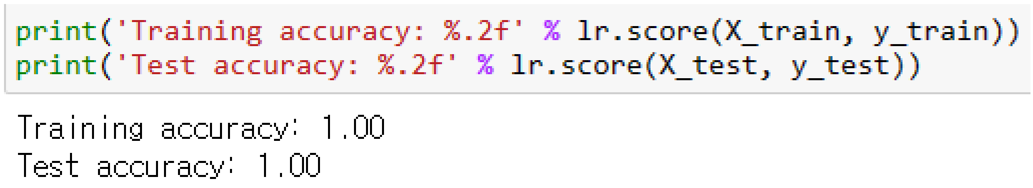


```
xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                       np.arange(x2_min, x2_max, resolution))

Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
Z = Z.reshape(xx1.shape)
plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)

for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                alpha=0.8, c=colors[idx], marker=markers[idx], label=cl)
```



Logistic Regression using Scikit Learn

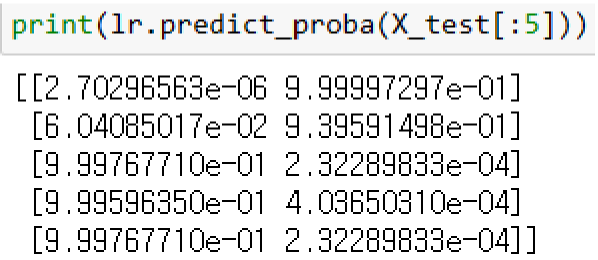
- Accuracy of the learned model - .score( )



```
print('Training accuracy: %.2f' % lr.score(X_train, y_train))
print('Test accuracy: %.2f' % lr.score(X_test, y_test))
```

```
Training accuracy: 1.00
Test accuracy: 1.00
```

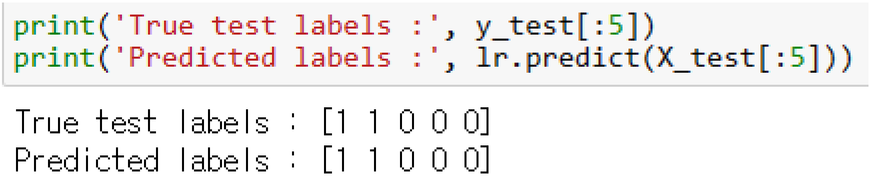
- Computing class probability - . predict_proba( )



```
print(lr.predict_proba(X_test[:5]))
```

```
[[2.70296563e-06 9.99997297e-01]
 [6.04085017e-02 9.39591498e-01]
 [9.99767710e-01 2.32289833e-04]
 [9.99596350e-01 4.03650310e-04]
 [9.99767710e-01 2.32289833e-04]]
```

- Predicting class labels - .predict( )

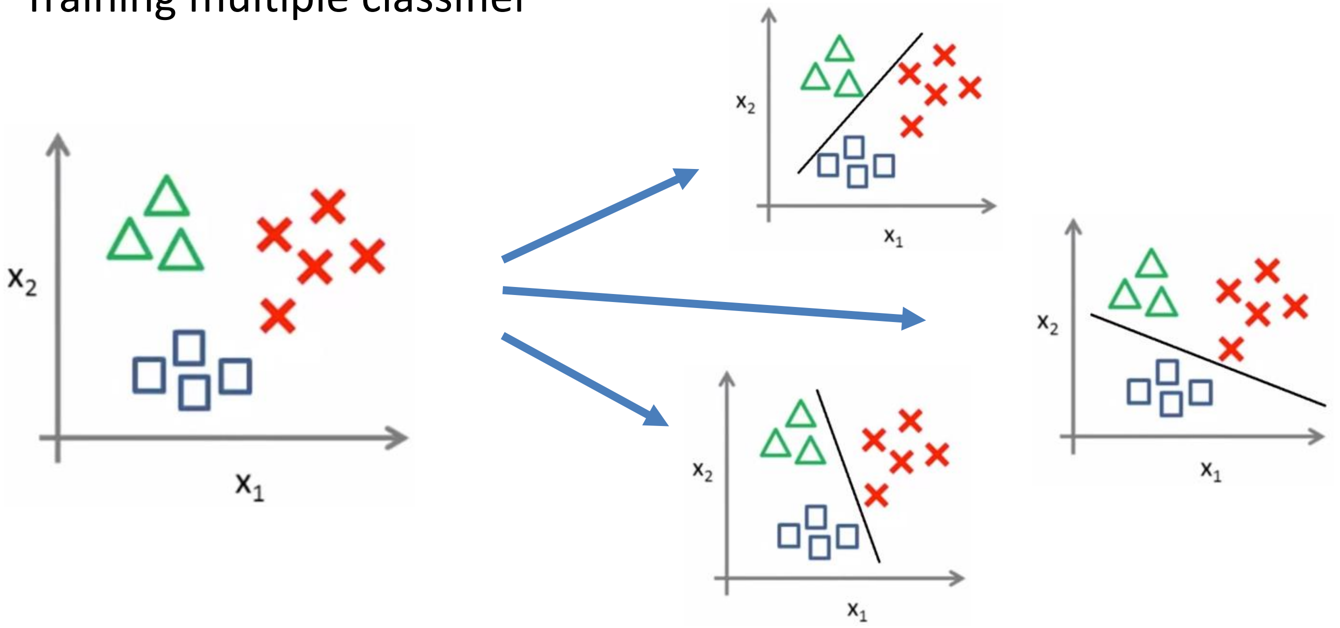


```
print('True test labels :', y_test[:5])
print('Predicted labels :', lr.predict(X_test[:5]))
```

```
True test labels : [1 1 0 0 0]
Predicted labels : [1 1 0 0 0]
```

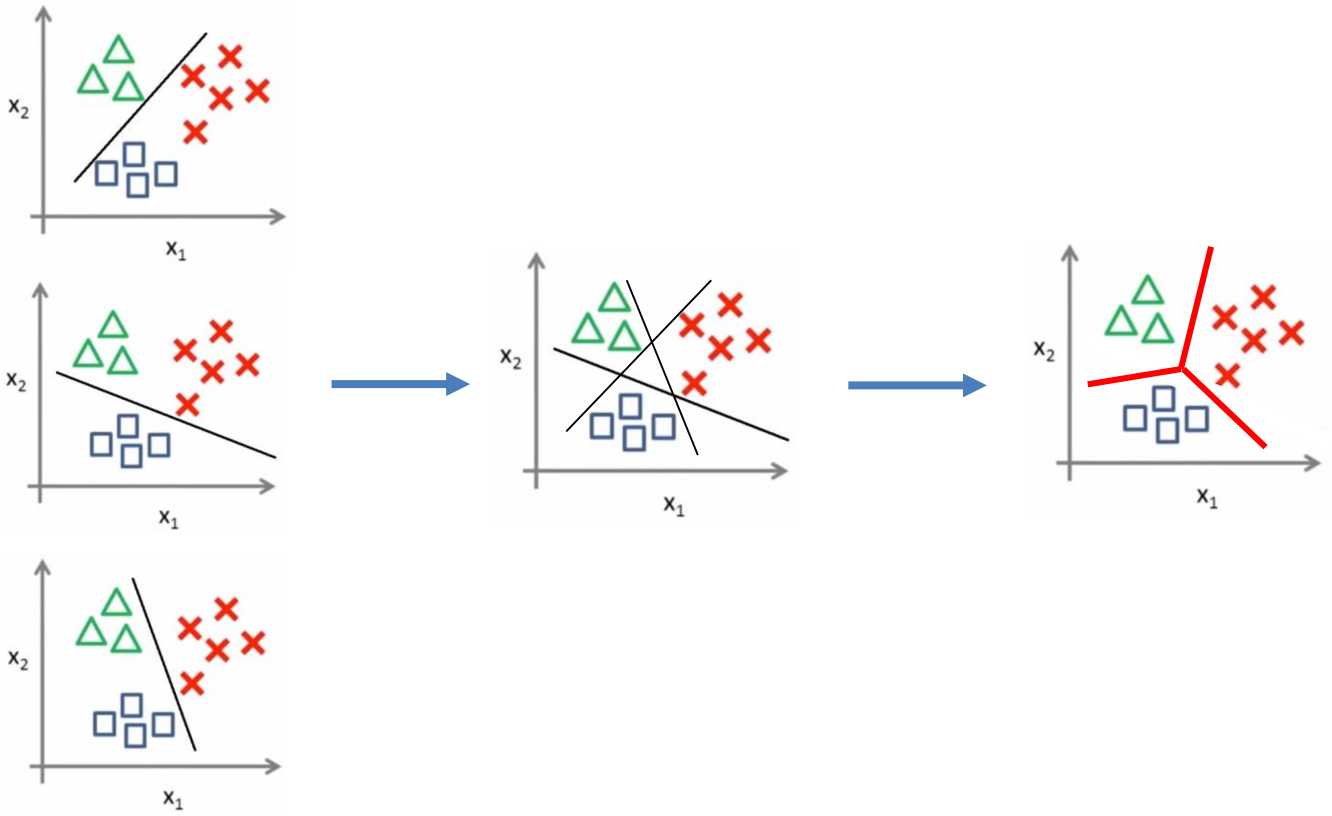
Multinomial Logistic Regression

- Multiple class case
  - Classes {0, 1} : binary classification
  - Classes {A, B, C} : multinomial classification → { (1, 0, 0), (0, 1, 0) (0, 0, 1) }
- OvR (One vs. Rest)
  - Training multiple classifier



Multinomial Logistic Regression

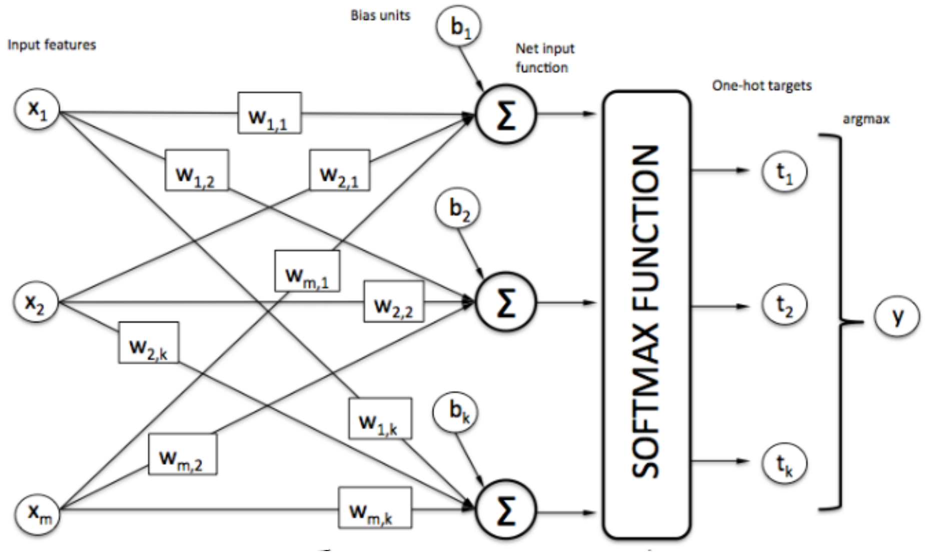
- OvR (One vs. Rest)
  - Decision boundary combining 3 classifier



Multinomial Logistic Regression

- Multinomial
  - Output function to predict y : softmax

x



	prob. of $\hat{y}$		label y = A
$z_1 = 0.86$	$\hat{y}_1 = 0.63$	$C_1 = 1$	$y_1 = 1$
$z_2 = 0.28$	$\hat{y}_2 = 0.35$	$C_2 = 0$	$y_2 = 0$
$z_3 = -2.85$	$\hat{y}_3 = 0.02$	$C_3 = 0$	$y_3 = 0$

$$z = w_0 x_1 + w_1 x_1 + w_2 x_2 + \cdots + b$$

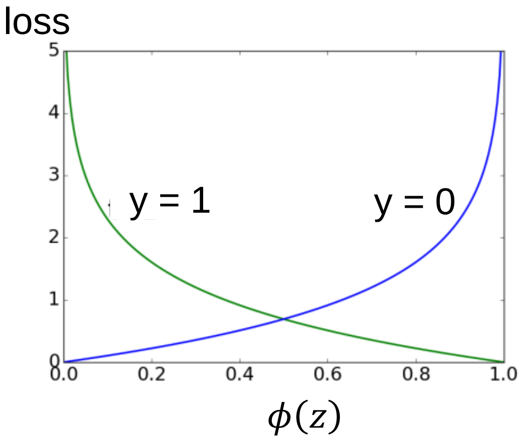
$$\boxed{\hat{y}_k = \frac{e^{z_k}}{\sum_i e^{z_i}}} \qquad Class_k = \begin{cases} 1 & if\ \ \hat{y}_k\ is\ max \\ 0 & otherwise \end{cases}$$

Cost Function

- Cost function (cross entropy)

$$J(\theta) = \sum J^{(i)}(\theta)$$

$$\boxed{J^{(i)}(\theta) = -\sum_{k} y_k \log \hat{y}_k}$$



- Example

$y_1 = 1$ $\hat{y}_1 = 0.63$

$y_2 = 0$ $\hat{y}_2 = 0.25$ ➡ $-\Sigma\, y \log \hat{y} = -(1 \times \log 0.63 + 0 \times \log 0.25 + 0 \times \log 0.12)$

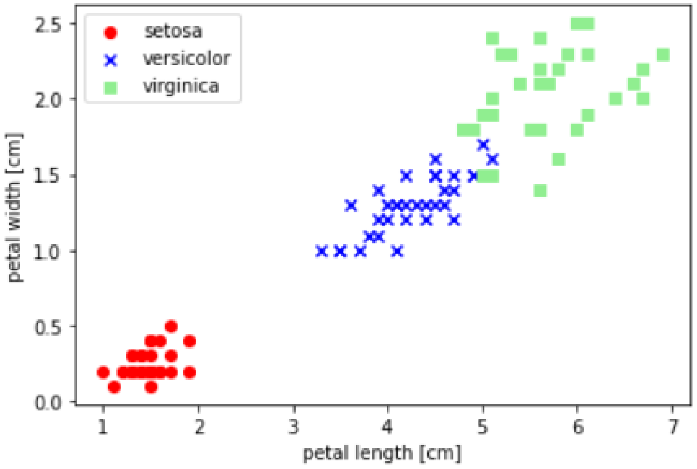
$y_3 = 0$ $\hat{y}_3 = 0.12$

※ For binary classification

$y = 1$ $\hat{y} = 0.63$ ➡ $-\Sigma\, y \log \hat{y} = -(1 \times \log 0.63 + 0 \times \log 0.37)$

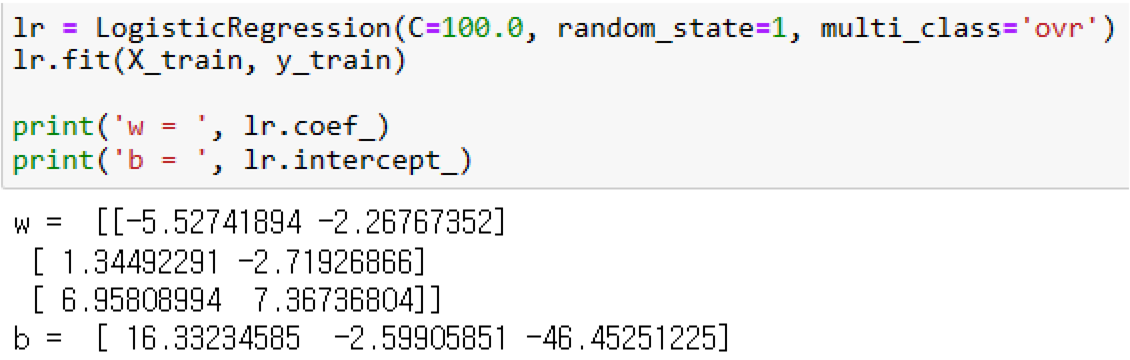
Multinomial Logistic Regression using Scikit Learn

- Load dataset from scikit learn
  - Select only features 2, 3
  - Select all class 0, 1, 2



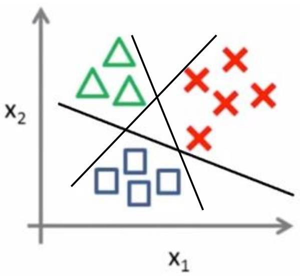
- Split data into training test dataset

- Training the model



```
lr = LogisticRegression(C=100.0, random_state=1, multi_class='ovr')
lr.fit(X_train, y_train)

print('w = ', lr.coef_)
print('b = ', lr.intercept_)
```

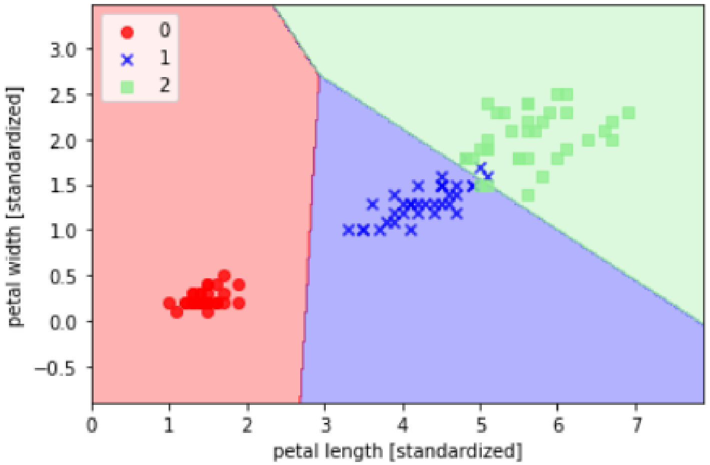
```
w =  [[-5.52741894 -2.26767352]
 [ 1.34492291 -2.71926866]
 [ 6.95808994  7.36736804]]
b =  [ 16.33234585  -2.59905851 -46.45251225]
```



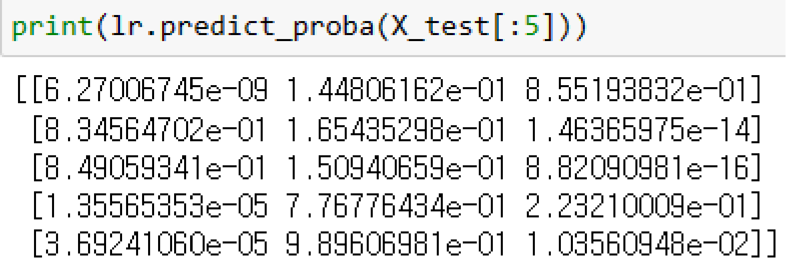
Machine Learning

Multinomial Logistic Regression using Scikit Learn

- The decision boundary



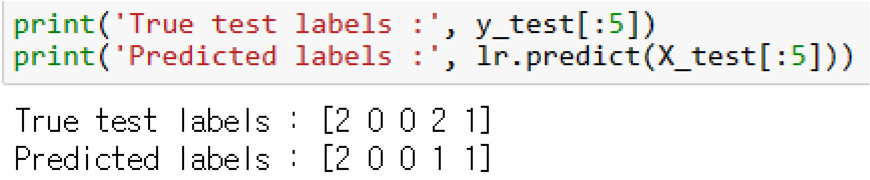
- Computing class probability



```
print(lr.predict_proba(X_test[:5]))
```

```
[[6.27006745e-09 1.44806162e-01 8.55193832e-01]
 [8.34564702e-01 1.65435298e-01 1.46365975e-14]
 [8.49059341e-01 1.50940659e-01 8.82090981e-16]
 [1.35565353e-05 7.76776434e-01 2.23210009e-01]
 [3.69241060e-05 9.89606981e-01 1.03560948e-02]]
```

- Predicting class labels



```
print('True test labels :', y_test[:5])
print('Predicted labels :', lr.predict(X_test[:5]))
```

```
True test labels : [2 0 0 2 1]
Predicted labels : [2 0 0 1 1]
```

Regularization

- Regularization
  - Add regularization term(*penalty*) to a cost function to prevent overfitting
- L2 regularization (ridge)
  - Add

$$\lambda\|\mathbf{w}\|_2^2 = \lambda\sum_{j=1}^{n} w_j^2 = \lambda({w_1}^2 + {w_2}^2 + \cdots + {w_n}^2)$$

  - Makes weights small, but not to 0. Less computationally expensive
  - Gradient descent

$$w_j = w_j - \alpha\frac{\partial J}{\partial w_j} - \alpha\lambda w_j$$

Regularization

- L1 regularization (lasso)
  - Add

$$\lambda\|\mathbf{w}\|_1 = \lambda\sum_{j=1}^{n}|w_j| = \lambda(|w_1| + |w_2| + \cdots + |w_n|)$$

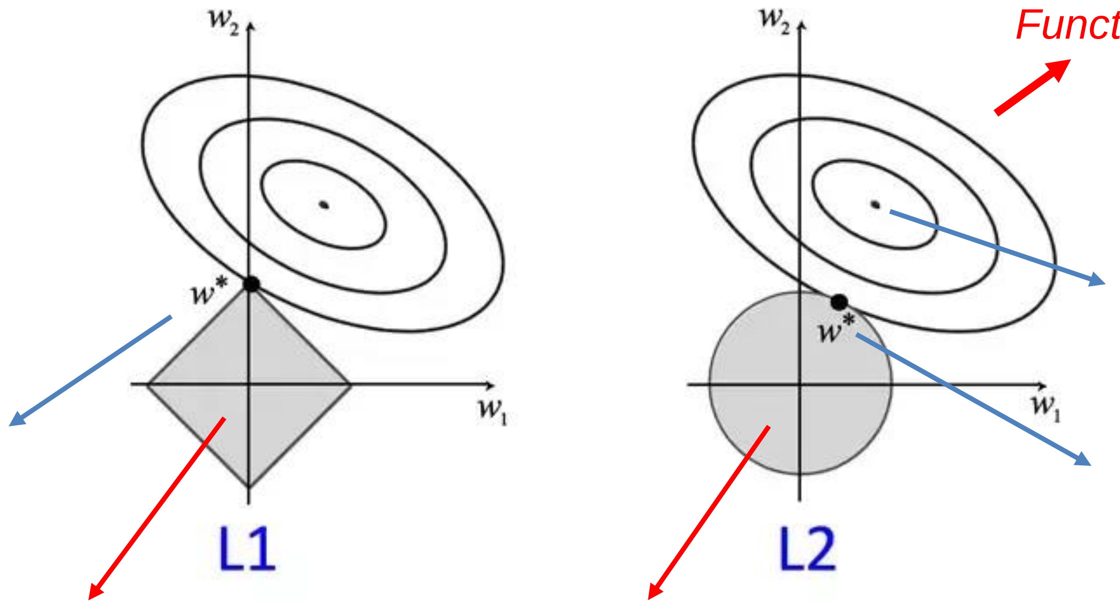
  - Makes some weights to 0 → feature selection. Robust to outliers
  - Gradient descent

$$w_j = w_j - \alpha\frac{\partial J}{\partial w_j} - \alpha\,\lambda\,sign(w_j)$$

Regularization

- Lasso vs Ridge

Original cost Function J(w)



w_1 = 0

Minimum without constraint

Minimum with constraint

$|w_1| + |w_2| \leq t$

$w_1{}^2 + w_2{}^2 \leq t$

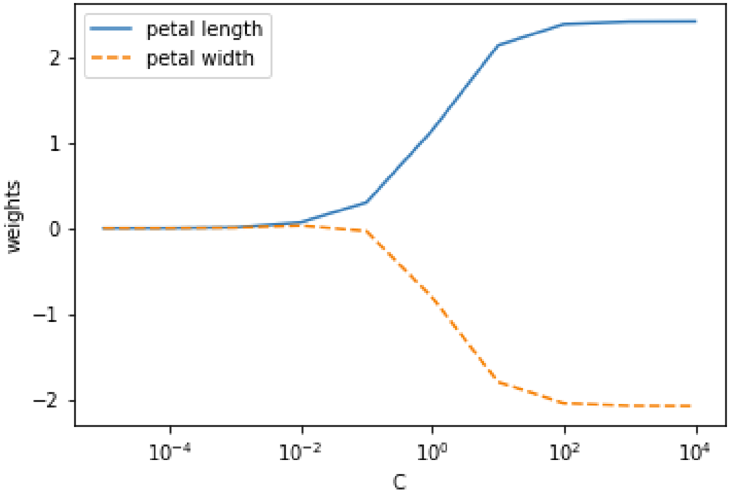
L2 Regularization in Scikit Learn

- Scikit learn LogisticRegression parameter C

$$C = \frac{1}{\lambda}$$

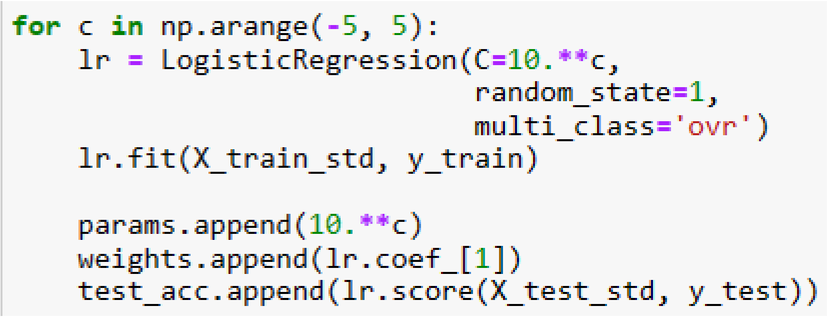
$$J(\boldsymbol{\theta}) = C\left[\sum_{i=1}^{n}\left[-y^{(i)}\log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right]\right] + \frac{1}{2}\|\mathbf{w}\|_2^2$$

- Effect of C
  - Small C → large λ → small w



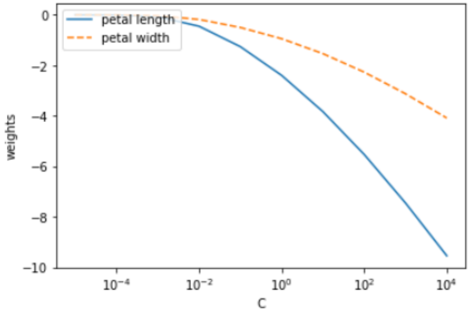
L2 Regularization in Scikit Learn

- Training the model with various C values

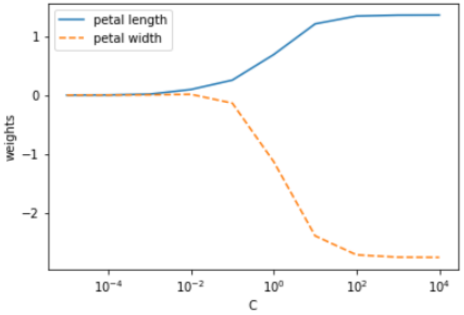


```
for c in np.arange(-5, 5):
    lr = LogisticRegression(C=10.**c,
                            random_state=1,
                            multi_class='ovr')
    lr.fit(X_train_std, y_train)

    params.append(10.**c)
    weights.append(lr.coef_[1])
    test_acc.append(lr.score(X_test_std, y_test))
```

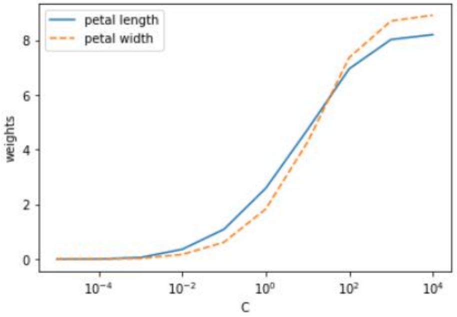
- Plot the parameter change



lr.coef_[0]



lr.coef_[1]



lr.coef_[2]