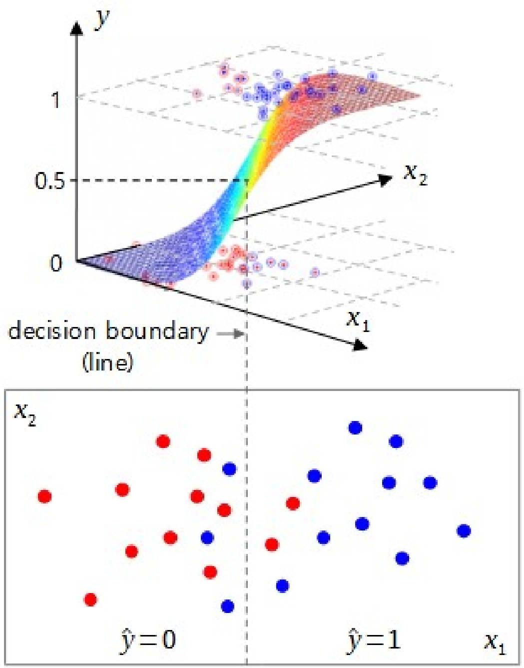
MX-AI




# 4. Logistic Regression

## Binary Classification

## Part 1: Overview and Objective Function for Logistic Regression

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

MX-AI

## Logistic Regression : Classification problem

- We want to predict the y value (y_pred) of the test data by learning the training data (x, y) as shown below. This is a classification problem.
- The y value of the training data is 0 or 1. This is a binary classification problem.
- We want the predicted value of y (y_pred) for the test data to be between 0 and 1. This is the probability that y is 1.
- If this probability is greater than 0.5, y is predicted to be 1, otherwise, y is predicted to be 0.
- Can we use a linear regression model to predict the y values for following data? The answer is no.

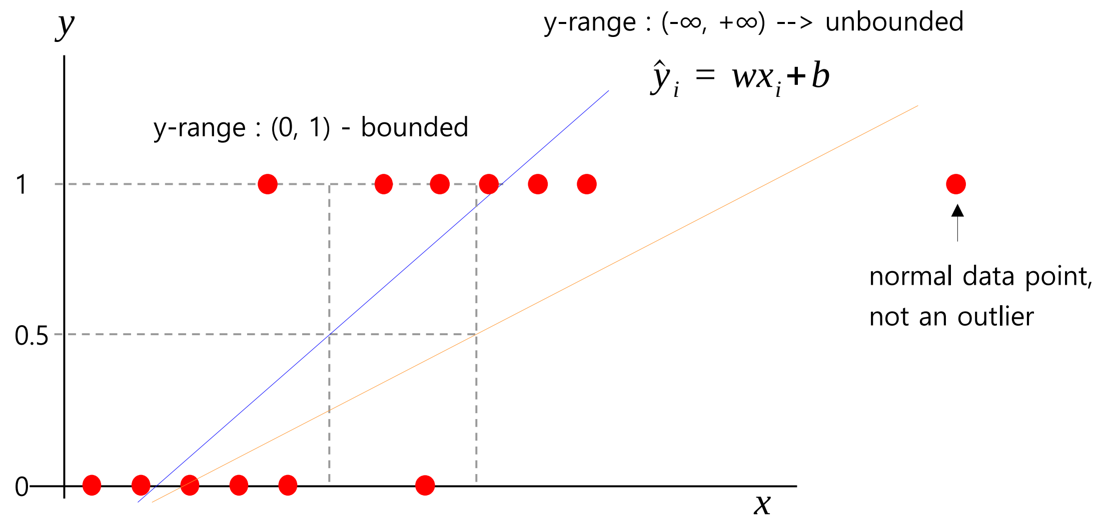
training data

| no | x | y |
|---|---|---|
| 1 | 1.2 | 0 |
| 2 | 10.0 | 1 |
| 3 | 3.5 | 0 |
| 4 | 8.2 | 1 |
| 5 | 2.0 | 0 |
| 6 | 9.4 | 1 |
| 7 | 1.5 | 0 |
| 8 | 7.8 | 1 |
| … | … | … |
| N | 2.5 | 0 |

predicted value

test data

| no | x | y | y_pred |
|---|---|---|---|
| 1 | 1.8 | ? | 0.10 |
| 2 | 6.2 | ? | 0.68 |



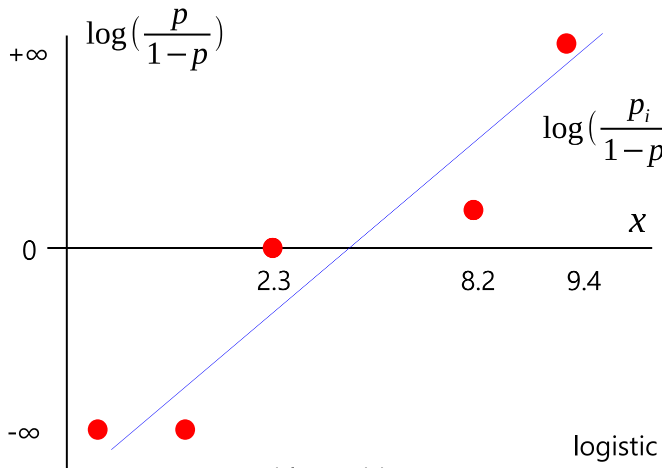
## Log odds (logit) and Linear Regression

- The y value is bounded to be 0 or 1, and the probability p is bounded to be between 0 and 1.
- Linear regression can be applied to the values unbounded from -∞ to +∞.
- Therefore, it is not appropriate to apply linear regression to the data below.
- However, linear regression can be applied by converting the range of the predicted value to -∞ ~ +∞.
- If the predicted value is expressed in log odds (logit), the range can be converted from -∞ to +∞.
- Linear regression can be applied in x and log odds (p) space instead of x and p space.
- When the x and log odds (p) space is transformed into x and p space, the regression line becomes a sigmoid function. This is the Logistic Regression.

| probability | odds | log odds (logit) |
|---|---|---|
| $p$ → | $\frac{p}{1-p}$ → | $\log\left(\frac{p}{1-p}\right)=logit(p)$ |
| (0 ~ 1) | (0 ~ +∞) | (-∞ ~ +∞) |

observed data — predicted (y_pred)

| no | x | y | p | log odds |
|---|---|---|---|---|
| 1 | 1.2 | 0 | 0.01 | -big |
| 2 | 1.8 | 0 | 0.01 | -big |
| 3 | 1.8 | 0 | 0.01 | -big |
| 4 | 2.3 | 0 | 0.50 | 0 |
| 5 | 2.3 | 1 | 0.50 | 0 |
| 6 | 8.2 | 1 | 0.67 | 0.7 |
| 7 | 8.2 | 1 | 0.67 | 0.7 |
| 8 | 8.2 | 0 | 0.67 | 0.7 |
| 9 | 9.4 | 1 | 0.99 | +big |
| 10 | 9.4 | 1 | 0.99 | +big |




$$\log\left(\frac{p_i}{1-p_i}\right)=wx_i+b$$

x and log odds(p) space

$$\frac{p_i}{1-p_i} = e^{wx_i+b}$$

$$p_i = e^{wx_i+b} - p_i e^{wx_i+b}$$

$$p_i = \frac{e^{wx_i+b}}{1+e^{wx_i+b}}$$

sigmoid or logistic function →

$$p_i = \frac{1}{1+e^{-(wx_i+b)}} = \hat{y}$$

MX-AI

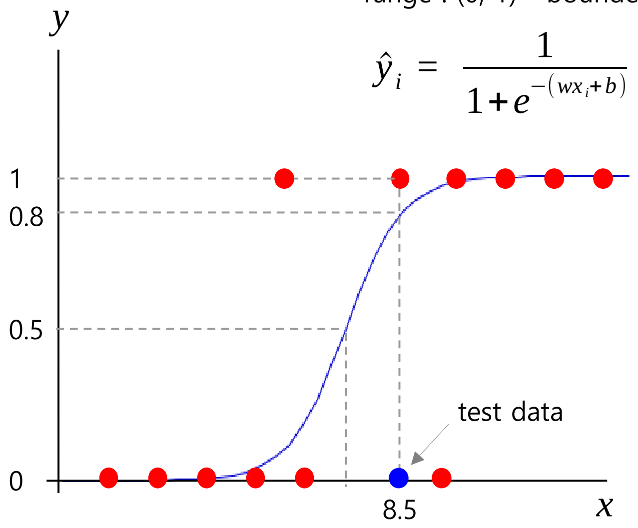
## Logistic or Sigmoid function

- Binary classification can be performed by fitting the logistic (or sgmoid) function to the training data in x and y space. w and b can be estimated.
- Use the estimated logistic function to predict the class y of the test data.

range : (0, 1) - bounded

$$\hat{y}_i = \frac{1}{1+e^{-(wx_i+b)}}$$

sigmoid
or
logistic function




x and y space

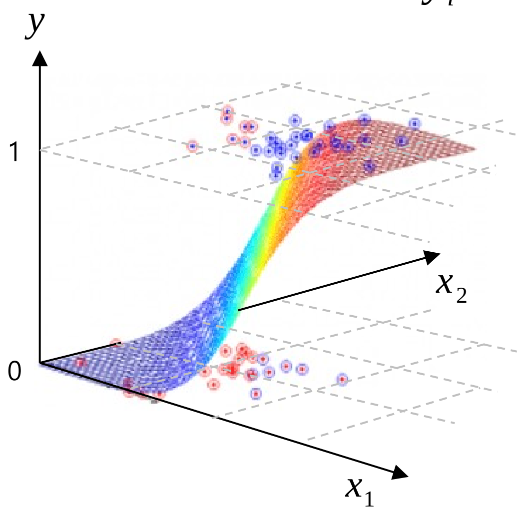
[ training data ]

| x | y |
|---|---|
| 1.2 | 0 |
| 10.0 | 1 |
| 3.5 | 0 |
| … | … |

[ test data ]

| x | p | y-hat |
|---|---|---|
| 8.5 | 0.8 | 1 |

$$\hat{y}_i = \frac{1}{1+e^{-(w_1x_{1,i}+w_2x_{2,i}+b)}}$$




[ multivariate ]

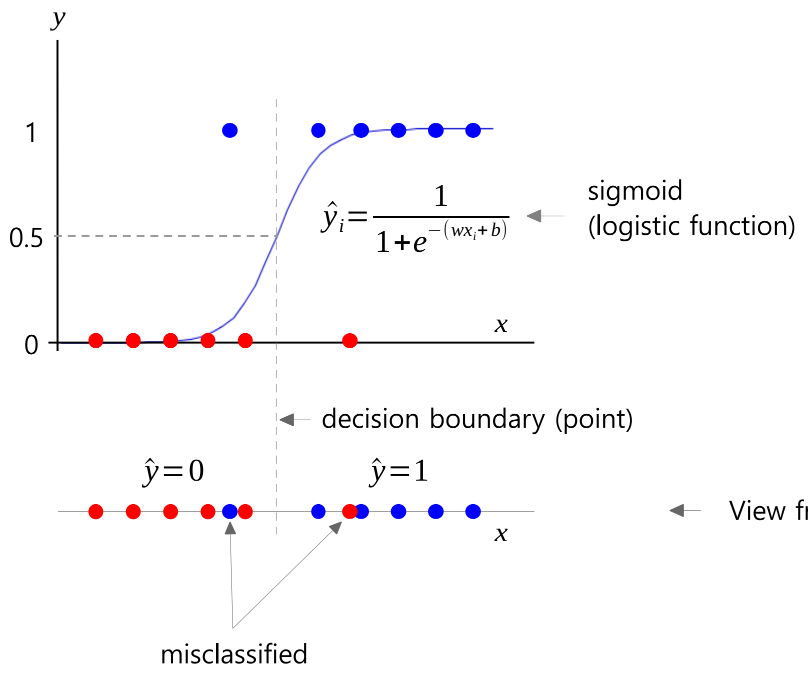
| x1 | x2 | y |
|---|---|---|
| 1.2 | 2.3 | 0 |
| 10.3 | 12.1 | 1 |
| 2.8 | 1.7 | 0 |
| … | … | … |

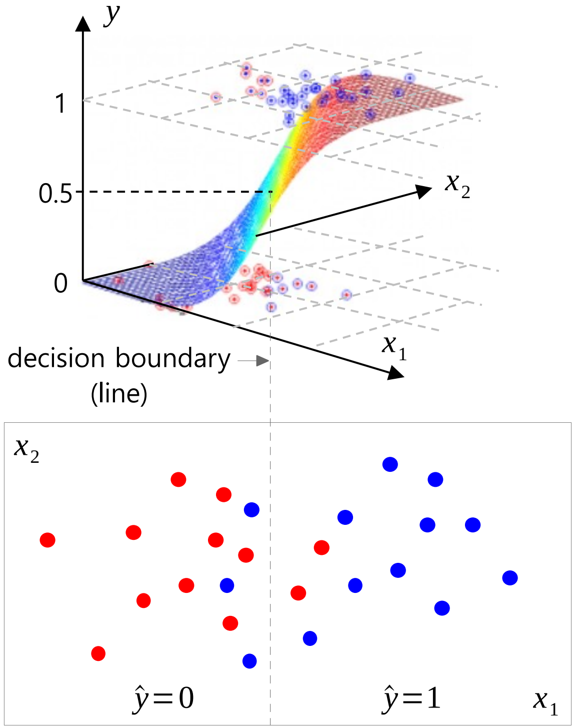
MX-AI

## ■ Decision boundary

- If the sigmoid value is greater than 0.5, the y class is predicted to be 1, otherwise it is predicted to be 0.
- The point where the sigmoid value is 0.5 is called the decision boundary.




View from above




Each data point in 3D space is projected into 2D space.

MX-AI

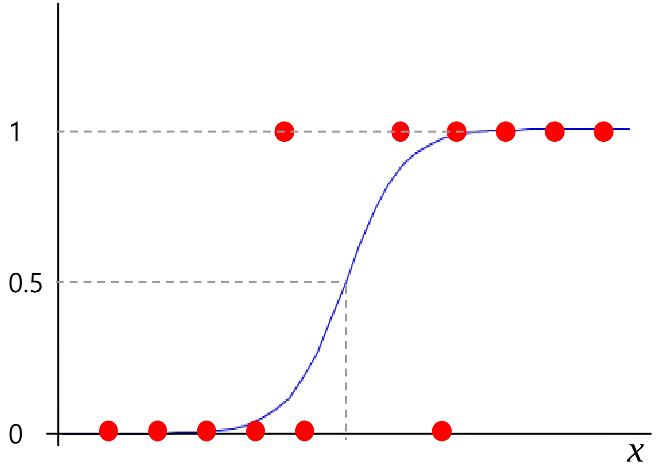
## Objective function and Regularization

- In linear regression, maximum likelihood estimation (MLE) was used to generate the objective function. In the same way, Logistic Regression can also use MLE to create an objective function that minimizes binary cross entropy.
- You can add a regularization term to the objective function in the same way as linear regression.

| $y$ | $\hat{y}$ | $P(y\|x)=\hat{y}^y(1-\hat{y})^{1-y}$ |
|---|---|---|
| 1 | 0.8 | $\rightarrow 0.8^1 \times 0.2^0 = 0.8$ |
| 1 | 0.2 | $\rightarrow 0.2^1 \times 0.8^0 = 0.2$ |
| 0 | 0.2 | $\rightarrow 0.2^0 \times 0.8^1 = 0.8$ |

The larger this value, the better the prediction.

$$\hat{y}_i(x|w,b) = \frac{1}{1+e^{-(wx_i+b)}}$$




$$\begin{cases} P(y=1|x;w,b)=\hat{y}(x;w,b) \\ P(y=0|x;w,b)=1-\hat{y}(x;w,b) \end{cases}$$

$$P(y|x)=\hat{y}^y(1-\hat{y})^{1-y}$$ (notations w, b are omitted)

The large the P(y|x), the better the prediction performance.

$$L(w,b)=\prod_{i=1}^{N} P(y_i \mid x_i;w,b)$$

$$\log(L(w,b))=\sum_{i=1}^{N} \log(\hat{y}_i^{y_i}(1-\hat{y}_i)^{1-y_i})$$

$$\log(L(w,b))=\sum_{i=1}^{N} [y_i \cdot \log \hat{y}_i + (1-y_i) \cdot \log(1-\hat{y}_i)]$$

- Objective function (binary cross entropy)

$$\max_{w,b} \sum_i \log(L(w,b))$$

$$= \min_{w,b} \sum_i [-y_i \cdot \log \hat{y}_i - (1-y_i) \cdot \log(1-\hat{y}_i)]$$

$$= \min_{w,b} \sum_i BCE$$

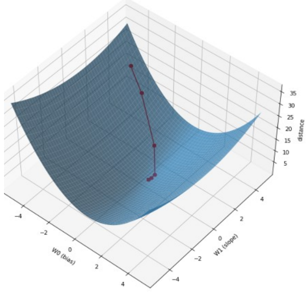
- Regularization

$$\min_{w,b} \sum_i BCE + \lambda \sum_{j=0}^{k} |w_j|$$ (Lasso)

$$\min_{w,b} \sum_i BCE + \lambda \sum_{j=0}^{k} w_j^2$$ (Ridge)

MX-AI

## Convexity of the objective function



Objective function
for Linear Regression

- Binary cross entropy, the objective function of logistic regression, is a convex function and has a unique solution.
- Since the second derivative of binary cross entropy is always greater than 0, we can see that it is a convex function.

$$J(w,b)=\sum_{i=1}^{N}\left[-y_i \cdot \log \hat{y}_i-(1-y_i)\cdot \log(1-\hat{y}_i)\right]$$

$$\hat{y}_i(x|w,b)=\frac{1}{1+e^{-(wx_i+b)}}$$

$$\hat{y}_i=\frac{1}{1+e^{-t}}, \quad (t=wx_i+b)$$

$$J_i=-y\cdot \log \hat{y}_t-(1-y)\cdot \log(1-\hat{y}_t)$$

→ binary cross entropy

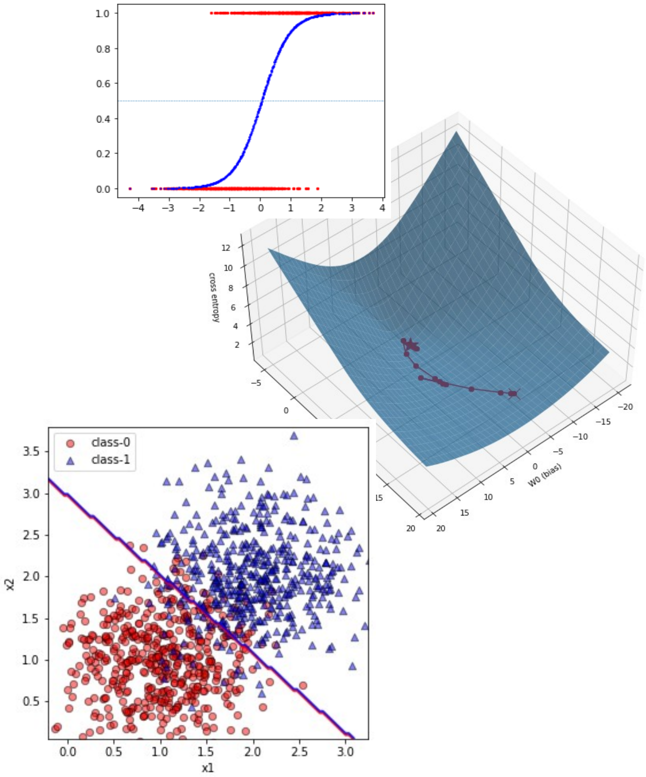
$$\frac{d}{dt}\hat{y}_i=\frac{e^{-t}}{(1+e^{-t})^2}=\frac{1+e^{-t}}{(1+e^{-t})^2}-\frac{1}{(1+e^{-t})^2}$$

$$=\frac{1}{1+e^{-t}}\left(1-\frac{1}{1+e^{-t}}\right)=\hat{y}_t(1-\hat{y}_t)$$

$$\frac{d}{dt}J_i = -\frac{y}{\hat{y}_t}\hat{y}_t(1-\hat{y}_t)+(1-y)\frac{\hat{y}_t(1-\hat{y}_t)}{1-\hat{y}_t} = -y+\hat{y}_t$$

$$\frac{d^2}{dt^2}J_i = \hat{y}_t(1-\hat{y}_t) = \frac{1}{1+e^{-t}}-\frac{1}{(1+e^{-t})^2} = \frac{e^{-t}}{(1+e^{-t})^2} > 0$$

MX-AI




# 4. Logistic Regression

## Binary Classification

## Part 2: Implementation of Logistic Regression

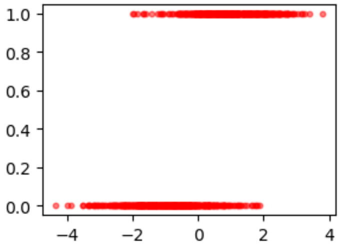
This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

## Implementation of binary classification in Logistic Regression using scipy.optimize

```
# [MXML-4-02] 1.bin_class(scipy).pyt
# Logistic Regression : binary classification
from scipy import optimize
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Create a simple dataset for binary classification
def bin_class_data(n):
    n1 = int(n / 2)
    a = np.random.normal(-1.0, 1.0, n1)
    b = np.random.normal(1.0, 1.0, n1)
    x = np.hstack([a, b]).reshape(-1, 1)
    y = np.hstack([np.zeros(n1), np.ones(n1)])
    return x, y
```




```
x, y = bin_class_data(n=1000) # create 1000 data points
X = np.hstack([np.ones([x.shape[0], 1]), x])
y = y.astype('int8')

# Visually check the data
plt.scatter(x, y, c='r', s=10, alpha=0.5)
plt.show()

# Split the data into training and test data
x_train, x_test, y_train, y_test = train_test_split(X, y)
```

```
# Loss function : mean of binary cross entropy
def bce_loss(W, args):
    tx = args[0]
    ty = args[1]
    trc = args[2]

    y_hat = 1.0 / (1 + np.exp(-np.dot(W, tx.T)))
    bce = -ty * np.log(y_hat + 1e-8) - \
          (1.0 - ty) * np.log(1.0 - y_hat + 1e-8)
    loss = bce.mean()

    # save the loss
    if trc == True:
        trace_W.append([W, loss])
    return loss
```

$$\hat{y}_i = \frac{1}{1+e^{-(wx_i+b)}}$$

$$BCE = -y_i \cdot \log \hat{y}_i - (1-y_i) \cdot \log(1-\hat{y}_i)$$

```
# Perform an optimization process
trace_W = []
result = optimize.minimize(fun = bce_loss,
                           x0 = [-5, 15],
                           args=[x_train, y_train, True])

# print the result. result.x contains the optimal parameters
print(result)
```

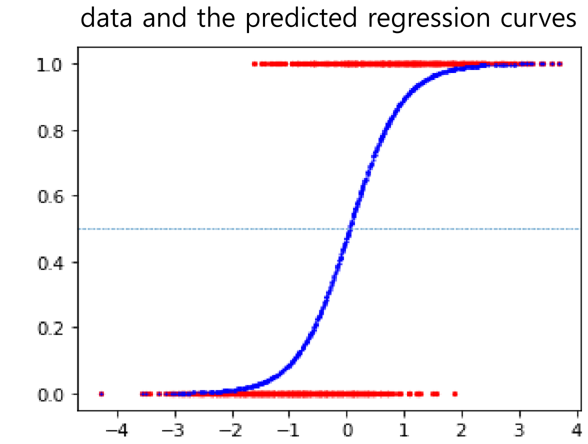
MX-AI

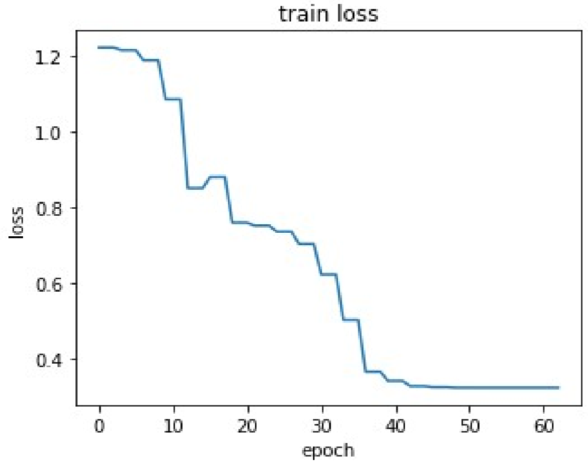
## ■ Implementation of binary classification in Logistic Regression using scipy.optimize

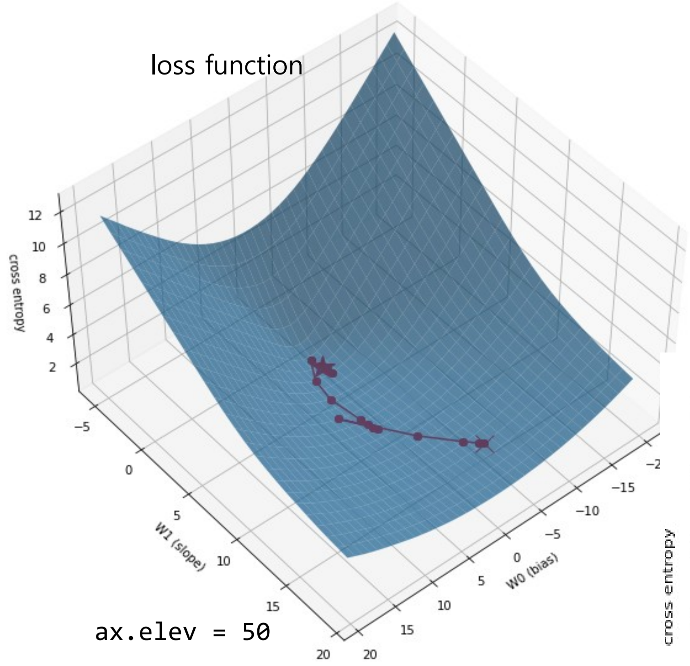
```
# Visually check the data and the predicted regression curves
y_hat = 1.0 / (1 + np.exp(-np.dot(result.x, x_train.T)))
plt.figure(figsize=(5, 4))
plt.scatter(x, y, s=5, c='r', label = 'data')
plt.scatter(x_train[:, 1], y_hat, c='blue', s=1, label='sigmoid')
plt.legend()
plt.axhline(y = 0.5, linestyle='--', linewidth=0.5)
plt.show()

# Measure the accuracy of test data
y_prob = 1.0 / (1 + np.exp(-np.dot(result.x, x_test.T)))
y_pred = (y_prob > 0.5).astype('int8')
acc = (y_pred == y_test).mean()
print('\nAccuracy of test data = {:.3f}'.format(acc))

# Visually check the loss function and the path to the optimal point
w0, w1 = np.meshgrid(np.arange(-20, 20, 1), np.arange(-5, 20, 1))
zs = np.array([bce_loss(np.array([a, b]),
              [x_train, y_train, False]) \
               for [a, b] in zip(np.ravel(w0), np.ravel(w1))])
z = zs.reshape(w0.shape)

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

# Drawing the surface of the loss function
ax.plot_surface(w0, w1, z, alpha=0.7)
```

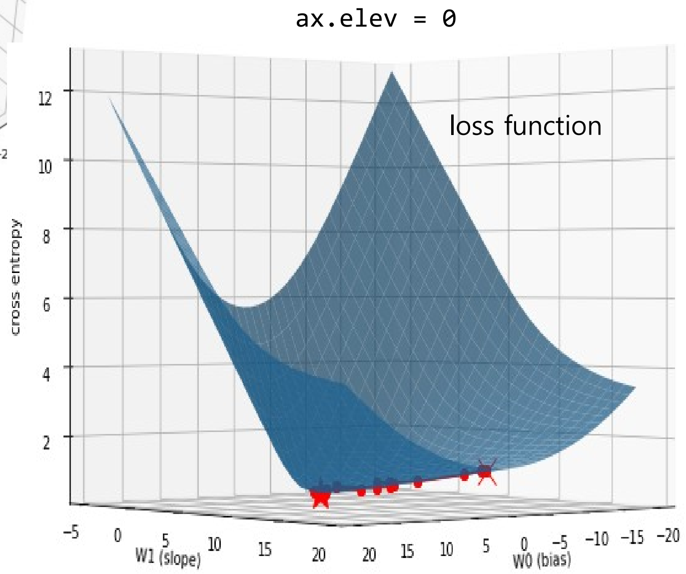
```
# Drawing the path to the optimal point
b = np.array([tw0 for [tw0, tw1], td in trace_W])
w = np.array([tw1 for [tw0, tw1], td in trace_W])
d = np.array([td for [tw0, tw1], td in trace_W])
ax.plot(b[0], w[0], d[0], marker='x', markersize=15, color="r")
ax.plot(b[-1], w[-1], d[-1], marker='*', markersize=20, color="r")
ax.plot(b, w, d, marker='o', color="r")

ax.set_xlabel('W0 (bias)')
ax.set_ylabel('W1 (slope)')
ax.set_zlabel('cross entropy')
ax.azim = 50
ax.elev = 50    # [50, 0]
plt.show()

# Visually see that the loss decreases as the iteration progresses
plt.figure(figsize=(5, 4))
plt.plot([e for w, e in trace_W], color='red')
plt.title('train loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

- Implementation of binary classification in Logistic Regression using scipy.optimize




Accuracy of test data = 0.820

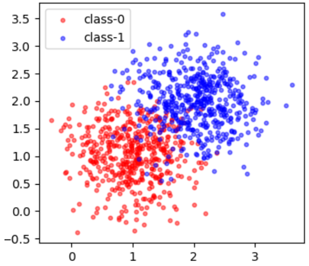
 

MX-AI

## Implementation of binary classification in Logistic Regression using sklearn: Decision Boundary

- Let's use sklearn's LogisticRegression() and check the decision boundary visually.

```
# [MXML-4-02] 2.bin_class(sklearn).py
# Logistic Regression : binary classification
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Create simple training data
x, y = make_blobs(n_samples=1000, n_features=2,
                  centers=[[1., 1.], [2., 2.]],
                  cluster_std=0.5)

# Visually check the data
color = ['red', 'blue']
for i in [0, 1]:
    idx = np.where(y == i)
    plt.scatter(x[idx, 0], x[idx, 1], c=color[i], s = 40,
                alpha = 0.5, label='class-'+str(i))
plt.legend()
plt.show()

# Split the data into training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y)
```



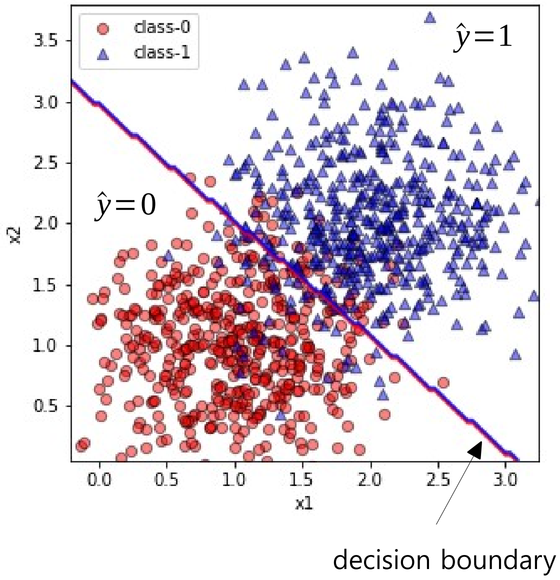

```
# Create a model and fit it to training data.
model = LogisticRegression()
model.fit(x_train, y_train)

# Predict the classes of test data, and measure the accuracy.
y_pred = model.predict(x_test)
acc = (y_pred == y_test).mean()
print('\nAccuracy of test data = {:.3f}'.format(acc))

# Visually check the decision boundary.
# reference: https://psrivasin.medium.com/plotting-decision-
             boundaries-using-numpy-and-matplotlib-f5613d8acd19
x1_min, x1_max = x_test[:, 0].min() -0.1, x_test[:,0].max() +0.1
y1_min, y1_max = x_test[:, 1].min() -0.1, x_test[:, 1].max() +0.1
x1, x2 = np.meshgrid(np.linspace(x1_min, x1_max, 100),
                     np.linspace(y1_min, y1_max, 100))
x_in = np.c_[x1.ravel(), x2.ravel()]  # shape = (10000, 2)

# Predict all the data points in the meshgrid area.
y_pred = model.predict(x_in)
```

MX-AI

## Implementation of binary classification in Logistic Regression using sklearn: Decision Boundary

- Let's use sklearn's LogisticRegression() and check the decision boundary visually.

```
# Drawing the data and decision boundary
y_pred = y_pred.reshape(x1.shape)  # shape = (100, 100)
plt.figure(figsize=(5,5))
m = ['o', '^']
color = ['red', 'blue']
for i in [0, 1]:
    idx = np.where(y == i)
    plt.scatter(x[idx, 0], x[idx, 1],
                c=color[i],
                marker = m[i],
                s = 40,
                edgecolor = 'black',
                alpha = 0.5,
                label='class-'+str(i))
plt.contour(x1, x2, y_pred, cmap=ListedColormap(['red', 'blue']), alpha=0.5)

plt.axis('tight')
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()


Result:
Accuracy of test data = 0.930
```



MX-AI

## ■ Multivariable Logistic Regression: breast cancer dataset

- Let's apply Logistic Regression to a breast cancer dataset containing multiple features (x).
- In this data, the number of features x is 30 and the target class (or label) consists of 0 and 1.
- Data normalization is applied because the size of each feature is different, and regularization is applied to prevent w for a specific feature from becoming too large.

$$\hat{y}_i = \frac{1}{1+e^{-(w_1 x_{1,i} + w_2 x_{2,i} + \dots + b)}} \rightarrow \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}} \qquad \vec{w} = [b, w_1, w_2, \dots, w_{30}] \qquad \vec{x} = [1, x_1, x_2, \dots, x_{30}]$$

| No | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | mean fractal dimension | radius error | texture error | … | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 17.99 | 10.38 | 122.80 | 1001.00 | 0.12 | 0.28 | 0.30 | 0.15 | 0.24 | 0.08 | 1.10 | 0.91 | … | 0 |
| 1 | 20.57 | 17.77 | 132.90 | 1326.00 | 0.08 | 0.08 | 0.09 | 0.07 | 0.18 | 0.06 | 0.54 | 0.73 | … | 0 |
| 2 | 19.69 | 21.25 | 130.00 | 1203.00 | 0.11 | 0.16 | 0.20 | 0.13 | 0.21 | 0.06 | 0.75 | 0.79 | … | 0 |
| 3 | 11.42 | 20.38 | 77.58 | 386.10 | 0.14 | 0.28 | 0.24 | 0.11 | 0.26 | 0.10 | 0.50 | 1.16 | … | 0 |
| 4 | 20.29 | 14.34 | 135.10 | 1297.00 | 0.10 | 0.13 | 0.20 | 0.10 | 0.18 | 0.06 | 0.76 | 0.78 | … | 0 |
| 5 | 12.45 | 15.70 | 82.57 | 477.10 | 0.13 | 0.17 | 0.16 | 0.08 | 0.21 | 0.08 | 0.33 | 0.89 | … | 0 |
| 6 | 18.25 | 19.98 | 119.60 | 1040.00 | 0.09 | 0.11 | 0.11 | 0.07 | 0.18 | 0.06 | 0.45 | 0.77 | … | 0 |
| … | … | … | … | … | … | … | … | … | … | … | … | … | … | |
| 569 | 7.76 | 24.54 | 47.92 | 181.00 | 0.05 | 0.04 | 0.00 | 0.00 | 0.16 | 0.06 | 0.39 | 1.43 | … | 1 |

MX-AI

## Implementation of multivariable Logistic Regression using scipy.optimize

```python
# [MXML-4-02] 3.bin_class(scipy_cancer).py
# Breast cancer dataset
from scipy import optimize
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt

# Read breast cancer dataset
x, y = load_breast_cancer(return_X_y=True)

# Split the data into training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Z-score normalization
# When normalzing the test data, use the mean and
# standard deviation from the training data.
x_mean = x_train.mean(axis=0).reshape(1, -1)
x_std = x_train.std(axis=0).reshape(1, -1)
x_train = (x_train - x_mean) / x_std
x_test = (x_test - x_mean) / x_std

# Add a column vector with all 1 to the feature matrix.
# [0.3, 0.4, ...] --> [1.0, 0.3, 0.4, ...]
# [0.1, 0.5, ...] --> [1.0, 0.1, 0.5, ...]
# [ ...]
x1_train = np.hstack([np.ones([x_train.shape[0], 1]), x_train])
x1_test = np.hstack([np.ones([x_test.shape[0], 1]), x_test])
```

```python
REG_CONST = 0.01 # regularization constant

# Loss function : mean of binary cross entropy
def bce_loss(W, args):
    train_x = args[0]
    train_y = args[1]
    test_x = args[2]
    test_y = args[3]

    # Calculate the loss of training data
    y_hat = 1.0 / (1 + np.exp(-np.dot(W, train_x.T)))
    train_ce = -train_y * np.log(y_hat + 1e-10) - \
                (1.0 - train_y) * np.log(1.0 - y_hat + 1e-10)
    train_loss = train_ce.mean() + REG_CONST * np.mean(np.square(W))

    # Calculate the loss of test data
    # It is independent of training and is measured later to observe
    # changes in loss.
    y_hat = 1.0 / (1 + np.exp(-np.dot(W, test_x.T)))
    test_ce = -test_y * np.log(y_hat + 1e-10) - \
               (1.0 - test_y) * np.log(1.0 - y_hat + 1e-10)
    test_loss = test_ce.mean() + REG_CONST * np.mean(np.square(W))

    # Save the loss
    trc_train_loss.append(train_loss)
    trc_test_loss.append(test_loss)

    return train_loss
```

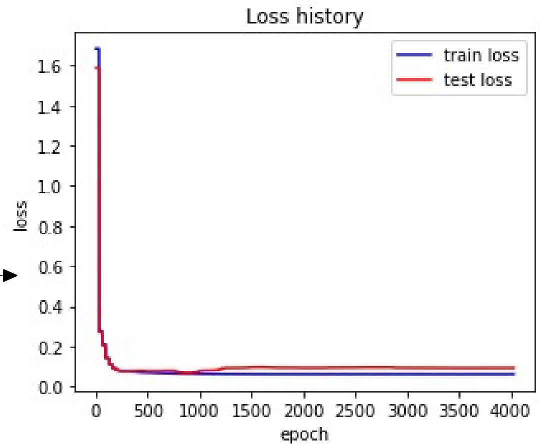
MX-AI

## Implementation of multivariable Logistic Regression using scipy.optimize

```
# Perform an optimization process
trc_train_loss = []
trc_test_loss = []
init_w = np.ones(x1_train.shape[1]) * 0.1
result = optimize.minimize(fun = bce_loss,
                           x0 = init_w,
                           args=[x1_train, y_train,\
                                 x1_test, y_test])

# print the result. result.x contains the optimal parameters
print(result)

# Measure the accuracy of test data
y_prob = 1.0 / (1 + np.exp(-np.dot(result.x, x1_test.T)))
y_pred = (y_prob > 0.5).astype('int8')
acc = (y_pred == y_test).mean()
print('\nAccuracy of test data = {:.3f}'.format(acc))

# Visually see that the loss decreases as the iteration
# progresses
plt.figure(figsize=(5, 4))
plt.plot(trc_train_loss, color='blue', label='train loss')
plt.plot(trc_test_loss, color='red', label='test loss')
plt.legend()
plt.title('Loss history')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

```
Results :

message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 0.0476923416770152
        x: [ 1.715e-01 -3.878e-01 ... -1.356e+00 -1.229e+00]
      nit: 113
      jac: [ 4.266e-06  1.231e-06 ...  2.987e-06  6.214e-06]
 hess_inv: [[ 1.040e+02 -1.642e+01 ... -1.941e+01  2.178e+01]
            [-1.642e+01  5.764e+01 ...  1.657e+01 -6.918e+01]
            ...
            [-1.941e+01  1.657e+01 ...  1.186e+02  1.016e+01]
            [ 2.178e+01 -6.918e+01 ...  1.016e+01  1.747e+02]]
     nfev: 3648
     njev: 114

Accuracy of test data = 0.979
```

As the iteration progresses, the loss of training and test data gradually decreases.



MX-AI

## ■ Implementation of multivariable Logistic Regression using sklearn

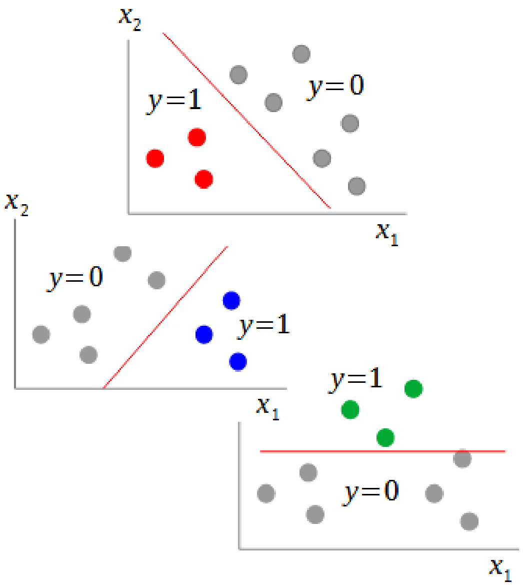
```
# [MXML-4-02] 4.bin_class(sklearn_cancer).py
# Using sklearn's LogisticRegression()
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer

# Read breast cancer dataset
x, y = load_breast_cancer(return_X_y=True)

# Split the data into training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2)

# Z-score normalization
# When normalzing the test data, use the mean and
# standard deviation from the training data.
x_mean = x_train.mean(axis=0).reshape(1, -1)
x_std = x_train.std(axis=0).reshape(1, -1)
x_train = (x_train - x_mean) / x_std
x_test = (x_test - x_mean) / x_std

# regularization constant (strength)
REG_CONST = 0.01
```

```
# Create a model and fit it to the training data.
# C: inverse of regularization strength
model = LogisticRegression(penalty='l2', C=1./REG_CONST,
                           max_iter=300)
model.fit(x_train, y_train)

# Predict the classes of test data and measure the accuracy
# of test data
y_pred = model.predict(x_test)
acc = (y_pred == y_test).mean()
print('\nAccuracy of test data = {:.3f}'.format(acc))
```

```
Result :

시험 데이터의 정확도 = 0.965
```

* Reference:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

C: float, default=1.0
**Inverse of regularization strength**; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.




# 4. Logistic Regression

## Multiclass Classification

## Part 3: One-vs-Rest (OVR)

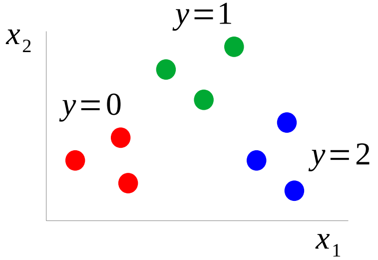
This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

## Multiclass classification – OvR (One-vs-Rest)

- Data with three classes (y = [0, 1, 2]) is one-hot encoded and then trained through three binary classification models (model-1, 2, 3).
- After training is complete, the test data is input into the three models to predict the probability that y is 0, 1, and 2, respectively, and the class with the highest probability is selected as the final class.

| | multiclass data | | | one-hot encoding | | |
|---|---|---|---|---|---|---|
| no | x1 | x2 | y | p(y=0) | p(y=1) | p(y=2) |
| 1 | 0.8 | 1.2 | 0 | 1 | 0 | 0 |
| 2 | 9.8 | 2.0 | 2 | 0 | 0 | 1 |
| 3 | 3.2 | 3.5 | 0 | 1 | 0 | 0 |
| 4 | 4.8 | 5.2 | 1 | 0 | 1 | 0 |
| 5 | 2.7 | 2.0 | 0 | 1 | 0 | 0 |
| 6 | 8.3 | 3.4 | 2 | 0 | 0 | 1 |
| 7 | 5.7 | 4.5 | 1 | 0 | 1 | 0 |
| 8 | 8.3 | 7.8 | 1 | 0 | 1 | 0 |
| … | … | … | … | … | … | … |
| N | 9.2 | 2.5 | 2 | 0 | 0 | 1 |




Perform three binary classifications

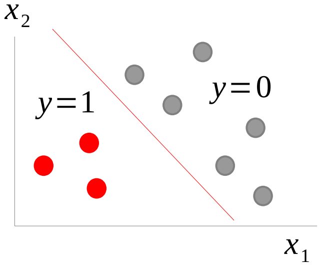
binary classification

| x1 | x2 | p(y=0) |
|---|---|---|
| 0.8 | 1.2 | 1 |
| 9.8 | 2.0 | 0 |
| 3.2 | 3.5 | 1 |
| 4.8 | 5.2 | 0 |
| 2.7 | 2.0 | 1 |
| 8.3 | 3.4 | 0 |
| 5.7 | 4.5 | 0 |
| 8.3 | 7.8 | 0 |
| … | … | … |
| 9.2 | 2.5 | 0 |

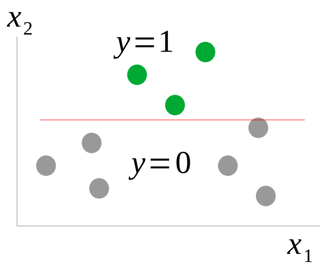
| x1 | x2 | p(y=1) |
|---|---|---|
| 0.8 | 1.2 | 0 |
| 9.8 | 2.0 | 0 |
| 3.2 | 3.5 | 0 |
| 4.8 | 5.2 | 1 |
| 2.7 | 2.0 | 0 |
| 8.3 | 3.4 | 0 |
| 5.7 | 4.5 | 1 |
| 8.3 | 7.8 | 1 |
| … | … | … |
| 9.2 | 2.5 | 0 |

| x1 | x2 | p(y=2) |
|---|---|---|
| 0.8 | 1.2 | 0 |
| 9.8 | 2.0 | 1 |
| 3.2 | 3.5 | 0 |
| 4.8 | 5.2 | 0 |
| 2.7 | 2.0 | 0 |
| 8.3 | 3.4 | 1 |
| 5.7 | 4.5 | 0 |
| 8.3 | 7.8 | 0 |
| … | … | … |
| 9.2 | 2.5 | 1 |

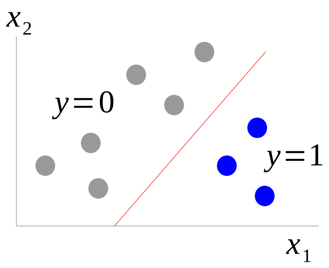
model-1




model-2




model-3

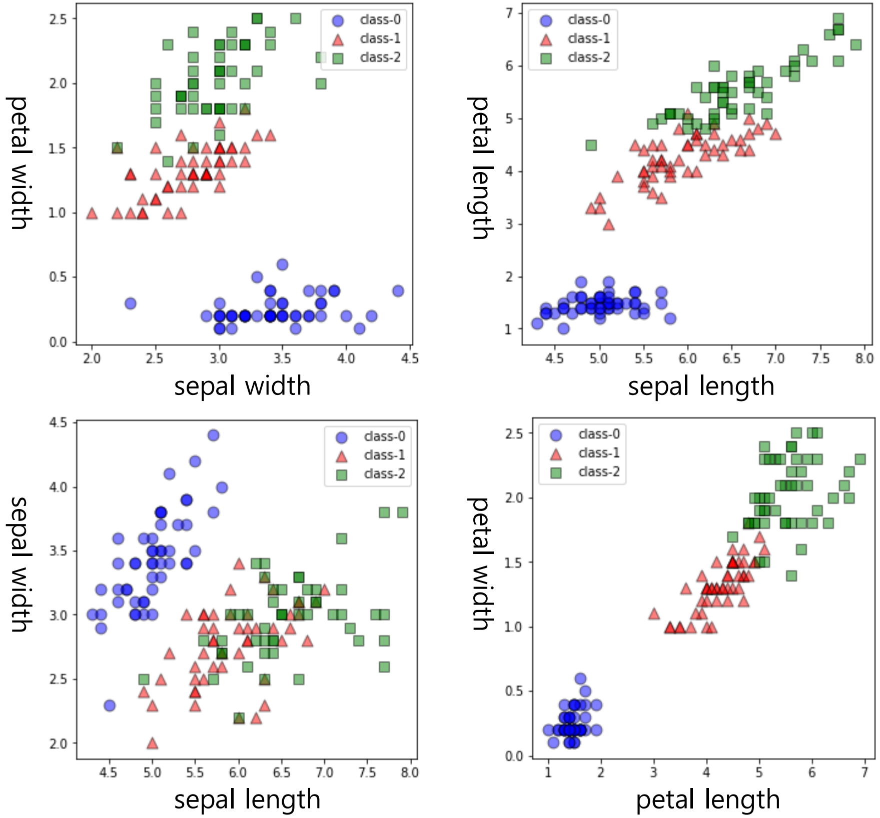


## ■ iris dataset

- The iris dataset has 4 features and 3 types of classes (multiclass).

multivariate — multiclass

| No | 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 | 1.4 | 0.2 | 0 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 50 | 7 | 3.2 | 4.7 | 1.4 | 1 |
| 51 | 6.4 | 3.2 | 4.5 | 1.5 | 1 |
| 52 | 6.9 | 3.1 | 4.9 | 1.5 | 1 |
| 147 | 6.5 | 3 | 5.2 | 2 | 2 |
| 148 | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| 149 | 5.9 | 3 | 5.1 | 1.8 | 2 |



MX-AI

## Implementation of OvR (One-vs-Rest)

- Create three binary classification models to train on multiclass Iris data and predict the classes (or labels ) for the test data.

```
# [MXML-4-03] 5.multiclass(ovr_1).py
# Multi-class classification (OvR : One-vs-Rest)
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np

# Read iris dataset
x, y = load_iris(return_X_y=True)

# one-hot encoding of the y labels.
y_ohe = OneHotEncoder().fit_transform(y.reshape(-1,1)).toarray()

# Split the data into the training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y_ohe)

# Perform the OvR. Since there are three labels, three models are used.
models = []
for m in range(y_train.shape[1]):
    # y for binary classification
    y_sub = y_train[:, m]
    models.append(LogisticRegression())
    models[-1].fit(x_train, y_sub)

# The labels of the test data are predicted using three trained models.
y_prob = np.zeros(shape=y_test.shape)

for m in range(y_test.shape[1]):
    y_prob[:, m] = models[m].predict_proba(x_test)[:, 1]
```

```
# y is predicted as the label with the highest value in y_prob.
y_pred = np.argmax(y_prob, axis=1)

# Measure the accuracy of the test data
y_true = np.argmax(y_test, axis=1)
acc = (y_true == y_pred).mean()
print('Accuracy of test data = {:.3f}'.format(acc))

# Check the estimated parameters.
for m in range(y_test.shape[1]):
    w = models[m].coef_
    b = models[m].intercept_
    print("\nModel-{}:".format(m))
    print("w:", w)
    print("b:", b)
```

```
Accuracy of test data = 0.900

Model-0:
w: [[-0.45755864  0.79917041 -2.21772626 -0.91415335]]
b: [6.70531347]

Model-1:
w: [[-0.09605778 -2.09059899  0.49517172 -0.9110781 ]]
b: [5.28401964]

Model-2:
w: [[-0.51546488 -0.25416219  2.90423561  2.04075574]]
b: [-13.6101709]
```

MX-AI

## Implementation of OvR (One-vs-Rest)

- Use the multi_class='ovr' function of sklearn's LogisticRegression.

```python
# [MXML-4-03] 6.multiclass(ovr_2).py
# Multiclass classification (OvR : One-vs-Rest)
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np

# Read iris dataset
x, y = load_iris(return_X_y=True)
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Use the multi_class='ovr' function of sklearn's LogisticRegression.
# Even if the 'ovr' is not set, multiclass classification is
# automatically performed by referring to the number of classes.
# This was explicitly set to facilitate understanding.
model = LogisticRegression(multi_class='ovr', max_iter=300)
model.fit(x_train, y_train)

# Predict the classes of the test data
y_pred = model.predict(x_test)

# Measure the accuracy of the test data
acc = (y_test == y_pred).mean()
print('\nAccuracy of test data = {:.3f}'.format(acc))
```

```python
# Check the estimated parameters.
print('\nmodel.coef_ =\n\n', model.coef_)
print('\nmodel.intercept_ =\n\n', model.intercept_)
```

```
Accuracy of test data = 0.967

model.coef_ =

 [[-0.45722088  0.82383712 -2.22131317 -0.89167802]
 [-0.28759514 -1.88491467  0.66679901 -1.12117065]
 [-0.34199158 -0.56239993  2.65778081  2.26003722]]

model.intercept_ =

 [  6.68321288   5.52451306 -13.15170356]
```

$$\hat{y}_{i,k} = \frac{\exp(w_k \cdot x_i + b_k)}{\sum_{k=0}^{C-1} \exp(w_k \cdot x_i + b_k)} \leftarrow \text{Softmax}$$

$$L_i(w,b) = \prod_{k=0}^{C-1} \hat{y}_{i,k}^{\,1(y_i=k)}$$

$$\log(L_i(w,b)) = \sum_{k=0}^{C-1} \log(\hat{y}_{i,k}^{\,1(y_i=k)})$$

$$= \sum_{k=0}^{C-1} 1(y_i=k) \cdot \log(\hat{y}_{i,k})$$

$$J(w,b) = \sum_{i=0}^{N-1} \sum_{k=0}^{C-1} y_{i,k} \cdot \log(\hat{y}_{i,k}) \leftarrow \text{Cross Entropy}$$

# 4. Logistic Regression

## Multiclass Classification

## Part 4: Multinomial (or Softmax) Regression

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

## Multinomial Logistic Regression (or Softmax Regression)

- Let's assume that the data with three classes (y=0,1,2) is trained with two binary classification models (model-1, 2) as shown below.
- Set a baseline class and perform binary classification on the remaining classes. Any class can be a baseline.
- While OvR actually requires training multiple binary models, this method estimates all parameters simultaneously with single model. We will look at this later.

multiclass data (x1, x2, y) / one-hot encoding (p(y=0), p(y=1), p(y=2))

| no | x1 | x2 | y | p(y=0) | p(y=1) | p(y=2) |
|---|---|---|---|---|---|---|
| 1 | 0.8 | 1.2 | 0 | 1 | 0 | 0 |
| 2 | 9.8 | 2.0 | 2 | 0 | 0 | 1 |
| 3 | 3.2 | 3.5 | 0 | 1 | 0 | 0 |
| 4 | 4.8 | 5.2 | 1 | 0 | 1 | 0 |
| … | … | … | … | … | … | … |
| N | 9.2 | 2.5 | 2 | 0 | 0 | 1 |

binary classification

| x1 | x2 | p(y=0) |
|---|---|---|
| 0.8 | 1.2 | 1 |
| 9.8 | 2.0 | 0 |
| 3.2 | 3.5 | 1 |
| 4.8 | 5.2 | 0 |
| … | … | … |
| 9.2 | 2.5 | 0 |

baseline class

| x1 | x2 | p(y=1) |
|---|---|---|
| 0.8 | 1.2 | 0 |
| 9.8 | 2.0 | 0 |
| 3.2 | 3.5 | 0 |
| 4.8 | 5.2 | 1 |
| … | … | … |
| 9.2 | 2.5 | 0 |

model-1

$$\log \frac{p(y=1)}{p(y=0)} = w_1 \cdot x + b_1$$

$$\frac{p(y=1)}{1 - p(y=1) - p(y=2)} = \exp(w_1 \cdot x + b_1)$$

| x1 | x2 | p(y=2) |
|---|---|---|
| 0.8 | 1.2 | 0 |
| 9.8 | 2.0 | 1 |
| 3.2 | 3.5 | 0 |
| 4.8 | 5.2 | 0 |
| … | … | … |
| 9.2 | 2.5 | 1 |

model-2

$$\log \frac{p(y=2)}{p(y=0)} = w_2 \cdot x + b_2$$

$$\frac{p(y=2)}{1 - p(y=1) - p(y=2)} = \exp(w_2 \cdot x + b_2)$$

simultaneous equations

$$p(y=1) = \frac{\exp(w_1 \cdot x + b_1)}{1 + \exp(w_1 \cdot x + b_1) + \exp(w_2 \cdot x + b_2)}$$

$$p(y=2) = \frac{\exp(w_2 \cdot x + b_2)}{1 + \exp(w_1 \cdot x + b_1) + \exp(w_2 \cdot x + b_2)}$$

$$p(y=0) = 1 - p(y=1) - p(y=2) = \frac{1}{1 + \exp(w_1 \cdot x + b_1) + \exp(w_2 \cdot x + b_2)}$$

MX-AI

- Multinomial Logistic Regression (or Softmax Regression)

  - Softmax function

$$w_0=0,\ b_0=0$$

$$p(y=1)=\frac{\exp(w_1\cdot x+b_1)}{1+\exp(w_1\cdot x+b_1)+\exp(w_2\cdot x+b_2)} = \frac{\exp(w_1\cdot x+b_1)}{\exp(w_0\cdot x+b_0)+\exp(w_1\cdot x+b_1)+\exp(w_2\cdot x+b_2)} = \frac{\exp(w_1\cdot x+b_1)}{\sum_{k=0}^{C-1}\exp(w_k\cdot x+b_k)}$$

softmax

(C : the number of classes)

$$p(y=2) = \frac{\exp(w_2\cdot x+b_2)}{1+\exp(w_1\cdot x+b_1)+\exp(w_2\cdot x+b_2)} = \frac{\exp(w_2\cdot x+b_2)}{\sum_{k=0}^{C-1}\exp(w_k\cdot x+b_k)}$$

$$p(y=0) = \frac{1}{1+\exp(w_1\cdot x+b_1)+\exp(w_2\cdot x+b_2)} = \frac{\exp(w_0\cdot x+b_0)}{\sum_{k=0}^{C-1}\exp(w_k\cdot x+b_k)}$$

MX-AI

## Loss function for Softmax Regression

- Just like we did for binary classification, we can use MLE to obtain the loss function for softmax regression. The loss function is cross entropy, which is a general expression for binary cross entropy.
- Regularization can also be applied to the cross entropy loss function.

| $y$ | $\hat{y}$ | | | $P(y\|x)=\prod_{k=0}^{C-1} \hat{y}_{i,k}^{1(y_i=k)}$ |
|---|---|---|---|---|
| | k= 0 | 1 | 2 | |
| 1 | 0.1 | 0.7 | 0.2 | → $0.1^0 \times 0.7^1 \times 0.2^0$ = 0.7 |
| 1 | 0.8 | 0.1 | 0.1 | → $0.8^0 \times 0.1^1 \times 0.1^0$ = 0.1 |
| 0 | 0.8 | 0.1 | 0.1 | → $0.8^1 \times 0.1^0 \times 0.1^0$ = 0.8 |

The larger this value, the better the prediction.

$$\hat{y}_{i,k} = \frac{\exp(w_k \cdot x_i + b_k)}{\sum_{k=0}^{C-1} \exp(w_k \cdot x_i + b_k)}$$

i : data index, k : class index
C: the number of classes

$$L_i(w,b) = \prod_{k=0}^{C-1} \hat{y}_{i,k}^{1(y_i=k)}$$

$1(y_i=k)$ : indicator function
1(true) = 1
1(false) = 0

$$\log(L_i(w,b)) = \sum_{k=0}^{C-1} \log(\hat{y}_{i,k}^{1(y_i=k)})$$

$$= \sum_{k=0}^{C-1} 1(y_i=k) \cdot \log(\hat{y}_{i,k}) : \ 1(y_i=k) \rightarrow y_{i,k}$$

$$J(w,b) = \sum_{i=0}^{N-1} \sum_{k=0}^{C-1} y_{i,k} \cdot \log(\hat{y}_{i,k})$$

N : the number of data points
A general expression for binary cross entropy.
C = 2 --> binary cross entropy.

- Objective function

$$\max_{w,b} J(w,b)$$

$$= \min_{w,b} \sum_{i=0}^{N-1} \sum_{k=0}^{C-1} [-y_{i,k} \cdot \log(\hat{y}_{i,k})]$$

$$= \min_{w,b} \sum_i CE_i$$

- Regularization

$$\min_{w,b} \sum_i CE_i + \lambda \sum_{k=0}^{C-1} |w_k| \quad \text{(Lasso)}$$

$$\min_{w,b} \sum_i CE_i + \lambda \sum_{k=0}^{C-1} w_k^2 \quad \text{(Ridge)}$$

- comparison: binary classification

$$\hat{y}_i = \frac{1}{1+\exp(-(wx_i+b))}$$

$$\min_{w,b} \sum_i BCE_i$$

$$P(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

$$= \prod_{k=0}^{1} \hat{y}_k^{1(y_i=k)}$$

Let $\begin{cases} \hat{y}_{k=0} = 1-\hat{y} \\ \hat{y}_{k=1} = \hat{y} \end{cases}$

MX-AI

## Implementation of Softmax Regression using scipy.optimize().

- Let's train and predict the iris data using Softmax and cross entropy. We will use the scipy library to better understand how it works.

```python
# [MXML-4-04] 7.multiclass(softmax_scipy).py
# Multiclass classification (Softmax regression)
from scipy import optimize
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import matplotlib.pyplot as plt
import numpy as np

# Read iris dataset.
x, y = load_iris(return_X_y=True)

# one-hot encoding of the y labels.
y_ohe = OneHotEncoder().fit_transform(y.reshape(-1,1)).toarray()

# Split the data into the training and test data.
x_train, x_test, y_train, y_test = train_test_split(x, y_ohe)

# Add a column vector with all 1 to the feature matrix.
x1_train = np.hstack([np.ones([x_train.shape[0], 1]), x_train])
x1_test = np.hstack([np.ones([x_test.shape[0], 1]), x_test])

REG_CONST = 0.01              # regularization constant
n_feature = x_train.shape[1]  # The number of features
n_class = y_train.shape[1]    # The number of classes

def softmax(z):
    s = np.exp(z) / np.sum(np.exp(z), axis=1).reshape(-1,1)
    return s
```

```python
# Loss function: mean of cross entropy
def ce_loss(W, args):
    train_x = args[0]  # shape=(112,5)
    train_y = args[1]  # shape=(112,3)
    test_x = args[2]
    test_y = args[3]
    W = W.reshape((n_class, n_feature + 1)) # shape=(3, 5)

    # Calculate the loss of training data
    z = np.dot(W, train_x.T).T                # shape=(112, 3)
    y_hat = softmax(z)
    train_ce = np.sum(-train_y * np.log(y_hat + 1e-10), axis=1)
    train_loss = train_ce.mean() + \
                 REG_CONST * np.mean(np.square(W))

    # Calculate the loss of test data
    # This has nothing to do with the training process
    # and is only meant to observe changes in loss later.
    z = np.dot(W, test_x.T).T
    y_hat = softmax(z)
    test_ce = np.sum(-test_y * np.log(y_hat + 1e-10), axis=1)
    test_loss = test_ce.mean() + \
                REG_CONST * np.mean(np.square(W))

    # Save the loss
    trc_train_loss.append(train_loss)
    trc_test_loss.append(test_loss)
    return train_loss
```

MX-AI

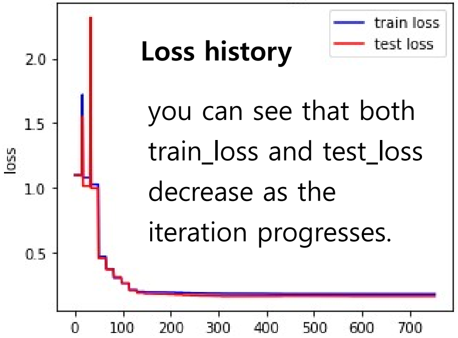
## Implementation of Softmax Regression using scipy.optimize().

- Let's train and predict the iris data using Softmax and cross entropy. We will use the scipy library to better understand how it works.

```
# Perform an optimization process
trc_train_loss = []
trc_test_loss = []
init_w = np.ones(n_class * (n_feature + 1))*0.1 # shape=(3, 5)→1D

# constraints: w0 = 0, b0 = 0
def b0_w0(w):
    n = np.arange(n_feature + 1)
    return w[n]

cons = [{'type':'eq', 'fun': b0_w0}]
result = optimize.minimize(ce_loss, init_w,
                           constraints=cons,
                           args=[x1_train, y_train, x1_test, y_test])

# print the result. result.x contains the optimal parameters
print(result)

# Measure the accuracy of test data
W = result.x.reshape(n_class, n_feature + 1)
z = np.dot(W, x1_test.T).T
y_prob = softmax(z)
y_pred = np.argmax(y_prob, axis=1)
y_true = np.argmax(y_test, axis=1)
acc = (y_pred == y_true).mean()
print('\nAccuracy of test data = {:.3f}'.format(acc))
```

```
# Visually see that the loss decreases as the iteration progresses
plt.figure(figsize=(5, 4))
plt.plot(trc_train_loss, color='blue', label='train loss')
plt.plot(trc_test_loss, color='red', label='test loss')
plt.legend()
plt.title('Loss history')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()

# Check the parameters
w = result.x.reshape((n_class, n_feature + 1))
print('\n', w)
```

```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.1782345411091662
       x: [ 0.000e+00  0.000e+00 ...
     nit: 47
     jac: [-1.664e-03 -3.386e-03 ...
    nfev: 754
    njev: 47

Accuracy of test data = 1.000

[[ 0.          0.          0.          0.          0.        ]
 [ 1.6938605  -0.2657086  -2.43369269  2.7247887  -0.34861255]
 [-2.9191233  -2.37860978 -4.03125913  5.64238059  4.55040411]]
```



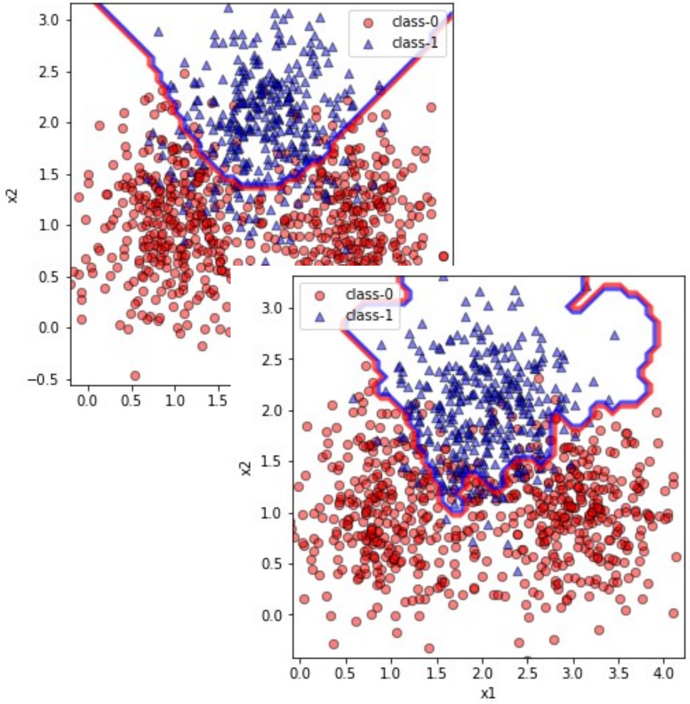
MX-AI

## Implementation of Softmax Regression using scikit-learn's LogisticRegression()..

- Let's train and predict the Iris dataset using the multi-class argument of the LogisticRegression as '**multinomial**'.

```
# [MXML-4-04] 8.multiclass(softmax_scipy).py
# Multi-class classification (Softmax regression)
# Use LogisticRegression(multi_class='multinomial')

from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Read iris dataset
x, y = load_iris(return_X_y=True)

# Split the data into the training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# Create a model and fit it to the training data.
# Use multi_class = 'multinomial'
model = LogisticRegression(multi_class='multinomial', max_iter=300)
model.fit(x_train, y_train)

# Predict the classes of the test data
y_pred = model.predict(x_test)

# Measure the accuracy
acc = (y_test == y_pred).mean()
print('Accuracy of test data = {:.3f}'.format(acc))
```

```
# Check the estimated parameters.
print('\nmodel.coef_ =\n\n', model.coef_)
print('\nmodel.intercept_ =\n\n', model.intercept_)

Results:

Accuracy of test data = 0.974

model.coef_ =

 [[-0.45791453  0.87695077 -2.30540384 -1.00408117]
  [ 0.65278864 -0.2266938  -0.27036567 -0.76413427]
  [-0.19487412 -0.65025698  2.57576951  1.76821544]]

model.intercept_ =

  [  9.48882859   1.08070931 -10.5695379 ]
```




# 4. Logistic Regression

## Locally Weighted Logistic Regression (LWLR)

## Part 5: Weighted Objective Function & Implementation of LWLR

This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

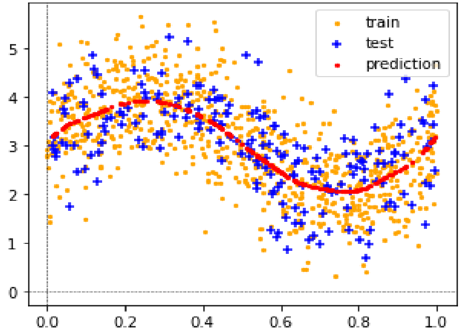
www.youtube.com/@meanxai

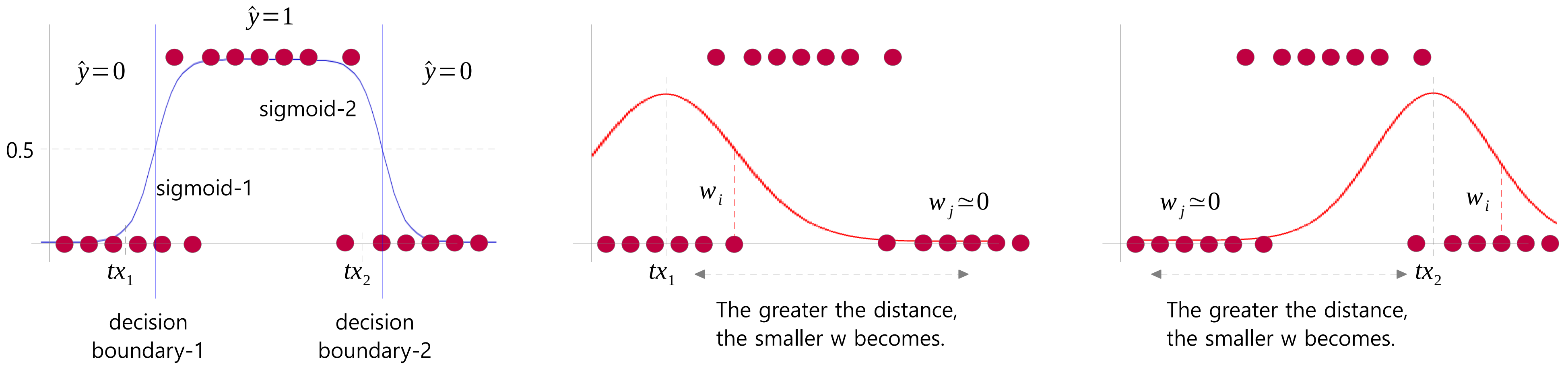
## Locally Weighted Logistic Regression (LWLR)

- The concept of LWLR is very similar to Locally Weighted Linear Regression, LWR. LWR can generate a non-linear regression curve, and LWLR can generate a non-linear decision boundary.
- It appears that two sigmoid functions are needed to classify the data in the left picture below. This will give you two decision lines. If tx1, the test data point, is on the left area, it can be predicted using the sigmoid-1, otherwise, it can be predicted using the sigmoid-2.
- Calculate the weight of each data point using a normal distribution in the same way as Locally Weighted Linear Regression, LWR. When px is on the left, the weights of the data points in the left area are large, so Logistic Regression is performed primarily using those data points.
- This is also a lazy learner.

* reference: [MXML-3-05]

Locally Weighted Regression (LWR)

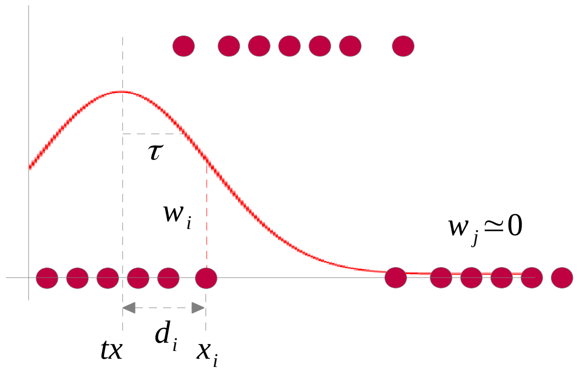


MX-AI

## Weights and Objective function

- Normal distribution is often used as a weight function (or kernel function).
- Calculate the distance d between the test data point tx and all training data points x, and calculate the weight w of each data point using the normal distribution for d.
- The closer the training data point is to tx, the larger the weight and vice versa.
- The standard deviation, tau of the normal distribution can be used to adjust the range of neighbors. The tau is a hyper-parameter.
- Weighted binary cross-entropy is used as the objective function.




- Weights

$$d_i = |tx - x_i|$$

two-dimensional distance

$$d_i = \sqrt{(tx_1 - x_{1,i})^2 + (tx_2 - x_{2,i})^2}$$

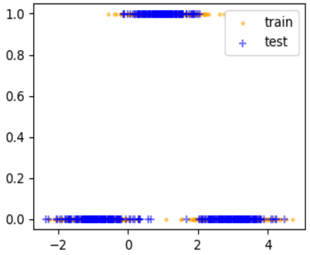
$$w_i = \exp\left(-\frac{d^2}{2\tau^2}\right) \quad \begin{cases} d_i \to 0 \;:\; w_i \to 1 \\ d_i \to \infty \;:\; w_i \to 0 \end{cases}$$

- Objective function

$$\min_{w,b} \sum_i w_i \cdot [-y_i \cdot \log \hat{y}_i - (1-y_i) \cdot \log(1-\hat{y}_i)]$$

## Implementation of LWLR using scipy.optimize

```
# [MXML-4-05] 9.lwlr(scipy).py
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
from sklearn.model_selection import train_test_split

# Generate a simple dataset
def lwlr_data1(n):
    n1 = int(n / 3)
    a = np.random.normal(-1.0, 0.5, n1)
    b = np.random.normal(1.0, 0.5, n1)
    c = np.random.normal(3.0, 0.5, n - n1 * 2)
    x = np.hstack([a, b, c]).reshape(-1, 1)
    y = np.hstack([np.zeros(n1), np.ones(n1), \
                   np.zeros(n - n1 * 2)])
    return x, y

# Generate training and test data
x, y = lwlr_data1(n=2000)
x_train, x_test, y_train, y_test = train_test_split(x, y)
x1_train = np.hstack([np.ones([x_train.shape[0], 1]), x_train])
x1_test = np.hstack([np.ones([x_test.shape[0], 1]), x_test])

# Visualize the dataset
plt.figure(figsize=(6, 3))
plt.scatter(x_train, y_train, s=5, c='orange', alpha=0.5,
            label='train')
```




```
plt.scatter(x_test, y_test, marker='+', s=30, c='blue', alpha=0.5,
            label='test')
plt.legend()
plt.show()

# Calculating the weights of training data points
# xx : training data, tx : test data
def get_weight(xx, tx, tau):
    distance = np.sum(np.square(xx - tx), axis=1)
    w = np.exp(-distance / (2 * tau * tau))
    return w

# the mean of weighted binary cross entropy
def wbce_loss(W, weight):
    y_hat = 1.0 / (1 + np.exp(-np.dot(W, x1_train.T)))
    bce = -y_train * np.log(y_hat + 1e-10) - \
          (1.0 - y_train) * np.log(1.0 - y_hat + 1e-10)
    bce *= weight
    return bce.mean()

y_prob = []
for tx in x1_test:
    weight = get_weight(x_train, tx, 0.6)
    result = optimize.minimize(wbce_loss, [0.1, 0.1], args=weight)
    y_prob.append(1.0 / (1 + np.exp(-np.dot(result.x, tx.T))))

y_prob = np.array(y_prob).reshape(-1,)
```

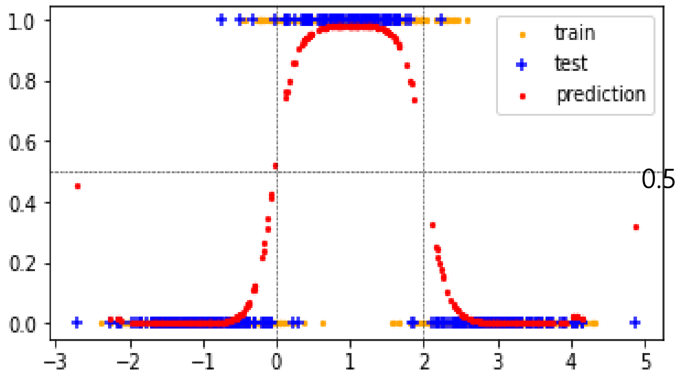
$$\frac{1}{N}\sum_i w_i \cdot [-y_i \cdot \log \hat{y}_i - (1-y_i) \cdot \log(1-\hat{y}_i)]$$

MX-AI

## ■ Implementation of LWLR using scipy.optimize

```
# Visually check the training and test data,
# and the predicted probability.
plt.figure(figsize=(6, 3))
plt.scatter(x_train, y_train, s=5, c='orange', label='train')
plt.scatter(x_test, y_test, marker='+', s=30, c='blue',
            label='test')
plt.scatter(x_test, y_prob, s=5, c='red', label='prediction')
plt.legend()
plt.axhline(y=0.5, ls='--', lw=0.5, c='black')
plt.axvline(x=0, ls='--', lw=0.5, c='black')
plt.axvline(x=2, ls='--', lw=0.5, c='black')
plt.show()

# Measure the accuracy of the test data
y_pred = (y_prob > 0.5).astype('int8')
acc = (y_pred == y_test).mean()
print('\nAccuracy of the test data = {:.3f}'.format(acc))
```

```
Results :

Accuracy of the test data = 0.980
```




Two sigmoids and two decision boundaries appear.

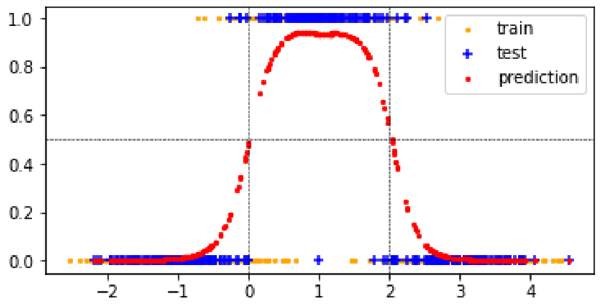
 

MX-AI

■ Implementation of LWLR using scikit-learn

```
# [MXML-4-05] 10.lwlr(sklearn).py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Generate training and test data
x, y = lwlr_data1(n=2000)
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Visualize the dataset
plt.figure(figsize=(6, 3))
plt.scatter(x_train, y_train, s=5, c='orange', label='train')
plt.scatter(x_test, y_test, marker='+', s=30, c='blue', label='test')
plt.legend()
plt.show()

# Calculating the weights of training data points
# xx : training data, xx : test data
def get_weight(xx, tx, tau):
    distance = np.sum(np.square(xx - tx), axis=1)
    w = np.exp(-distance / (2 * tau * tau))
    return w

y_prob = []
for tx in x_test:
    weight = get_weight(x_train, tx, 0.6)
    model = LogisticRegression()
    model.fit(x_train, y_train, sample_weight = weight)
    y_prob.append(model.predict_proba(tx.reshape(-1, 1))[:, 1])
y_prob = np.array(y_prob).reshape(-1,)
```

```
# Visually check the training and test data,
# and predicted probability.
plt.figure(figsize=(6, 3))
plt.scatter(x_train, y_train, s=5, c='orange', label='train')
plt.scatter(x_test,y_test,marker='+',s=30,c='blue',label='test')
plt.scatter(x_test, y_prob, s=5, c='red', label='prediction')
plt.legend()
plt.axhline(y=0.5, ls='--', lw=0.5, c='black')
plt.axvline(x=0, ls='--', lw=0.5, c='black')
plt.axvline(x=2, ls='--', lw=0.5, c='black')
plt.show()

# Measure the accuracy of the test data
y_pred = (y_prob > 0.5).astype('int8')
acc = (y_pred == y_test).mean()
print('\nAccuracy of the test data = {:.3f}'.format(acc))

Accuracy of the test data = 0.965
```



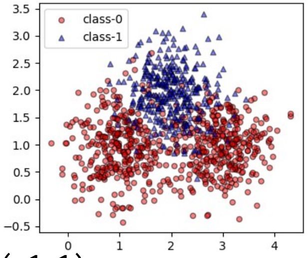

 

MX-AI

## ■ Implementation of LWLR for multivariable data set.

- Create a nonlinear decision boundary using LWLR, and observe the change in decision boundary as tau changes.

```
# [MXML-4-05] 11.lwlr_2(sklearn).py
# Check the non-linear decision boundary
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Generate a simple dataset
def lwlr_data2(n, s):
    n1 = int(n /  3)
    x, y = [], []
    for a, b, c, m in [(1, 1, 0, n1),
                       (2, 2, 1, n-n1*2),
                       (3, 1, 0, n1)]:
        x1 = np.random.normal(a, s, m).reshape(-1,1)
        x2 = np.random.normal(b, s, m).reshape(-1,1)
        x.extend(np.hstack([x1, x2]))
        y.extend(np.ones(m) * c)
    x = np.array(x).reshape(-1, 2)
    y = np.array(y).astype('int8').reshape(-1, 1)
    return x, y.reshape(-1,)
x, y = lwlr_data2(n=1000, s=0.5)

# Visually check the data distribution.
m = ['o', '^']
color = ['red', 'blue']
plt.figure(figsize=(5,5))
for i in [0, 1]:
```




```
    idx = np.where(y == i)
    plt.scatter(x[idx, 0], x[idx, 1],
                c=color[i],
                marker = m[i],
                s = 40,
                edgecolor = 'black',
                alpha = 0.5,
                label='class-'+str(i))
plt.legend()
plt.show()

# Split the data into the training and test data
x_train, x_test, y_train, y_test = train_test_split(x, y)

# Calculating the weights of training data points
# xx : training data, xx : test data
def get_weight(xx, tx, tau):
    distance = np.sum(np.square(xx - tx), axis=1)
    w = np.exp(-distance / (2 * tau * tau))
    return w

# Predict the classes of the test data
y_prob = []
tau = 1.0
for tx in x_test:
    weight = get_weight(x_train, tx, tau)
    model = LogisticRegression()
    model.fit(x_train, y_train, sample_weight = weight)
    y_prob.append(model.predict_proba(tx.reshape(-1, 2))[:, 1])
y_prob = np.array(y_prob).reshape(-1,)
```

MX-AI

## Implementation of LWLR for multivariable data set.

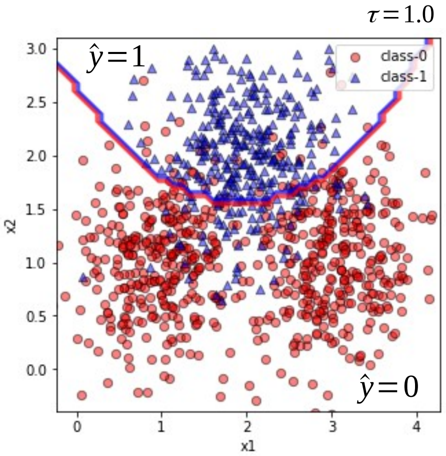
- Create a nonlinear decision boundary using LWLR, and observe the change in decision boundary as tau changes.

```
# Measure the accuracy of the test data
y_pred = (y_prob > 0.5).astype('int8')
acc = (y_pred == y_test).mean()
print('\nAccuracy of the test data = {:.3f}'.format(acc))

# Visualize the non-linear decision boundary
# reference :
# https://psrivasin.medium.com/plotting-decision-boundaries-using
#         -numpy-and-matplotlib-f5613d8acd19
x_min, x_max = x_test[:, 0].min() - 0.1, x_test[:,0].max() + 0.1
y_min, y_max = x_test[:, 1].min() - 0.1, x_test[:, 1].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 50),
                     np.linspace(y_min, y_max, 50))
x_in = np.c_[xx.ravel(), yy.ravel()]

# Predict the classes of the data points in the x_in variable.
y_prob = []
for tx in x_in:
    weight = get_weight(x_train, tx, tau)
    model = LogisticRegression()
    model.fit(x_train, y_train, sample_weight = weight)
    y_prob.append(model.predict_proba(tx.reshape(-1, 2))[:, 1])

y_prob = np.array(y_prob).reshape(-1,)
y_pred = (y_prob > 0.5).astype('int8')

# Draw the decision boundary
y_pred = np.round(y_pred).reshape(xx.shape)
```

```
plt.figure(figsize=(5, 5))
for i in [0, 1]:
    idx = np.where(y == i)
    plt.scatter(x[idx, 0], x[idx, 1],
                c=color[i],
                marker = m[i],
                s = 40,
                edgecolor = 'black',
                alpha = 0.5,
                label='class-' + str(i))
plt.contour(xx, yy, y_pred, cmap=ListedColormap(['red', 'blue']),
            alpha=0.5)
plt.axis('tight')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```

```
Results:

Accuracy of the test data = 0.920
```

non-linear decision boundary



MX-AI

- Implementation of LWLR for multivariable data set.
  - Create a nonlinear decision boundary using LWLR, and observe the change in decision boundary as tau changes.

$$w_i = \exp\left(-\frac{d^2}{2\tau^2}\right)$$

$\tau = 0.1$ Accuracy = 0.880 $\tau = 0.5$ Accuracy = 0.900 $\tau = 2.0$ Accuracy = 0.845 $\tau = 10.0$ Accuracy = 0.820