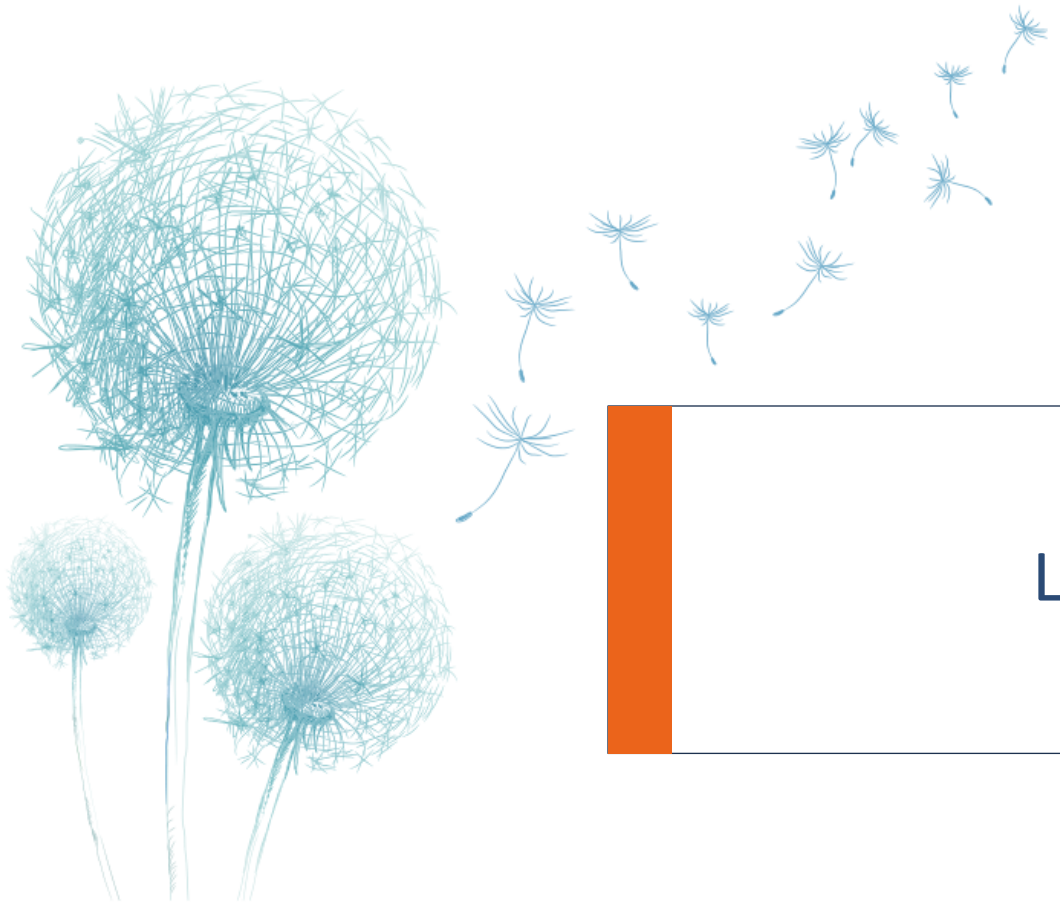
Neural networks and deep learning



Logistic Regression

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Logistic Regression

Logistic Regression

- Predict results on a binary outcome variable
  - E.g.,
    - Whether or not a patient has a disease
    - Whether a facial image is male or female
  - The outcome is not continuous or distributed normally

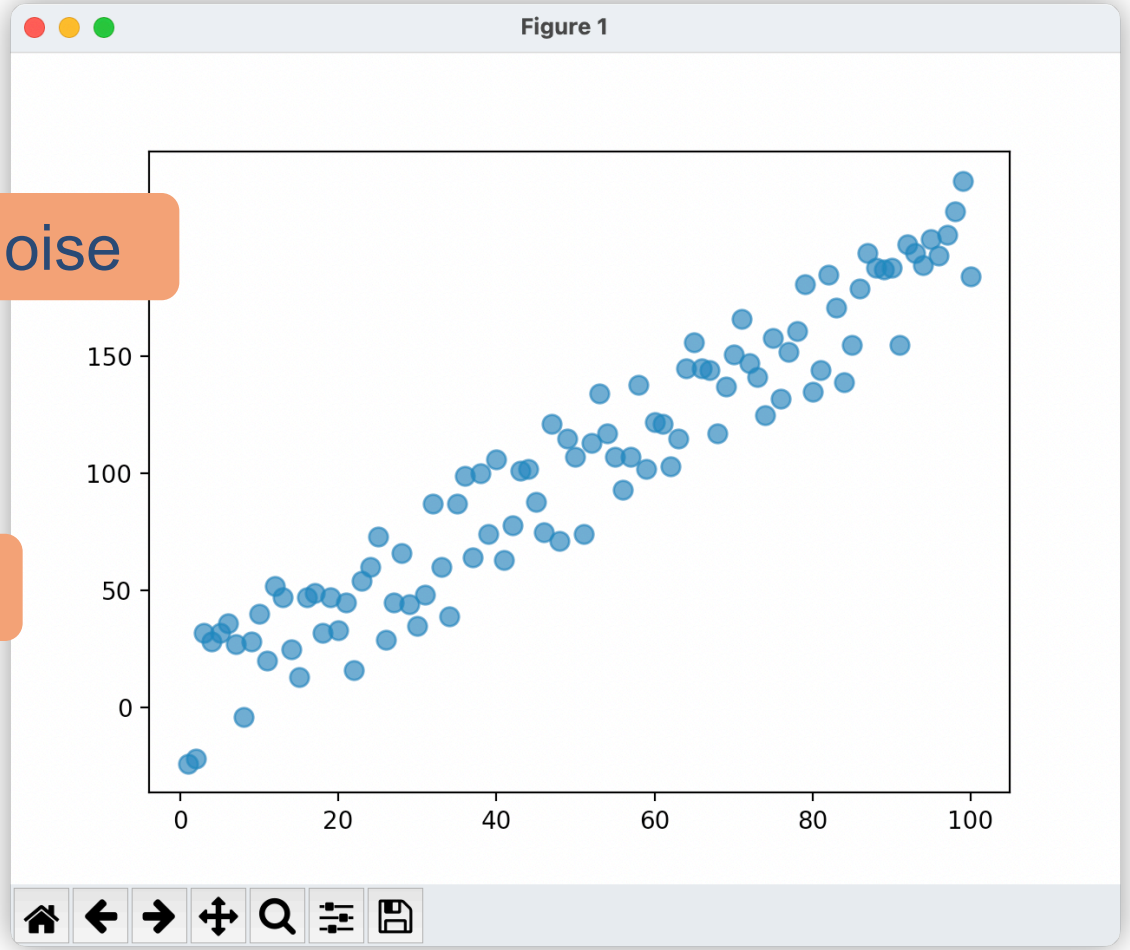
Linear Regression

https://github.com/kevinsuo/CS7357/blob/master/logistic-regression/lr-example1.py

```python
from numpy.random import randint
# Import the random number generation function in the numpy library

import matplotlib.pyplot as plt
# Import plot library

import numpy as np

X = np.array([x for x in range(1, 101)]).reshape(-1, 1)
# X = 1,2,...,n


y = np.array(list(map(lambda x: 2 * x + randint(-30, 30), X)))
# y=2X+r, r=random(-30,+30)

y_true = np.array(list(map(lambda x: 2 * x, X)))
# y=2x

#plt.plot(X, y_true, c='b')
# draw a line graph

plt.scatter(X, y, alpha=0.7, s=60)
# draw the points

#plt.title('Random Scatter')
plt.show()
```

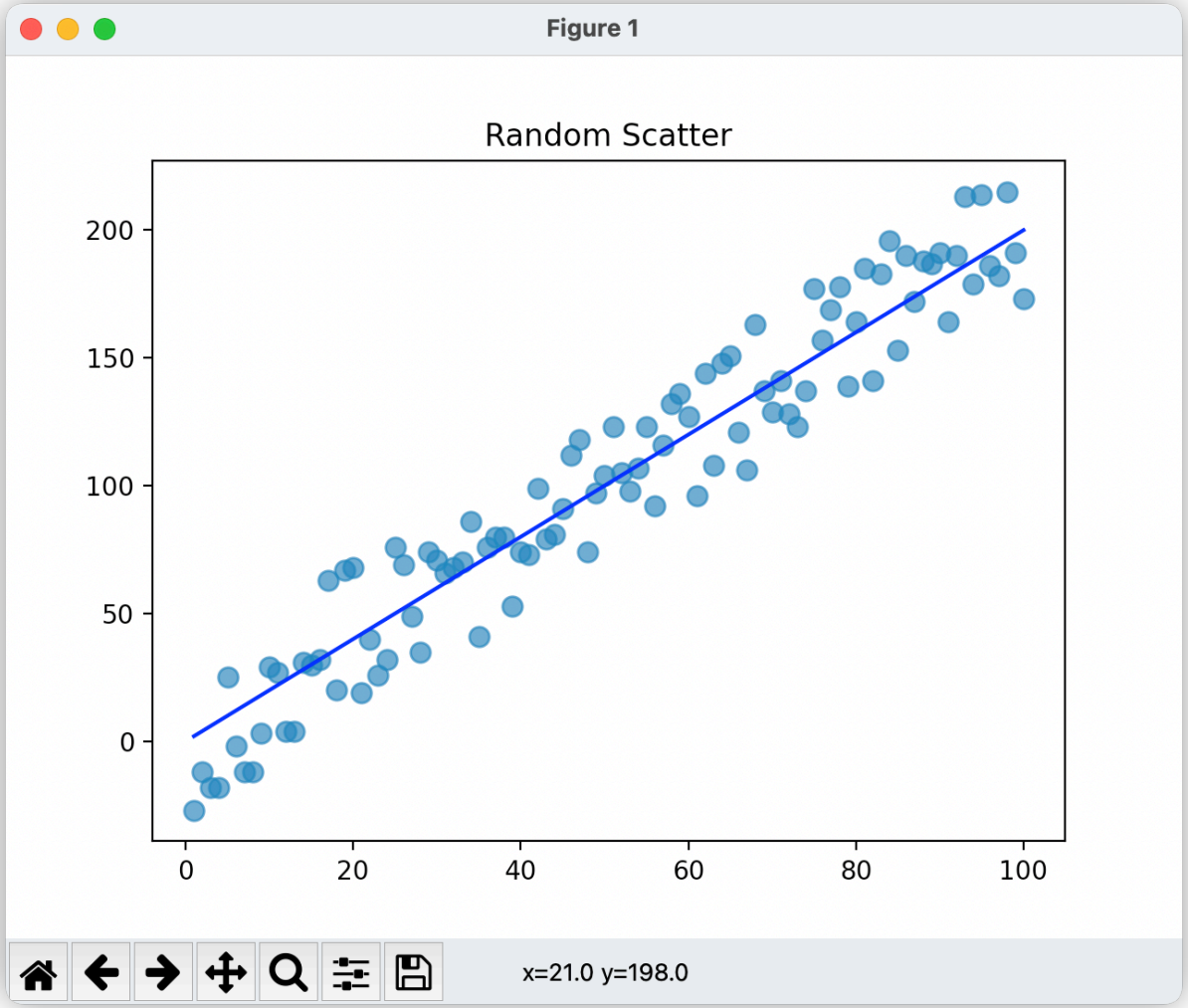
Add noise

y=2x



Linear Regression

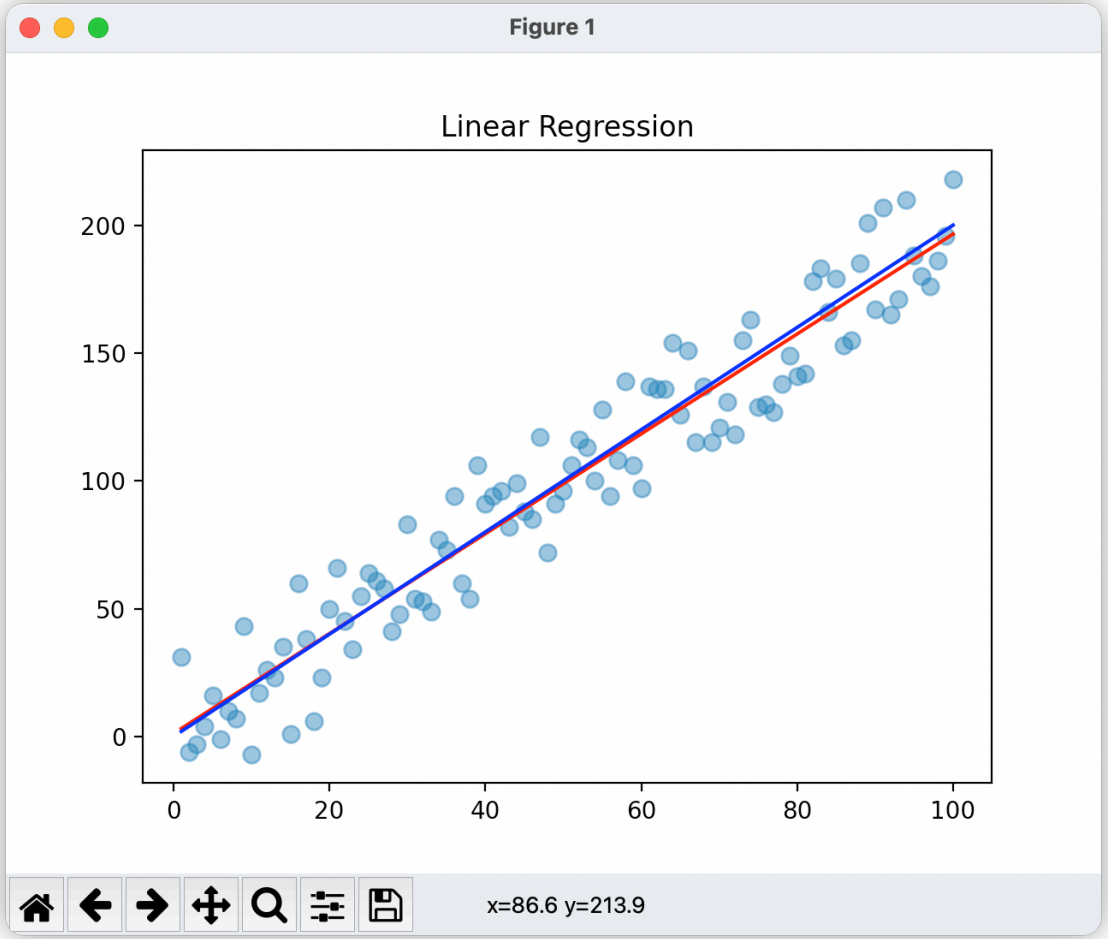
```python
from numpy.random import randint
# Import the random number generation function in the numpy library

import matplotlib.pyplot as plt
# Import plot library

import numpy as np

X = np.array([x for x in range(1, 101)]).reshape(-1, 1)
# X = 1,2,...,n


y = np.array(list(map(lambda x: 2 * x + randint(-30, 30), X)))
# y=2X+r, r=random(-30,+30)

y_true = np.array(list(map(lambda x: 2 * x, X)))
# y=2x

plt.plot(X, y_true, c='b')
#draw a line graph

plt.scatter(X, y, alpha=0.7, s=60)
# draw the points

plt.title('Random Scatter')
plt.show()
```



Linear Regression

The blue line in the figure is y=2x, and the red line is the fitted straight line obtained after regression

```python
from numpy.random import randint
# Import the random number generation function in the numpy library

import matplotlib.pyplot as plt
# Import plot library

import numpy as np

X = np.array([x for x in range(1, 101)]).reshape(-1, 1)
# X = 1,2,...,n

y = np.array(list(map(lambda x: 2 * x + randint(-30, 30), X)))
# y=2X+r, r=random(-30,+30)

y_true = np.array(list(map(lambda x: 2 * x, X)))
# y=2x

from sklearn.linear_model import LinearRegression
# Import the linear regression module in the sklearn library

lr = LinearRegression() # define a linear regression model
lr.fit(X, y) # Fit the model to the data
y_pred = lr.predict(X)

plt.scatter(X, y, alpha=0.5, s=50)
plt.plot(X, y_pred, c='r')
plt.plot(X, y_true, c='b')
plt.title('Linear Regression')
plt.show()
```



https://github.com/kevinsuo/CS7357/blob/master/logistic-regression/lr-example2.py

A problem with linear regression

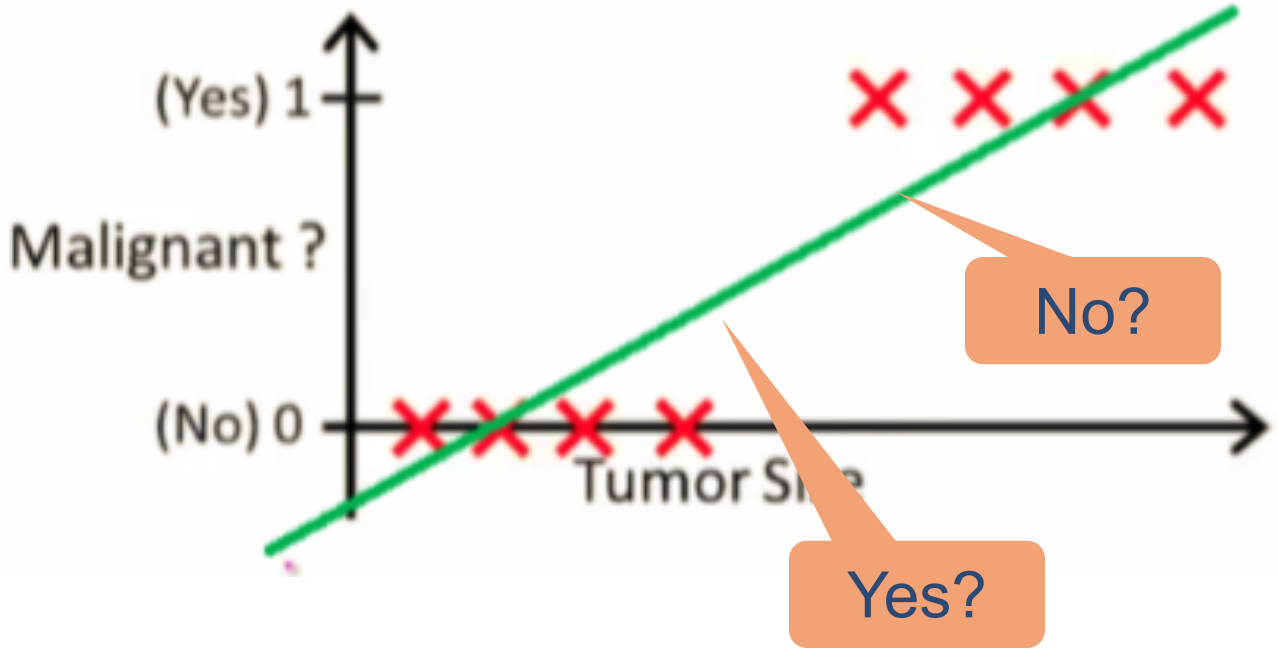
- Unlike linear regression, logistic regression is mainly used to solve classification problems.

- Can linear regression do the same thing?

A problem with linear regression

- Example: the judgment of malignant tumors and benign tumors

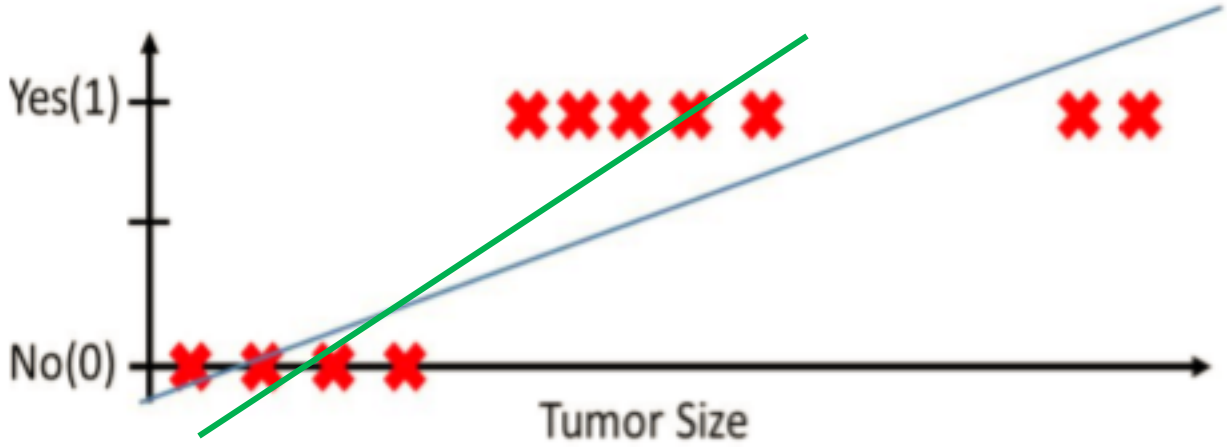
A size smaller than a certain value is benign, otherwise it is malignant



A problem with linear regression

- Example: the judgment of malignant tumors and benign tumors

"Noise" has a particularly large impact on linear equations and will greatly reduce classification accuracy



A problem with linear regression

- The line seems to oversimplify the relationship
- It gives predictions that cannot be observable values of Y for extreme values of X.
- The approach is analogous to fitting a linear model to the probability of the event, but now we need 1 or 0

Logistic regression

- Logistic regression and linear regression are similar. The biggest difference is that logistic regression is used for ***classification***.
  - Linear regression is to find a line that fits all points in the space
    - Linear regression can help us predict student scores
  - The essence of logistic regression is the same as linear regression, but it adds one step. Logistic regression uses the ***sigmoid function*** to convert the output of linear regression to return the probability value, and then the probability value can be mapped to two or more discrete classes
    - Logistic regression can help predict whether a student will pass

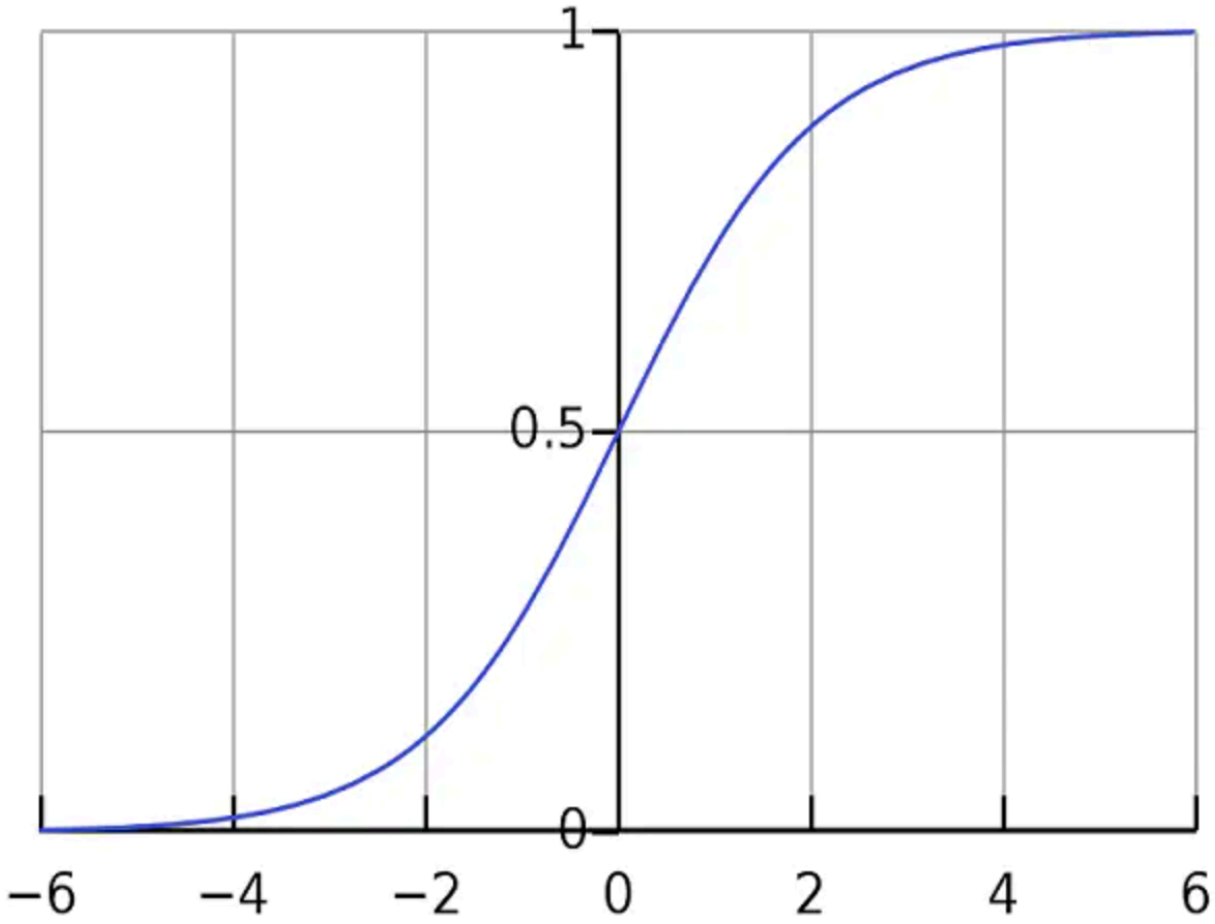
Sigmoid Function

$$g(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid Function can map linear values to the range [0-1].

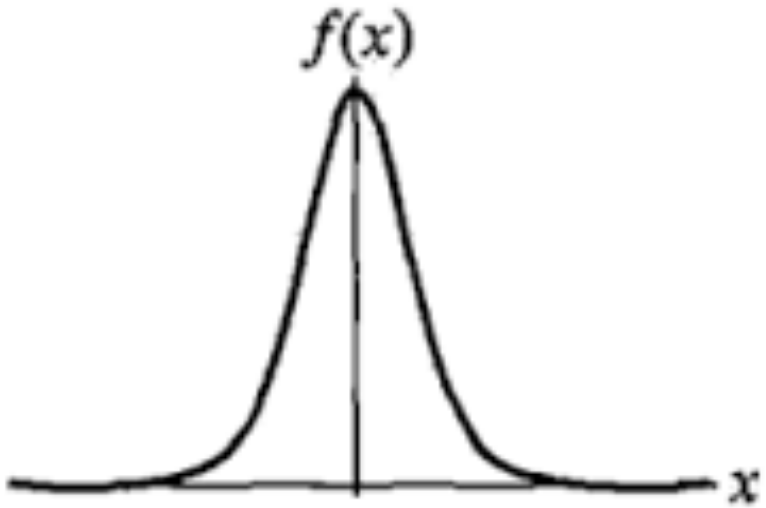
If the mapping result is less than 0.5, it is considered a negative sample.

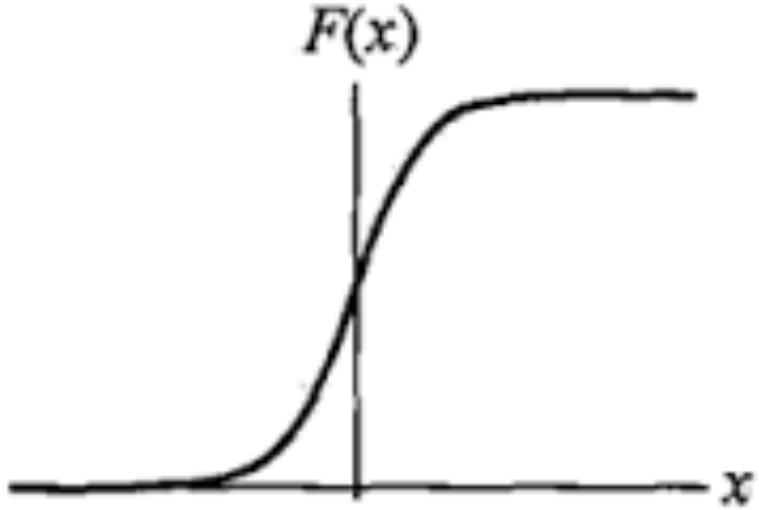
If it is greater than 0.5, it is considered a positive sample.



Sigmoid Function

- Its density function and distribution function





The shape of the logistic distribution is similar to that of the normal distribution, but the tail of the logistic distribution is longer

Sigmoid Function

$$g(x) = \frac{1}{1 + e^{-x}}$$

The mails are classified into spam mails and normal mails.

When the Sigmoid Function is calculated, it is less than 0.5, it is considered as a spam mail, and greater than 0.5 is a non-spam mail

Logistic regression

- Linear regression

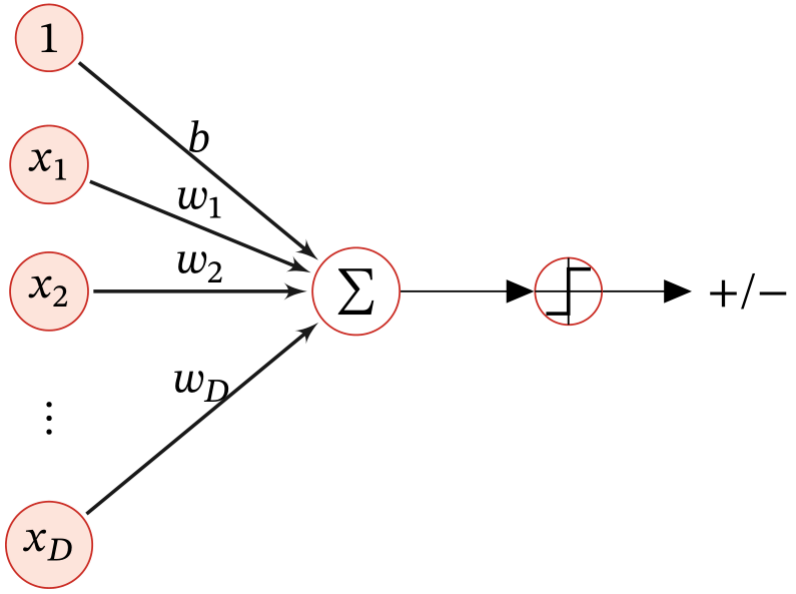
$$z = \theta_0 + \theta_1 \cdot x_1 + \theta \cdot x_2 + \cdots$$

- Logistic regression

$$h = g(z) = \frac{1}{1 + e^{-z}}$$

Logistic regression

$$g(f(\boldsymbol{x};\boldsymbol{w})) = \begin{cases} 1 & \text{if} \quad f(\boldsymbol{x};\boldsymbol{w}) > 0 \\ 0 & \text{if} \quad f(\boldsymbol{x};\boldsymbol{w}) < 0 \end{cases}$$



Logistic regression

- Linear regression
  - A house is 1,250 square feet in size; it has 3 bedrooms; it is built in 2005; …etc., how much the house can be sold?

- Logistic regression
  - A house is 1,250 square feet in size; it has 3 bedrooms; it is built in 2005; …etc., is it a good house?

Cost function of logistic regression

- Cost of x_i

$$y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))$$

$y^{(i)}$ refers to the predicted result

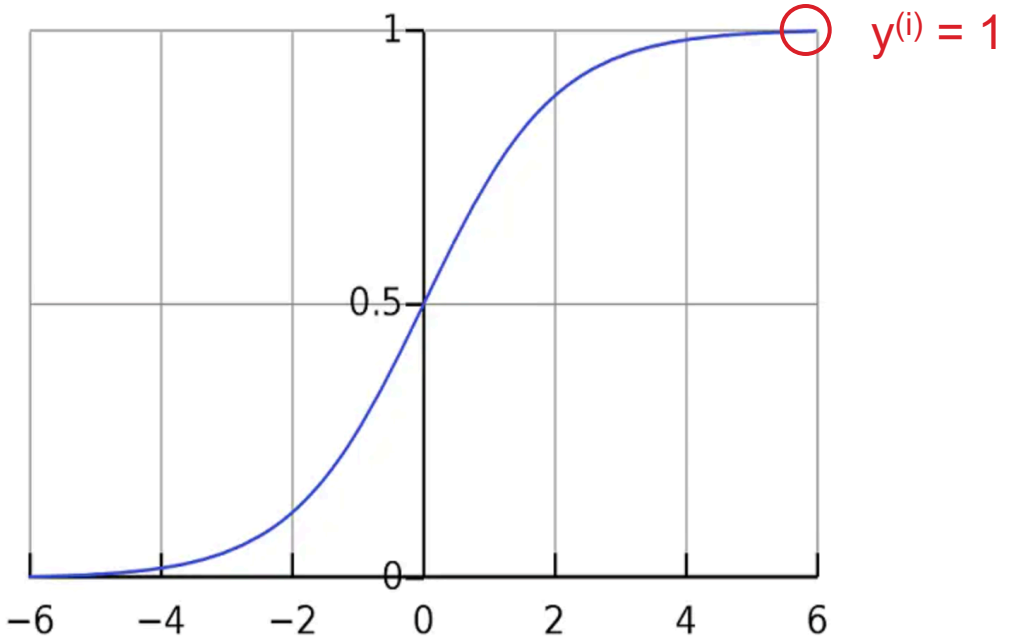
$h_\theta(x^{(i)})$ refers to the original value of x_i

Cost function of logistic regression

- Cost of x_i: Case1: suppose x_i's original value $h_\theta(x^{(i)})=1$, predicted result $y^{(i)}$ is 1

$$y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) = 0$$

It means no cost if predicted value is the same as the original value

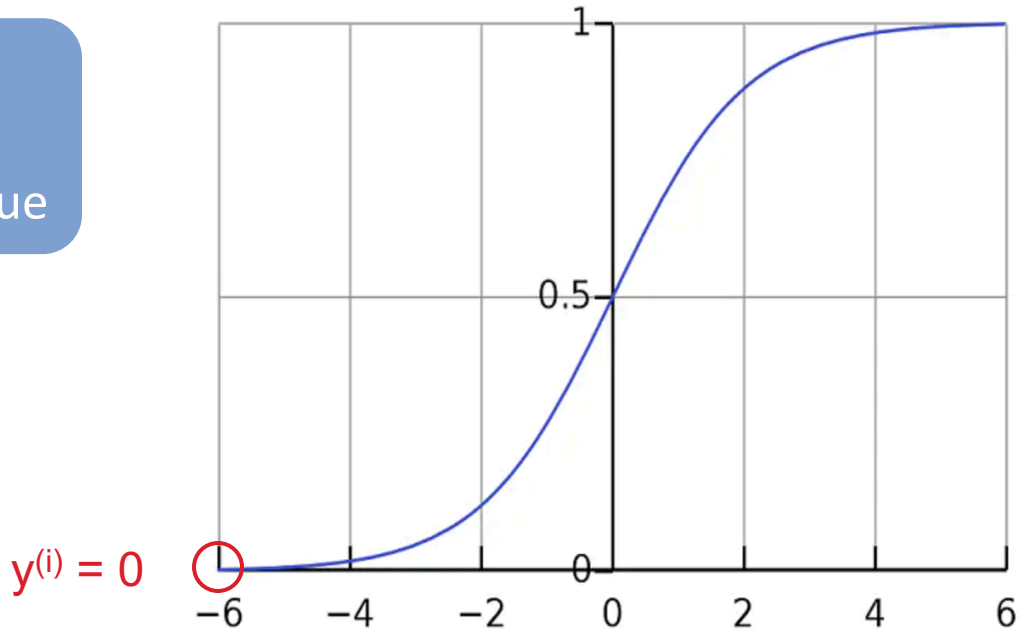


Cost function of logistic regression

- Cost of x_i: Case2: suppose x_i's original value $h_\theta(x^{(i)})=0$, predicted result $y^{(i)}$ is 0

$$y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) = 0$$

It means no cost if predicted value is the same as the original value



Cost function of logistic regression

- Cost of x_i:

$$y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))$$

The more consistent the original value is with the predicted result, the smaller the loss function will be; otherwise, the loss function will be larger.

Cost function of logistic regression

- Cost of all x_i (i=1,2,3,…)

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

- How to get it?

Assume:

$$P\ (\ y = 1 \mid x\ ;\theta\ ) = h_\theta(\ x\ )$$

$$P\ (\ y = 0 \mid x\ ;\theta\ ) = 1 - h_\theta(\ x\ )$$

$$h\ (\ x\ ) = g\ (\ z\ ) = g\ (\ \theta^T x\ )$$

Combine the two together as:

$$p\ (\ y \mid x\ ;\theta\ ) = (\ h_\theta(\ x\ )\ )^y (\ 1 - h_\theta(\ x\ )\ )^{1\ -\ y} \qquad (\ .3\ .\ .1\ )$$

Cost function of logistic regression

So we can get the likelihood function:

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}; \theta)$$
$$= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x))^{1 - y^{(i)}} \quad (.3.2)$$

So take the natural logarithm of both sides at the same time to get:

$$l(\theta) = \log L(\theta)$$
$$= \sum_{i=1}^{m} \left[y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log (1 - h(x^{(i)})) \right] \quad (3.3)$$

Cost function of logistic regression

- Find the maximum value of the above formula, introduce the factor $-\frac{1}{m}$, and convert it to find the minimum value of the following formula:

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

Cost function of logistic regression

- Cost of all x_i (i=1,2,3,…)

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

- Minimize cost function

$$J = \frac{-1}{m} \cdot [\sum_{i=1}^{m} y_i \cdot \log h_i + (1 - y_i) \cdot \log 1 - h_i]$$

$$\frac{\partial J}{\partial \theta_n} = \frac{-1}{m} \cdot [\sum_{i=1}^{m} \frac{y_i}{h_i} \cdot h_i^2 \cdot x_n \cdot \frac{1 - h_i}{h_i} + \frac{1 - y_i}{1 - h_i} \cdot -h_i^2 \cdot x_n \cdot \frac{1 - h_i}{h_i}]$$

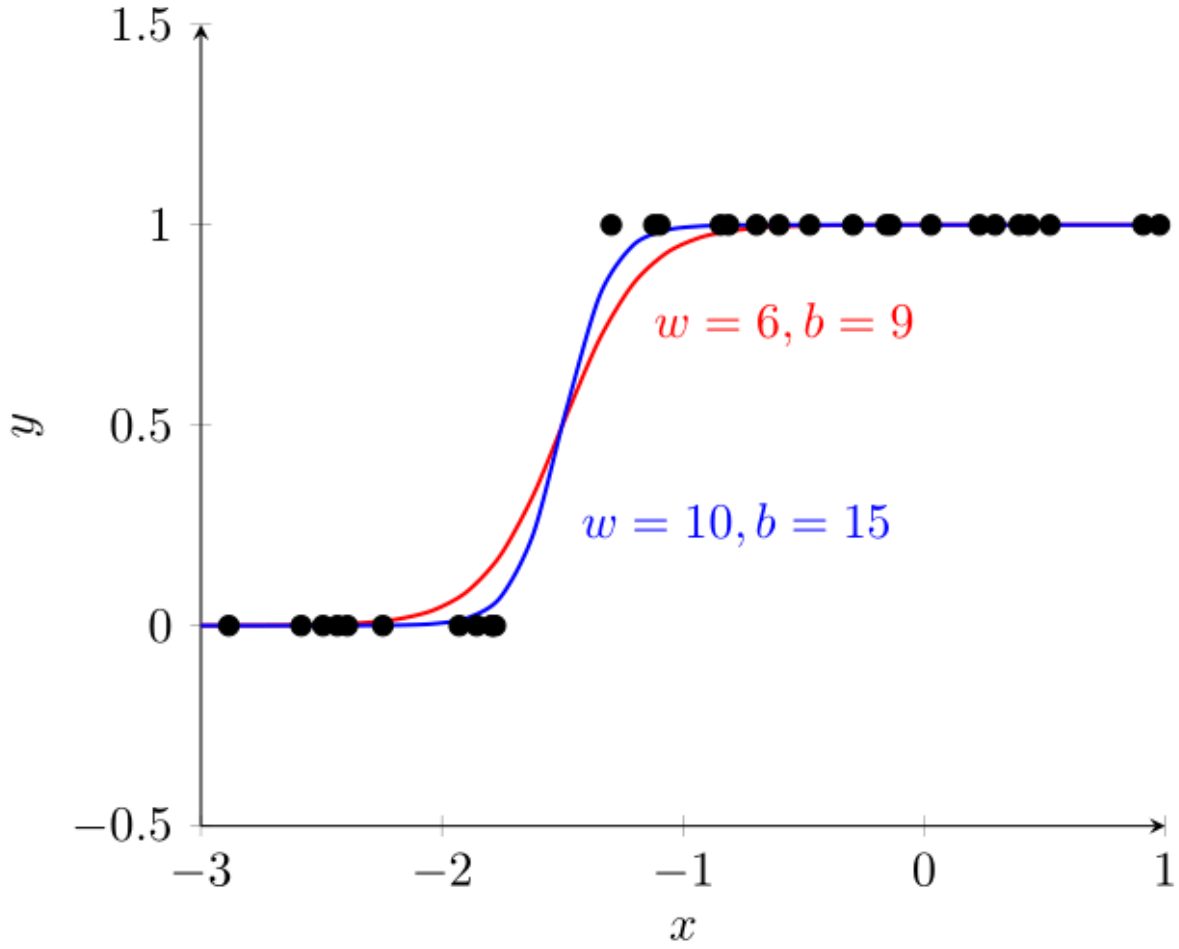
partial derivatives

$$\frac{\partial J}{\partial \theta_n} = \frac{-1}{m} \cdot [\sum_{i=1}^{m} x_n \cdot (1 - h_i) \cdot y_i - x_n \cdot h_i \cdot (1 - y_i)]$$

$$\frac{\partial J}{\partial \theta_n} = \frac{1}{m} \cdot x_i \cdot [\sum_{i=1}^{m} h_i - y_i]$$

Find w, b

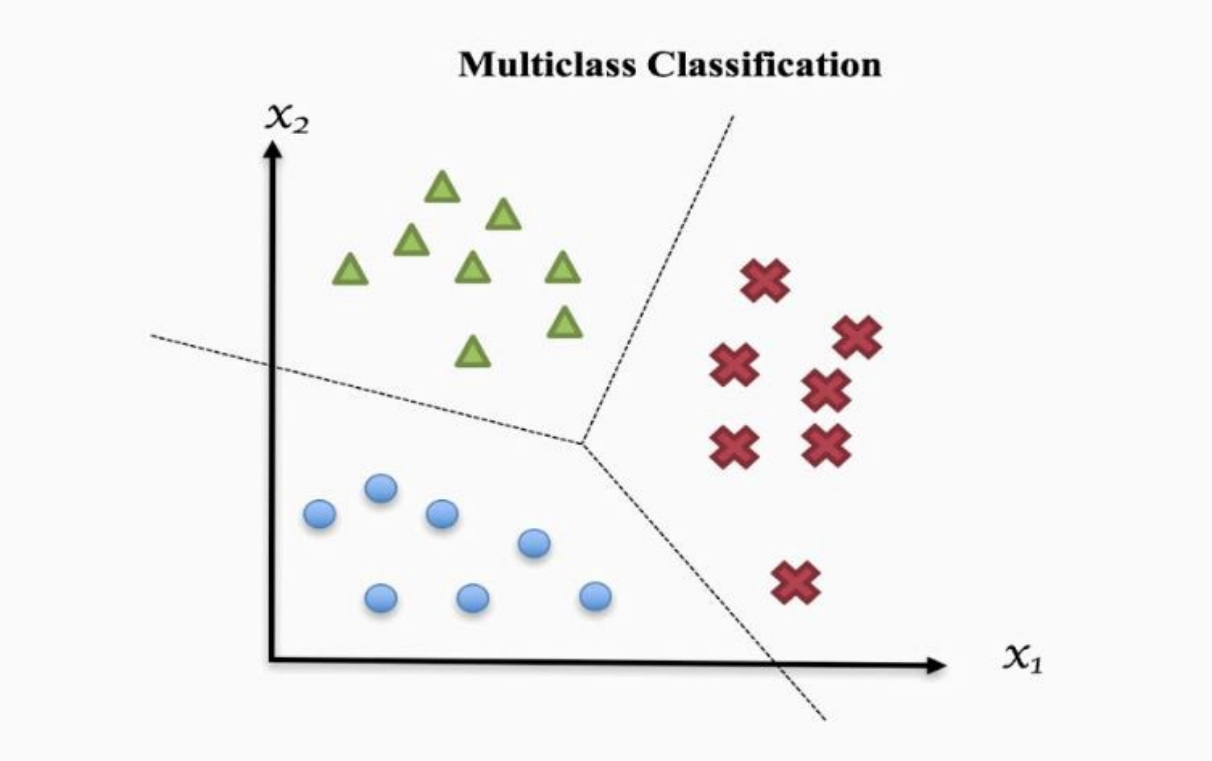
Logistic regression



Multi-class Classification

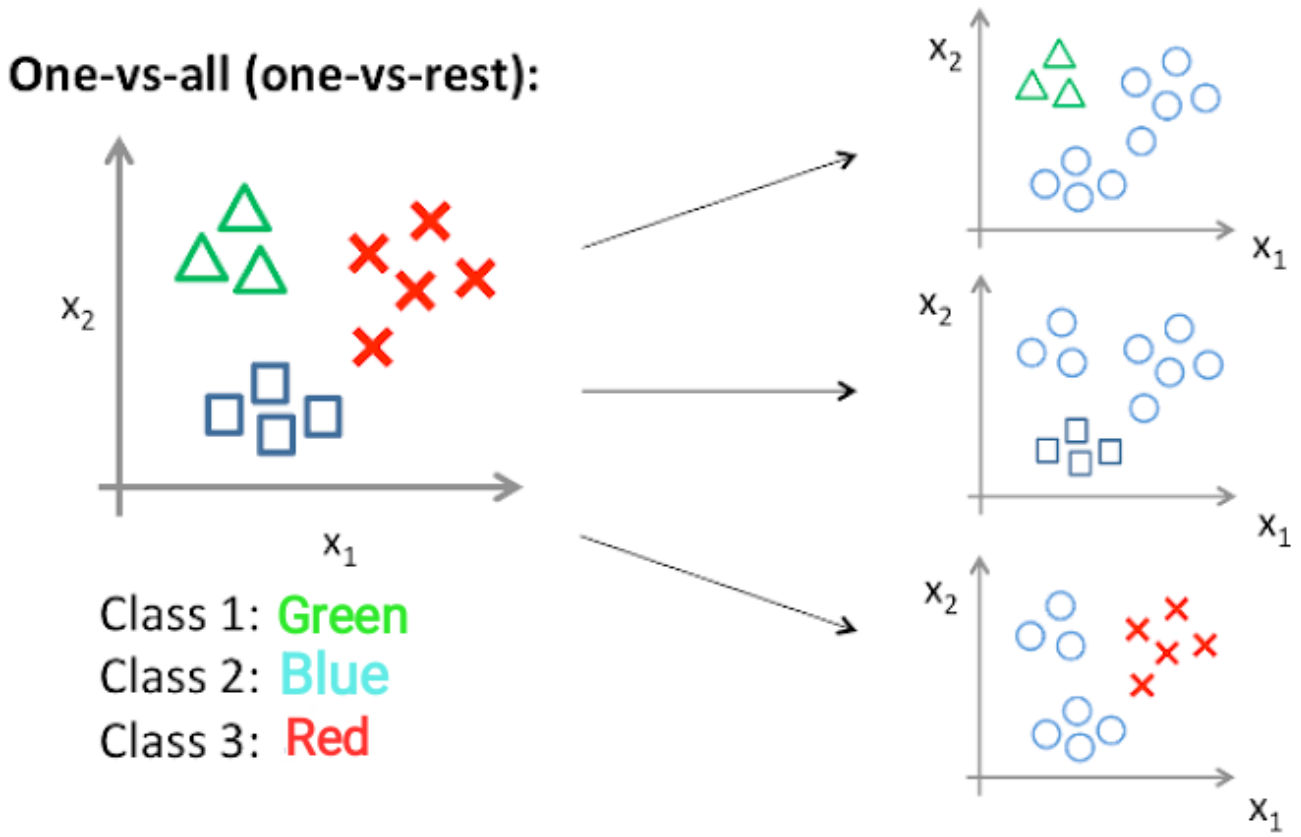
Multi-class Classification

- Logistic regression is usually only able to perform binary classifications. Is there a way to make logistic regression achieve multiple classifications?



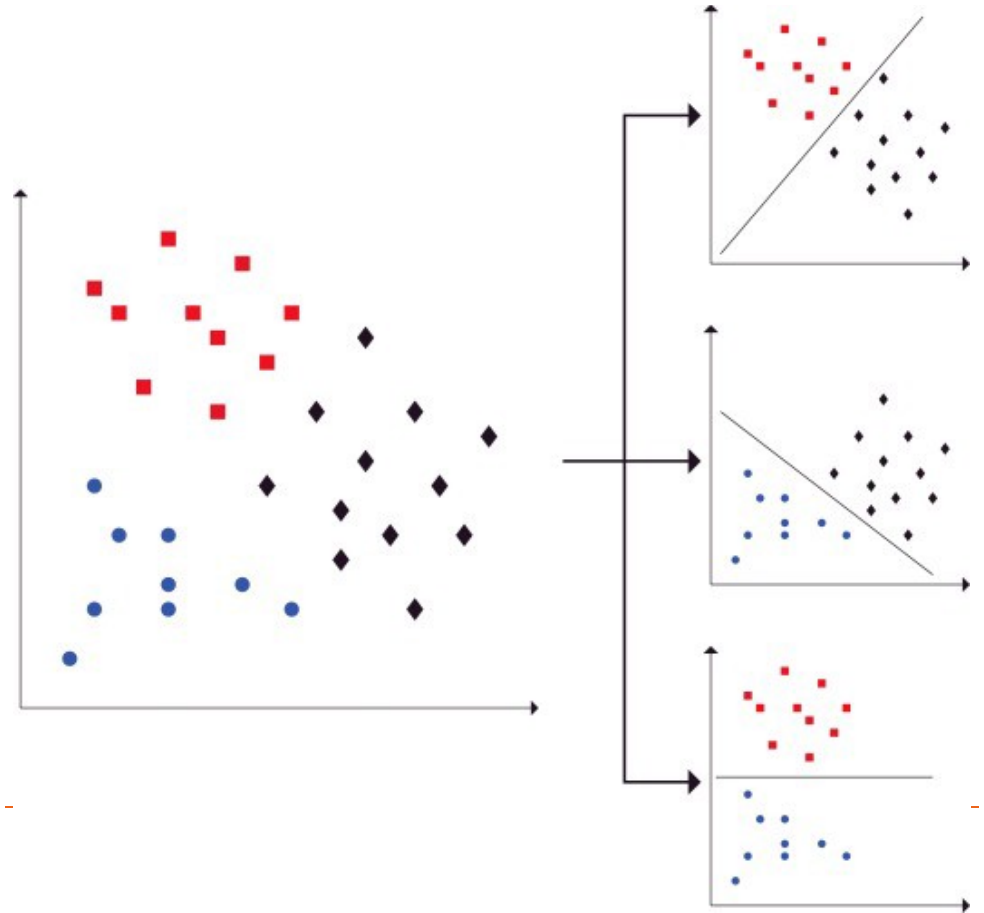
Multi-class Classification

- One-vs-rest:
  - To be classified into three categories: A, B, and C, then you can treat A as positive data, and B and C as negative data.



Multi-class Classification

- One-vs-one :
  - Choose two types of samples each time to do binary logistic regression



Multi-class Classification

- One-vs-rest:
  - +: fast speed
  - -: low accuracy

- One-vs-one :
  - +: high accuracy
  - -: slow speed

Logistic Regression Example

- Step 1: Data preprocessing
- Step 2: Construct various auxiliary functions, such as activation function, optimization function, etc.
- Step 3: Synthesize the above auxiliary functions and combine them into a model

Logistic Regression Example

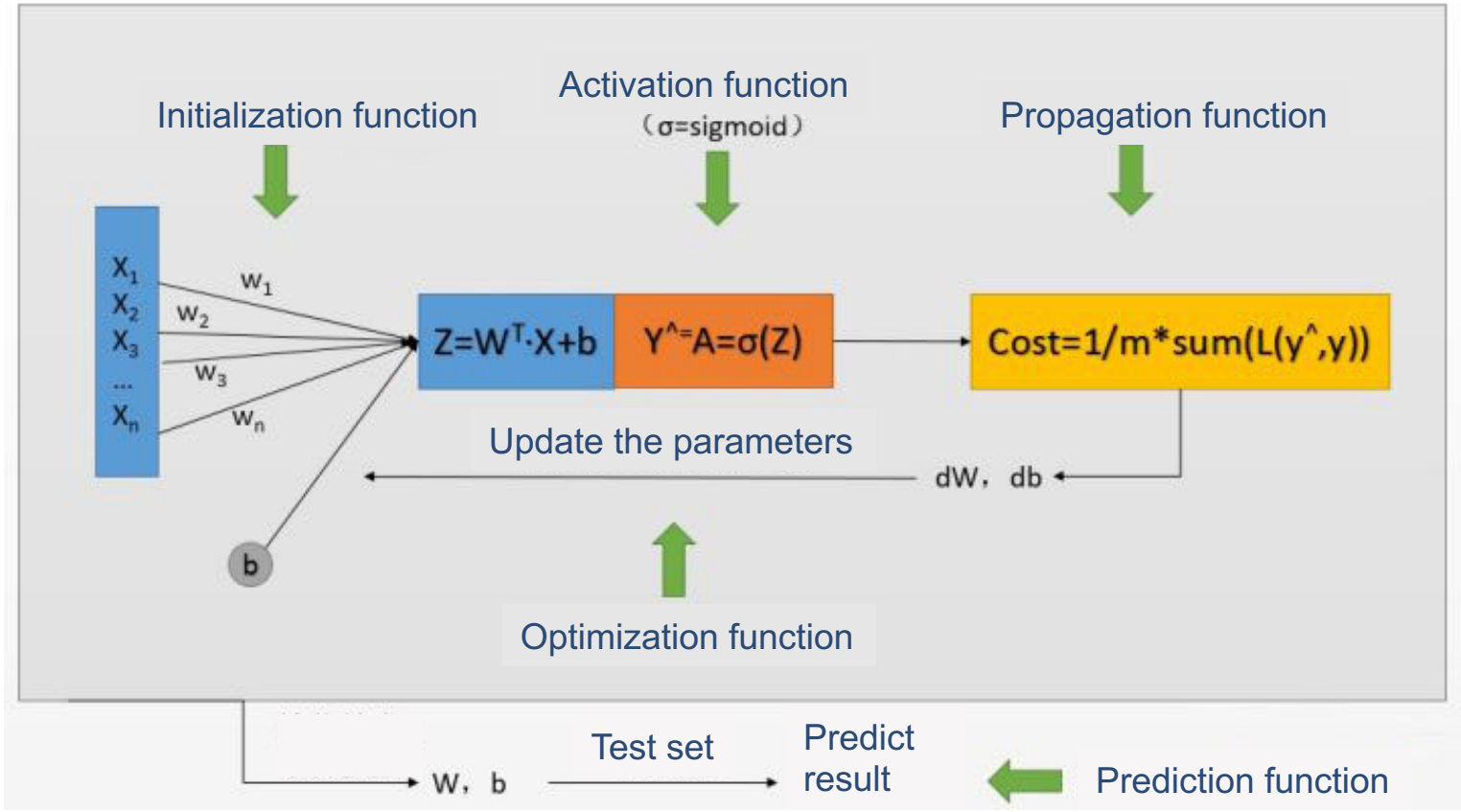
- Step 1: Data preprocessing
  - Figure out the shape and dimensions of the data
  - Convert data (such as pictures) into vectors (image to vector) for easy processing
  - Standardize the data for better training

Logistic Regression Example

- Step 2: Construct various auxiliary functions, such as activation function, optimization function, etc.
  - Activation function (here we use the sigmoid function)
  - Parameter initialization function (used to initialize W and b)
  - Propagation function (here is used to find loss cost and derivation of W and b, namely dW, db)
  - Optimization function (update W and b iteratively to minimize cost)
  - Prediction function (predict based on learned W and b)

Logistic Regression Example

- All functions relationship:



Logistic Regression Example

- Step 3: Synthesize the above auxiliary functions and combine them into a model
  - Input training set, prediction set, hyperparameters, and then give model parameters and accuracy

Logistic Regression Example

https://github.com/kevinsuo/CS7357/blob/master/logistic-regression/logr.py

- Step 1: Data import and preprocessing

```
# Import data, "_orig" represents the original data here, we need to further process it before we can use it:
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
```

- Data is stored in h5 format: https://en.wikipedia.org/wiki/Hierarchical_Data_Format

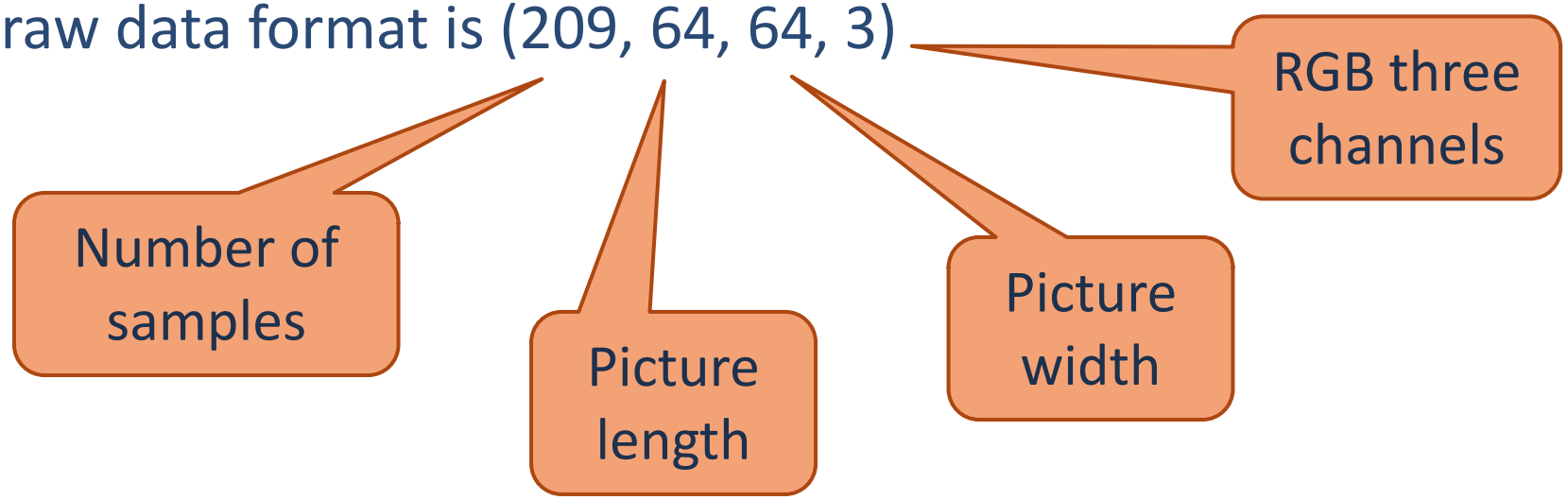
- Use function load_dataset() to import the data

Logistic Regression Example

Step 1: Data import and preprocessing

```
#Get some basic parameters from the data set, such as the number of training samples m, and the image size:
m_train = train_set_x_orig.shape[0]  # Training set size 209
m_test = test_set_x_orig.shape[0]    # Test set size 50
num_px = train_set_x_orig.shape[1]  # Picture width 64，  Size is 64×64
```

- The raw data format is (209, 64, 64, 3)

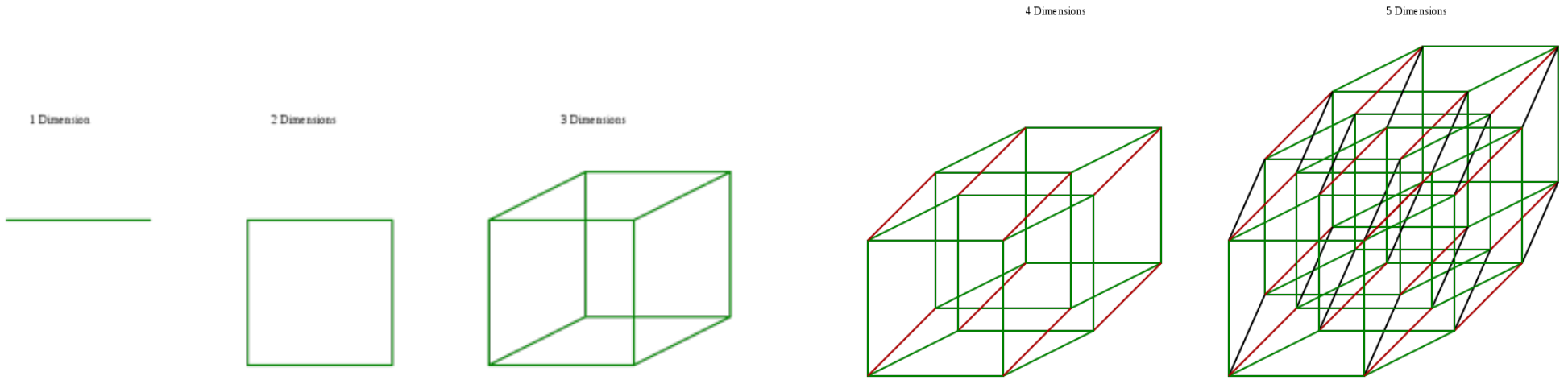


Logistic Regression Example

Step 1: Data import and preprocessing

```
#Vectorize image data (flatten):
train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0],-1).T
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0],-1).T
```

- Four-dimensional vector flattened into two-dimensional vector for better computation



Logistic Regression Example

Step 1: Data import and preprocessing

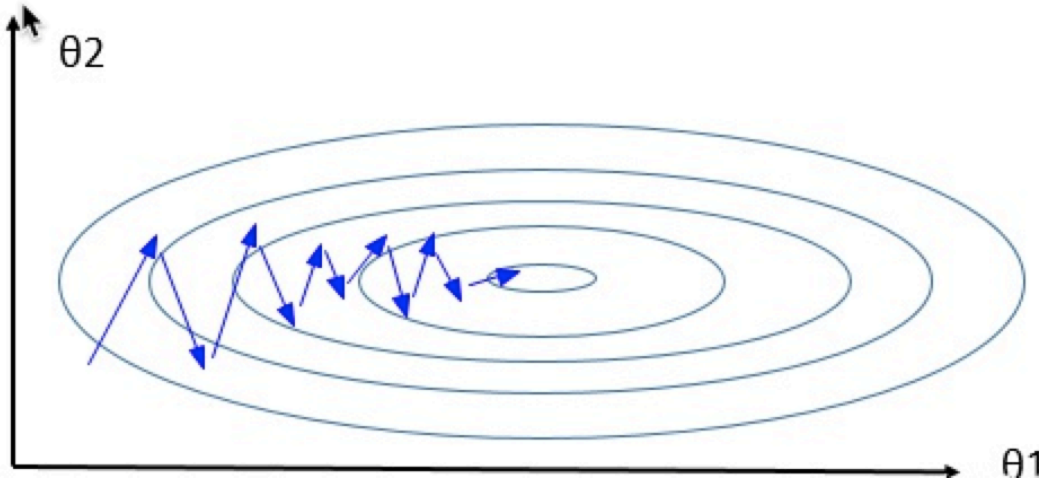
```
#Standardize the data:
train_set_x = train_set_x_flatten/255.
test_set_x = test_set_x_flatten/255.
```

Standardization: Because the range of the three channels of RGB is 255, we can directly divide the image by 255.

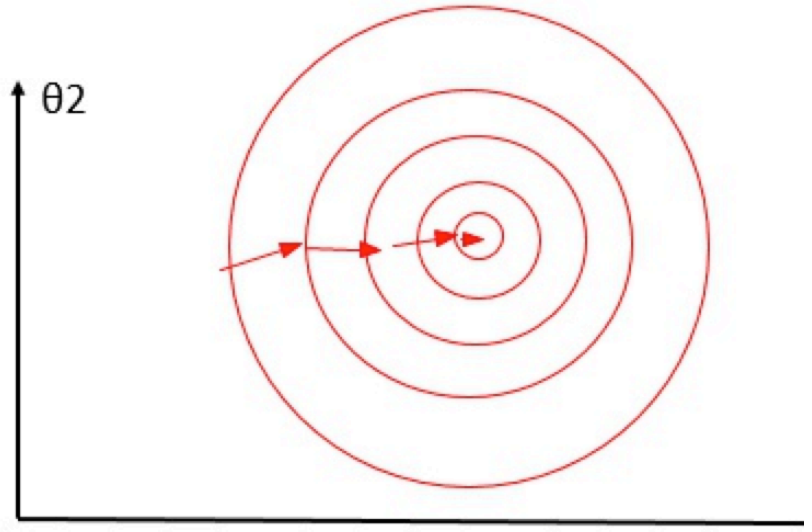
- Why do we need standardization?

Why do we need standardization?

Not standardized



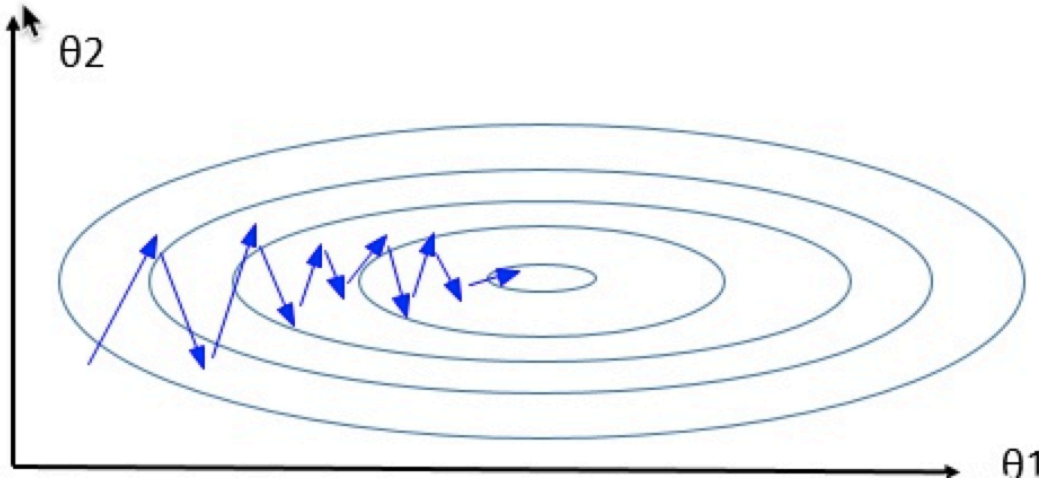
Standardized



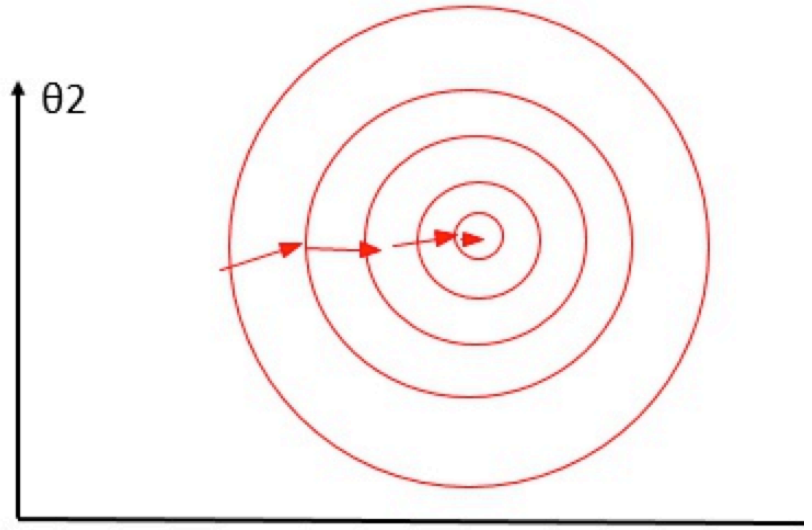
The range of different features of the original data may vary greatly. The large fluctuation range causes the contour map to become very "flat", and it is easy to take detours when the gradient drops, so the gradient drops will be slower and the accuracy is not high

Why do we need standardization?

Not standardized



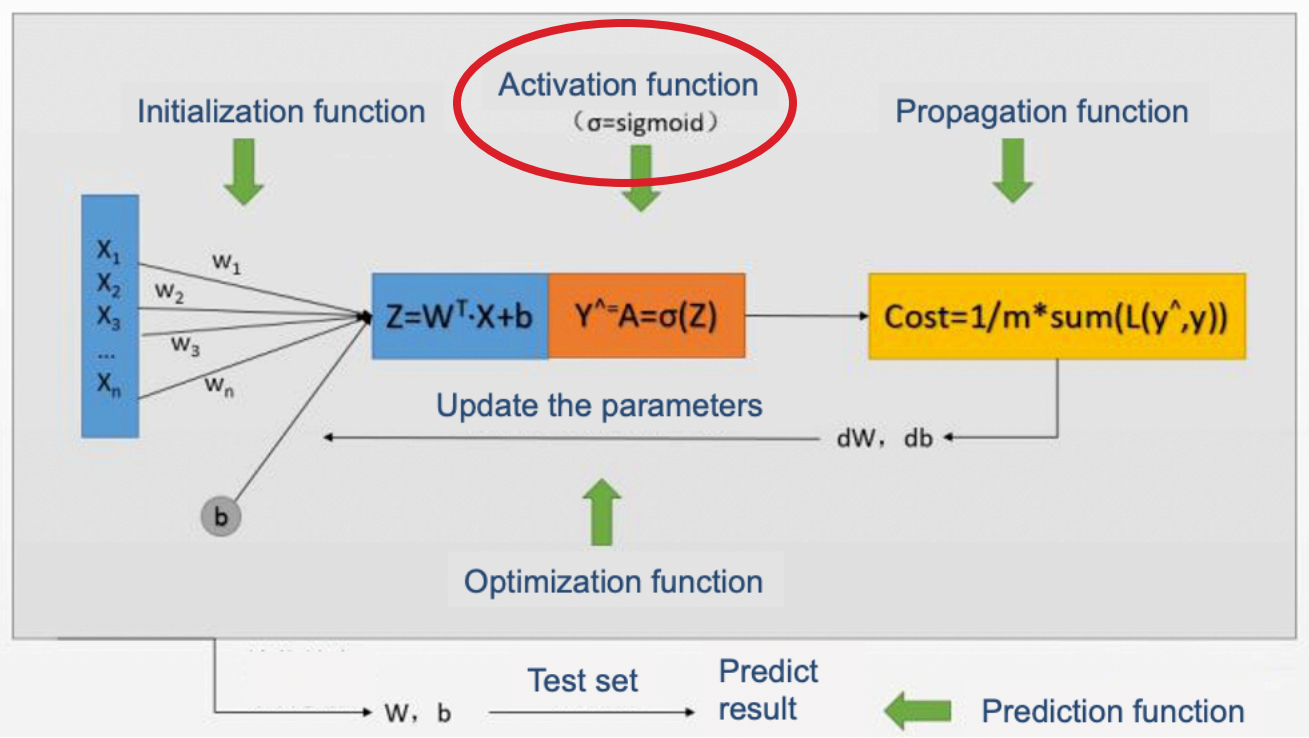
Standardized



After standardization (also known as normalization), the contours become regular, and the gradient is easily reduced.

Logistic Regression Example

- Step 2: Construct various auxiliary functions



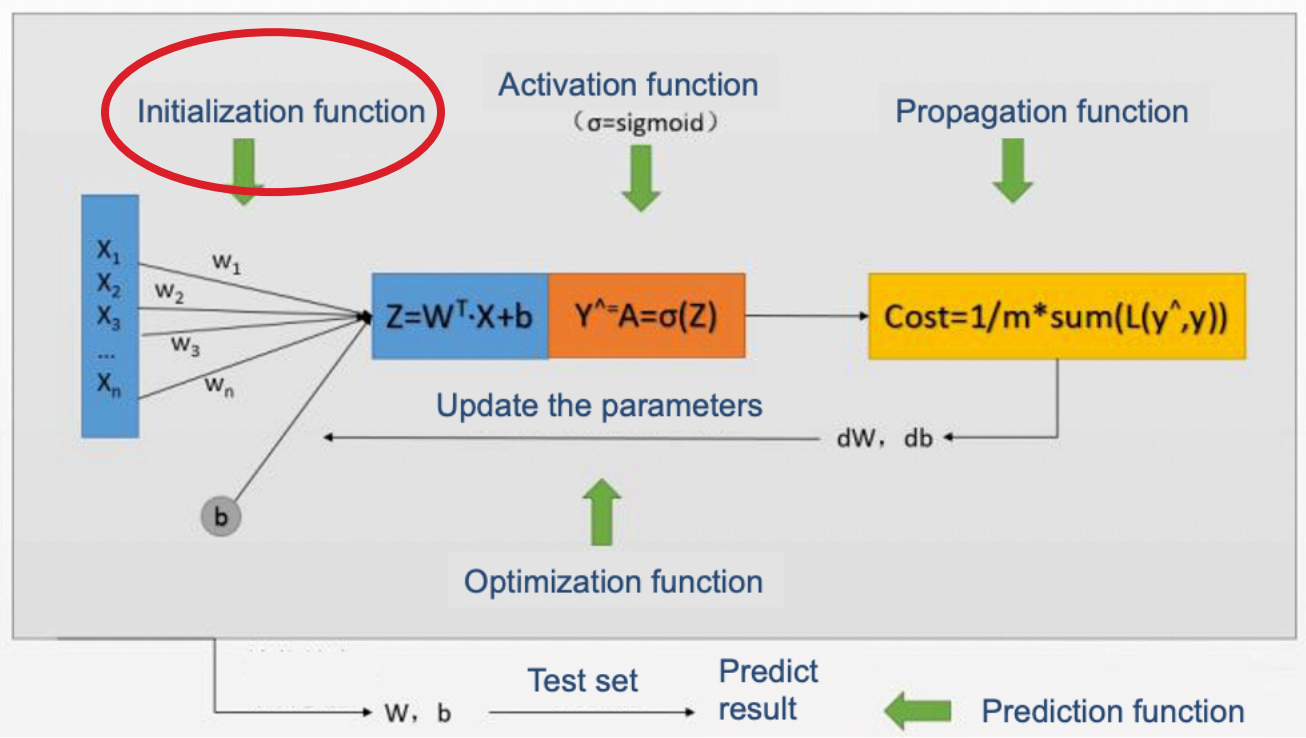
Activation function/sigmoid function

```
def sigmoid(z):
    a = 1/(1+np.exp(-z))
    return a
```

$$S(x) = \frac{1}{1 + e^{-x}}$$

Logistic Regression Example

- Step 2: Construct various auxiliary functions

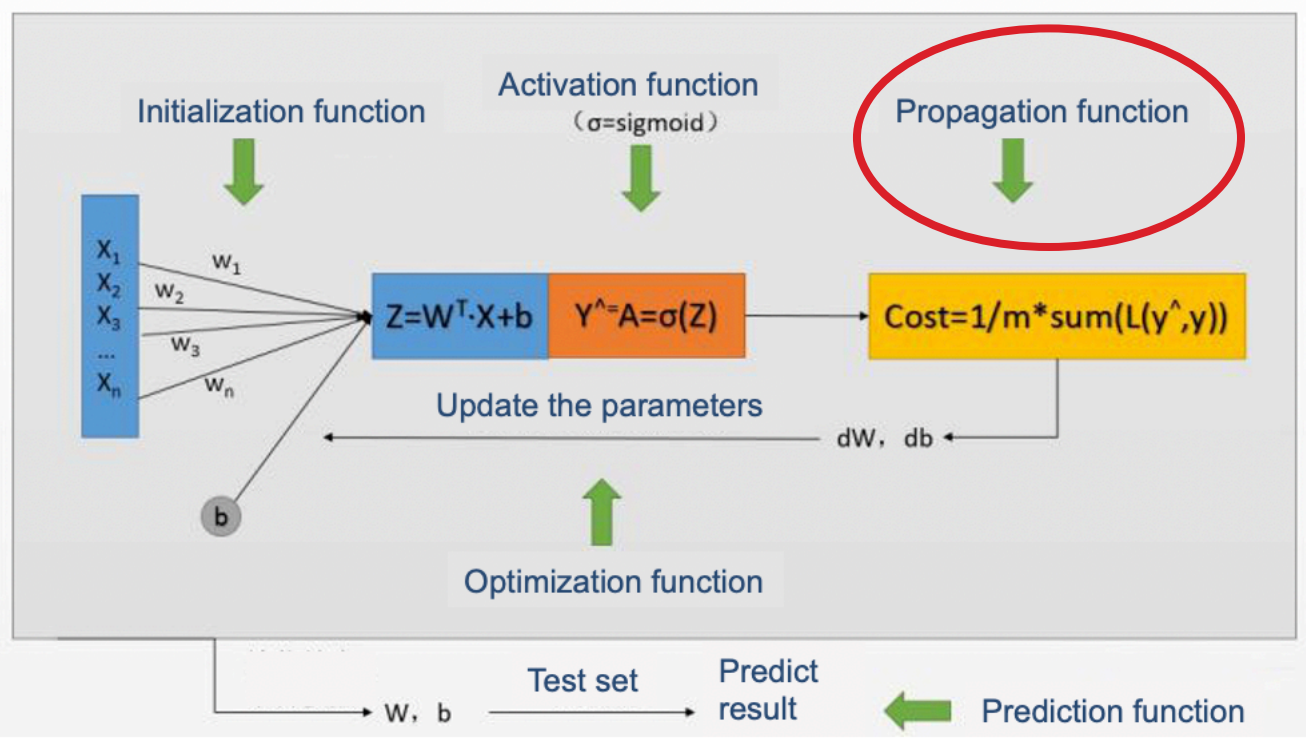



```
def initialize_with_zeros(dim):
    w = np.zeros((dim,1))
    b = 0
    return w,b
```

W is a column vector

Logistic Regression Example

- Step 2: Construct various auxiliary functions

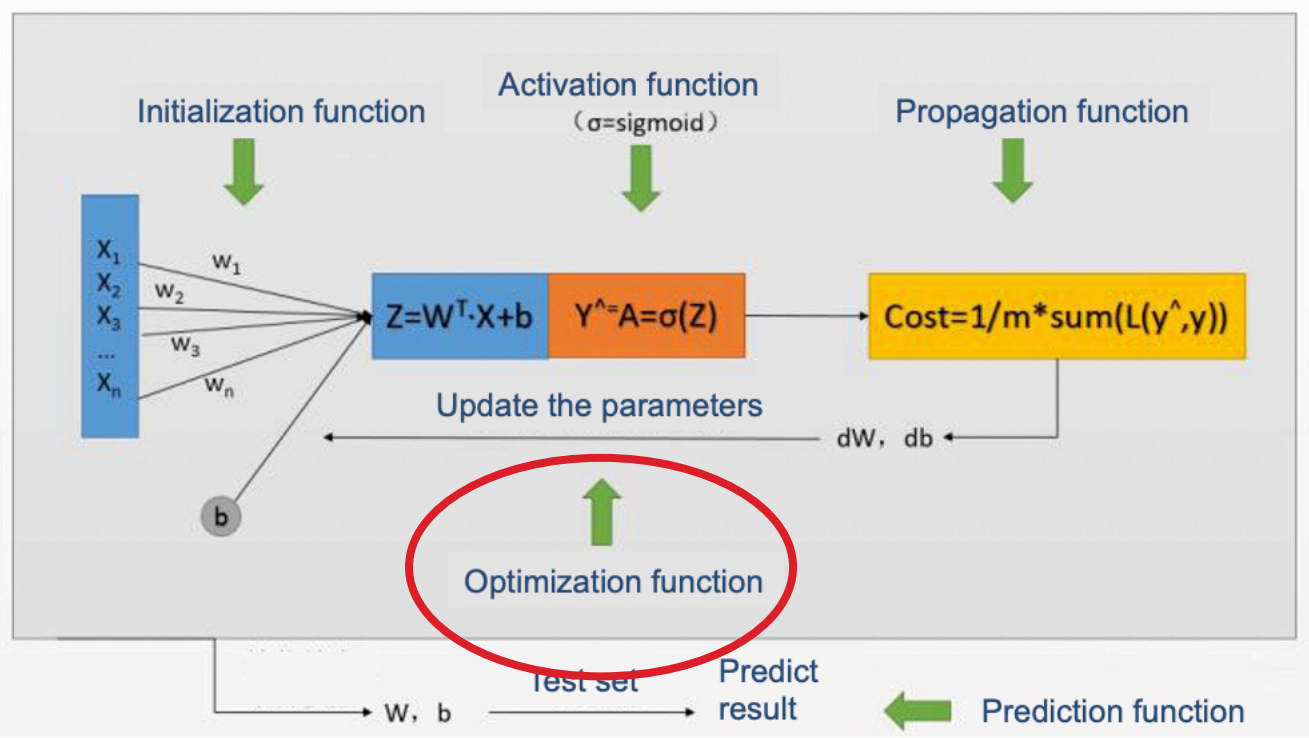



```python
def propagate(w, b, X, Y):
    """
Passage:
    w - weight, shape: (num_px * num_px * 3, 1)
    b - bias term, a scalar
    X - data set, shape: (num_px * num_px * 3, m), m is the number of samples
    Y - real label, shape: (1,m)

    return value:
    cost, dw, db, the latter two are placed in a dictionary grads
    """
    #Get the number of samples m:
    m = X.shape[1]

    # Forward propagation:
    A = sigmoid(np.dot(w.T,X)+b)    # Call the sigmoid function earlier
    cost = -(np.sum(Y*np.log(A)+(1-Y)*np.log(1-A)))/m

    # Backpropagation:
    dZ = A-Y
    dw = (np.dot(X,dZ.T))/m
    db = (np.sum(dZ))/m

    # Return value
    grads = {"dw": dw,
             "db": db}

    return grads, cost
```

Get grads and cost

Logistic Regression Example

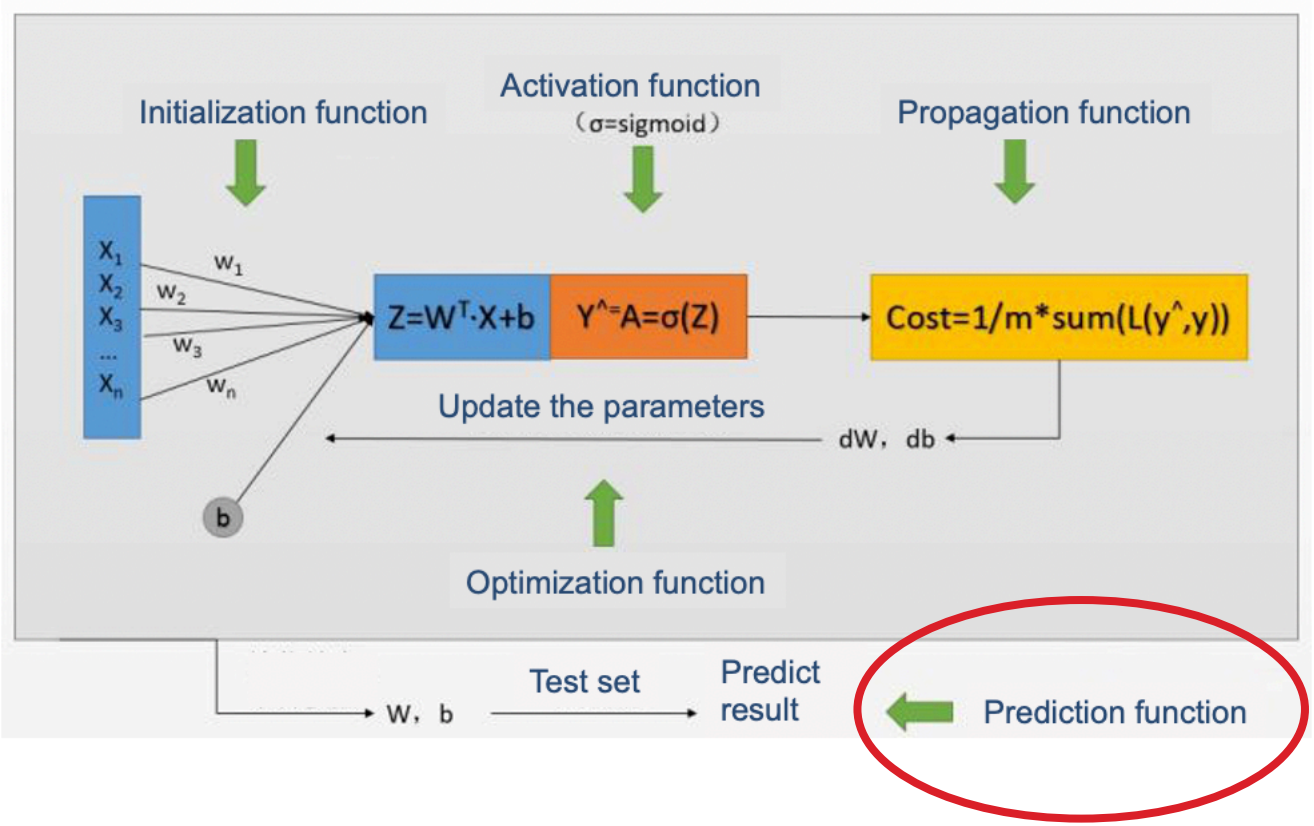
- Step 2: Construct various auxiliary functions




```python
def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost = False):
    #Define a costs array to store the cost after every several iterations,
    #so that you can draw a graph to see the change trend of the cost:
    costs = []
    #To iterate:
    for i in range(num_iterations):
        # Use propagate to calculate the cost and gradient after each iteration:
        grads, cost = propagate(w,b,X,Y)
        dw = grads["dw"]
        db = grads["db"]

        # Use the gradient obtained above to update the parameters:
        w = w - learning_rate*dw
        b = b - learning_rate*db

        # Every 100 iterations, save a cost to see:
        if i % 100 == 0:
            costs.append(cost)

        # We can print out the cost every 100 times to see the progress of the model
        if print_cost and i % 100 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))
    #After iteration, put the final parameters into the dictionary and return:
    params = {"w": w,
              "b": b}
    grads = {"dw": dw,
             "db": db}
    return params, grads, costs
```

Logistic Regression Example

- Step 2: Construct various auxiliary functions




```python
def predict(w,b,X):
    m = X.shape[1]
    Y_prediction = np.zeros((1,m))

    A = sigmoid(np.dot(w.T,X)+b)
    for  i in range(m):
        if A[0,i]>0.5:
            Y_prediction[0,i] = 1
        else:
            Y_prediction[0,i] = 0

    return Y_prediction
```

Having learned the parameters W and b, we can get the predicted value by passing our data through a model equipped with these parameters

Logistic Regression Example

- Step 3: Synthesize the above auxiliary functions and combine them into a model

```
def logistic_model(X_train,Y_train,X_test,Y_test,learning_rate=0.1,num_iterations=2000,print_cost=False):
    #Obtain feature dimensions, initialization parameters:
    dim = X_train.shape[0]
    W,b = initialize_with_zeros(dim)

    #Gradient descent, iteratively find the model parameters:
    params,grads,costs = optimize(W,b,X_train,Y_train,num_iterations,learning_rate,print_cost)
    W = params['w']
    b = params['b']

    #Use the learned parameters to predict:
    prediction_train = predict(W,b,X_train)
    prediction_test = predict(W,b,X_test)

    #Calculate the accuracy rate, respectively on the training set and test set:
    accuracy_train = 1 - np.mean(np.abs(prediction_train - Y_train))
    accuracy_test = 1 - np.mean(np.abs(prediction_test - Y_test))
    print("Accuracy on train set:",accuracy_train )
    print("Accuracy on test set:",accuracy_test )

   #In order to facilitate analysis and inspection, we store all the parameters
   #and hyperparameters obtained in a dictionary and return them:
    d = {"costs": costs,
         "Y_prediction_test": prediction_test ,
         "Y_prediction_train" : prediction_train ,
         "w" : W,
         "b" : b,
         "learning_rate" : learning_rate,
         "num_iterations": num_iterations,
         "train_acy":accuracy_train,
         "test_acy":accuracy_test
        }
    return d
```

Logistic Regression Example

- Set parameter and run our model
  - num_iterations = 2000, learning_rate = 0.005
  - d = logistic_model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 2000, learning_rate = 0.005, print_cost = True)

```
d = logistic_model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 2000, learning_rate = 0.005, p
```

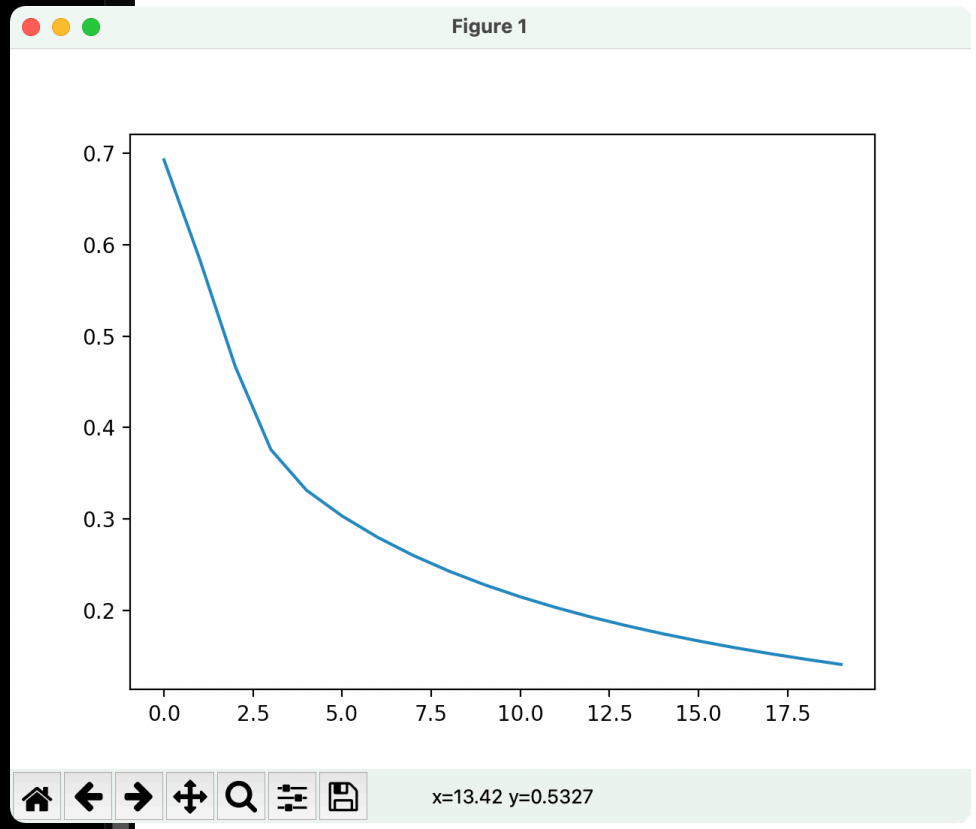
Logistic Regression Example

https://github.com/kevinsuo/CS7357/blob/master/logistic-regression/logr.py

```
fish /Users/ksuo/Google Drive/github/CS7357/logistic-regression
ksuo@ltksup50143mac ~/G/g/C/logistic-regression> python3 logr.py
(12288, 209)
(1, 209)
(12288, 50)
(1, 50)
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.584508
Cost after iteration 200: 0.466949
Cost after iteration 300: 0.376007
Cost after iteration 400: 0.331463
Cost after iteration 500: 0.303273
Cost after iteration 600: 0.279880
Cost after iteration 700: 0.260042
Cost after iteration 800: 0.242941
Cost after iteration 900: 0.228004
Cost after iteration 1000: 0.214820
Cost after iteration 1100: 0.203078
Cost after iteration 1200: 0.192544
Cost after iteration 1300: 0.183033
Cost after iteration 1400: 0.174399
Cost after iteration 1500: 0.166521
Cost after iteration 1600: 0.159305
Cost after iteration 1700: 0.152667
Cost after iteration 1800: 0.146542
Cost after iteration 1900: 0.140872
Accuracy on train set: 0.9904306220095693
Accuracy on test set: 0.7
ksuo@ltksup50143mac ~/G/g/C/logistic-regression>
```

As the training progresses, the cost continues to decrease

The accuracy rate on the training set is over 99%, and the accuracy rate on the test set is 70%



Softmax

Softmax Regression: Generalization of Logistic Regression

logistic regression

$$h_\theta(x) = g(\theta^T x)$$

$$= \frac{1}{e^{\theta_1^T x} + e^{\theta_2^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x} \\ e^{\theta_2^T x} \end{bmatrix}$$

loss function

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

softmax regression

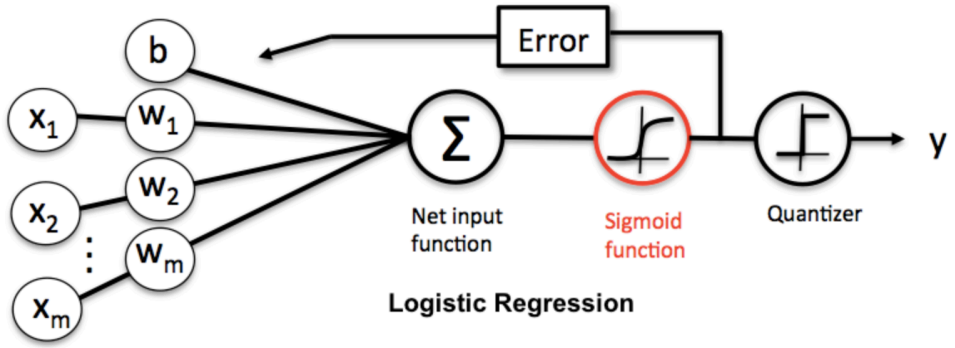
$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)}; \theta) \\ p(y^{(i)} = 2|x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)}; \theta) \end{bmatrix}$$

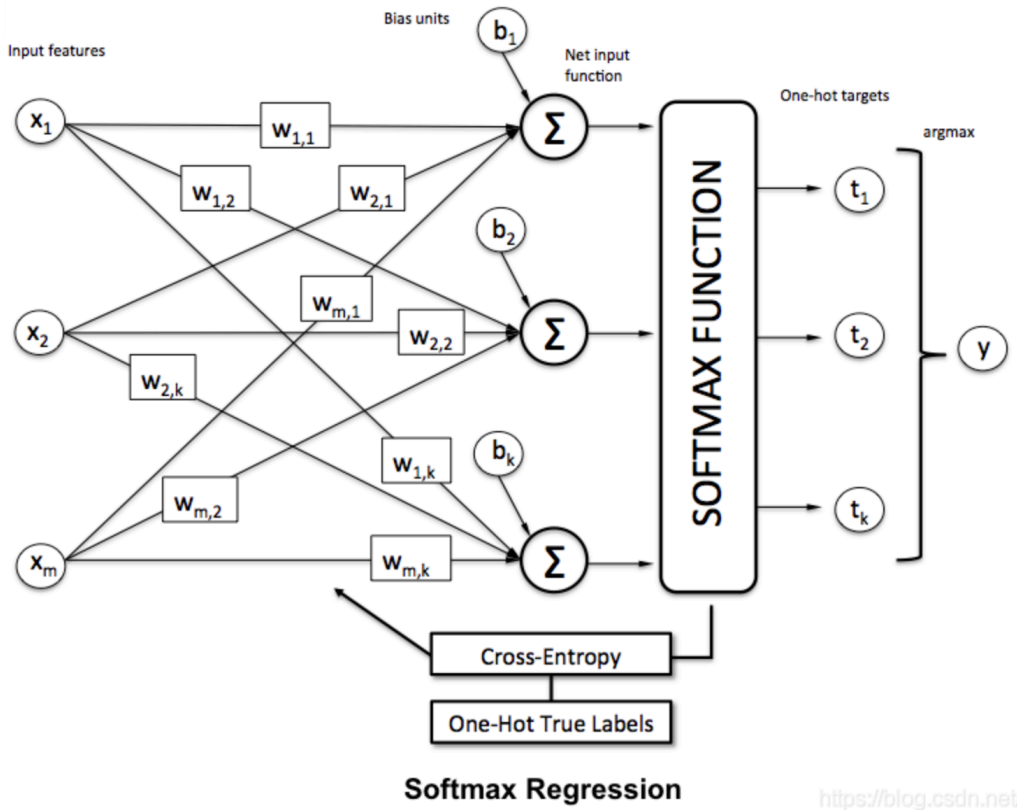
$$= \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} \sum_{j=1}^{k} 1 \left\{ y^{(i)} = j \right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}} \right]$$

Softmax regression

- Softmax regression is a general form of logistic regression. When the number of categories k = 2, softmax regression degenerates into logistic regression

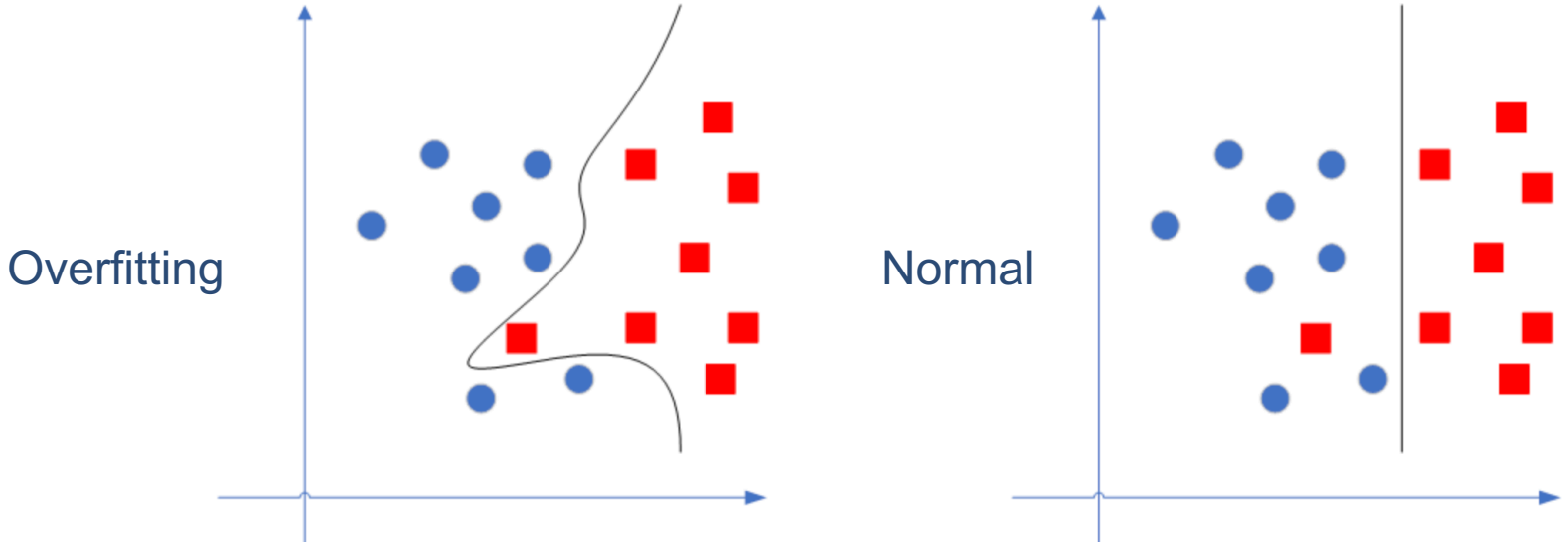




Regularization of Logistic Regression

- Regularization is to reduce the accuracy of the logistic regression model

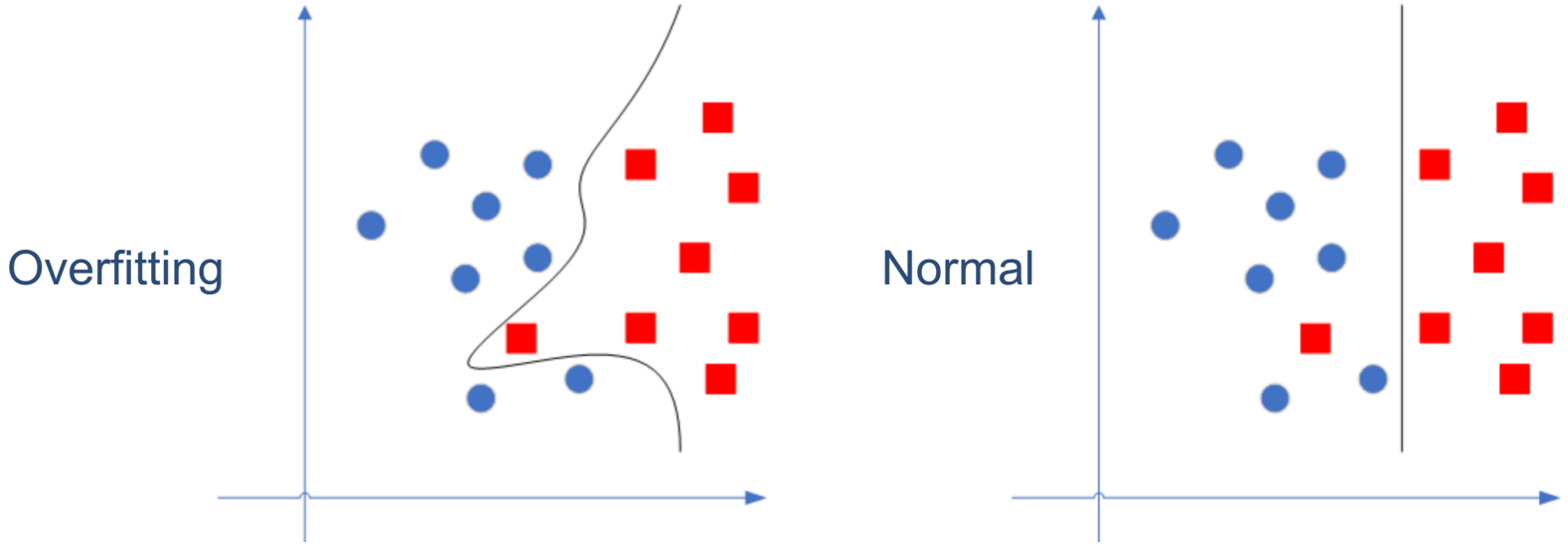
- Isn't the accuracy of the model the higher the better?



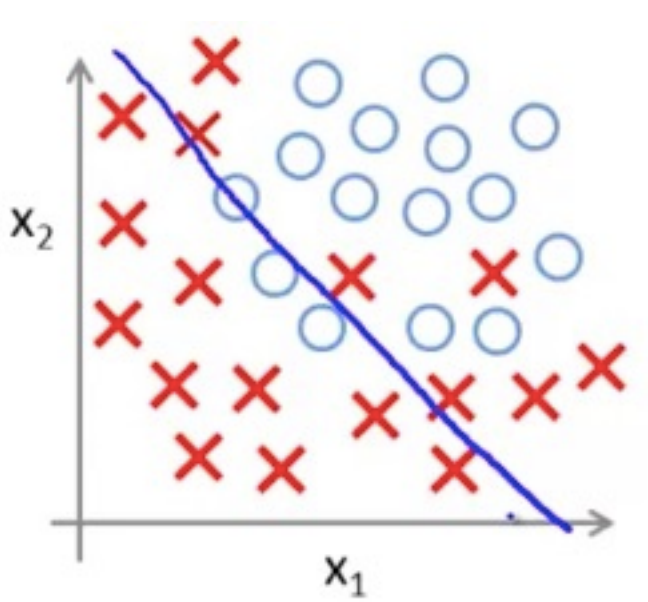
Regularization of Logistic Regression

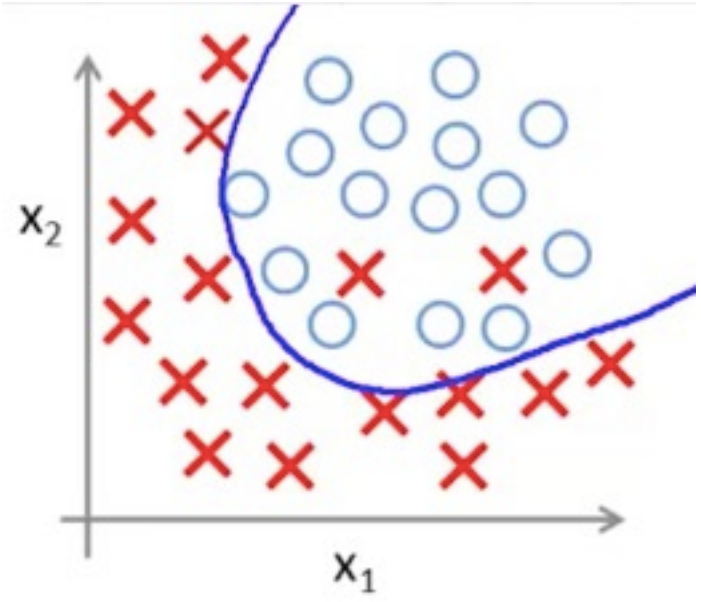
- If the accuracy is too high, it can match the data of the training set very well, but once new data is available, it will perform very poorly

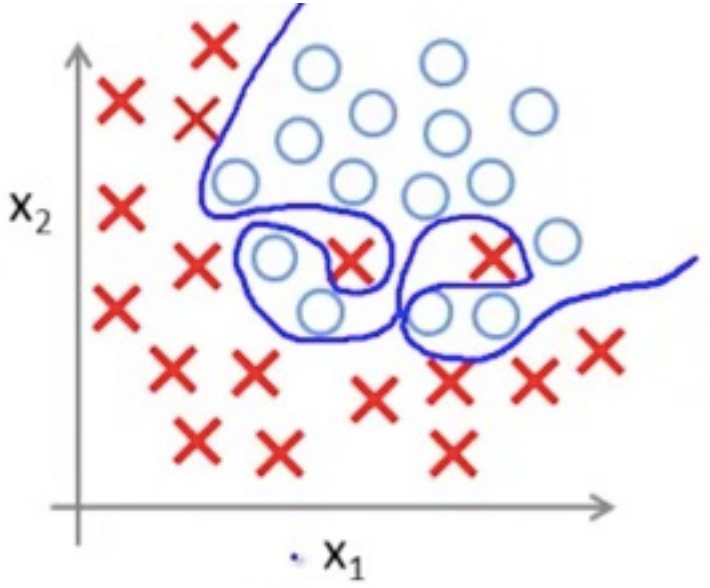
- The accuracy can be worse sometimes, but the generalization performance, which is the ability to recognize new data, is better



Overfitting of Logistic Regression







Decision boundary

$$h(\theta) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$$h(\theta) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta x_2^2 + \theta_5 x_1 x_2)$$

$$h(\theta) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

Overfitting of Logistic Regression

$$h(\theta) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$$

- A high-order polynomial may fit the training set well and can fit almost all training data.
- But this function is too large and there are too many variables. If there is not enough data to constrain this model with too many variables
- Overfitting means that it performs very well on the training set (known data), but performs very badly on unknown data

Perceptron

Perceptron

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN [1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

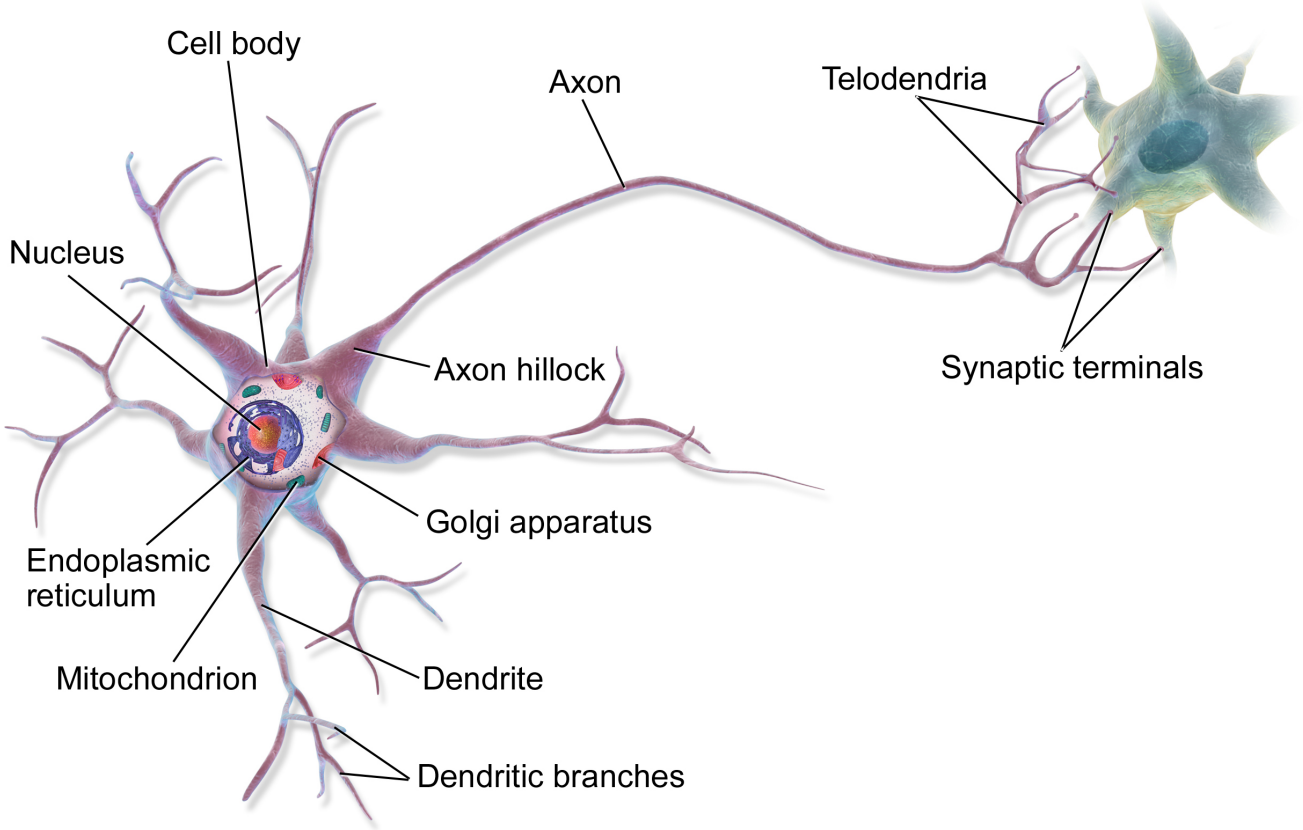
HAVING told you about the giant digital computer known as I.B.M. 704 and how it has been taught to play a fairly creditable game of chess, we'd like to tell you about an even more remarkable machine, the perceptron, which, as its name implies, is capable of what amounts to original thought. The first perceptron has yet to be built,

The New Yorker, December 6, 1958 P. 44

The IBM 704 computer

Perceptron

- Perceptron is a simple abstraction of biological nerve cells. The structure of nerve cells can be roughly divided into: dendrites, synapses, cell bodies and axons.

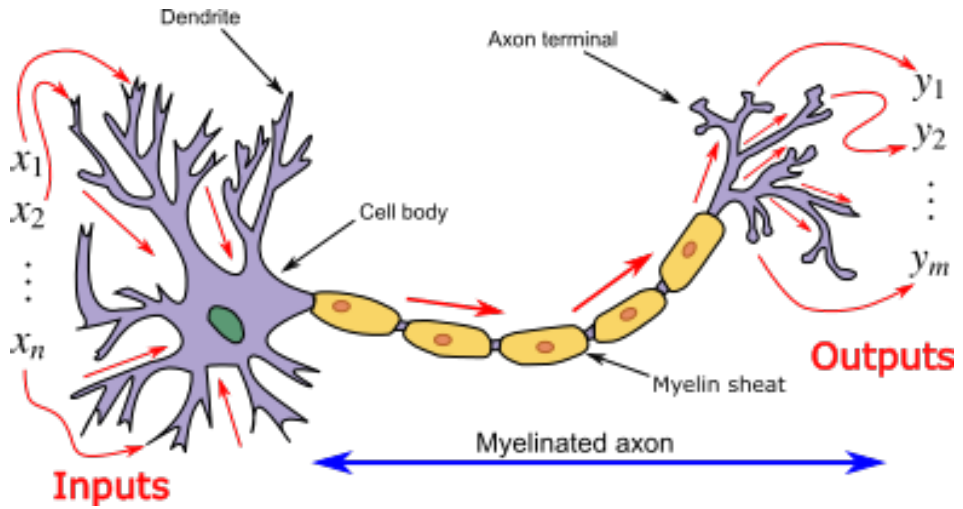


Perceptron

- A single nerve cell can be regarded as a machine with only two states-‘yes’ when activated, and ‘no’ when inactive.

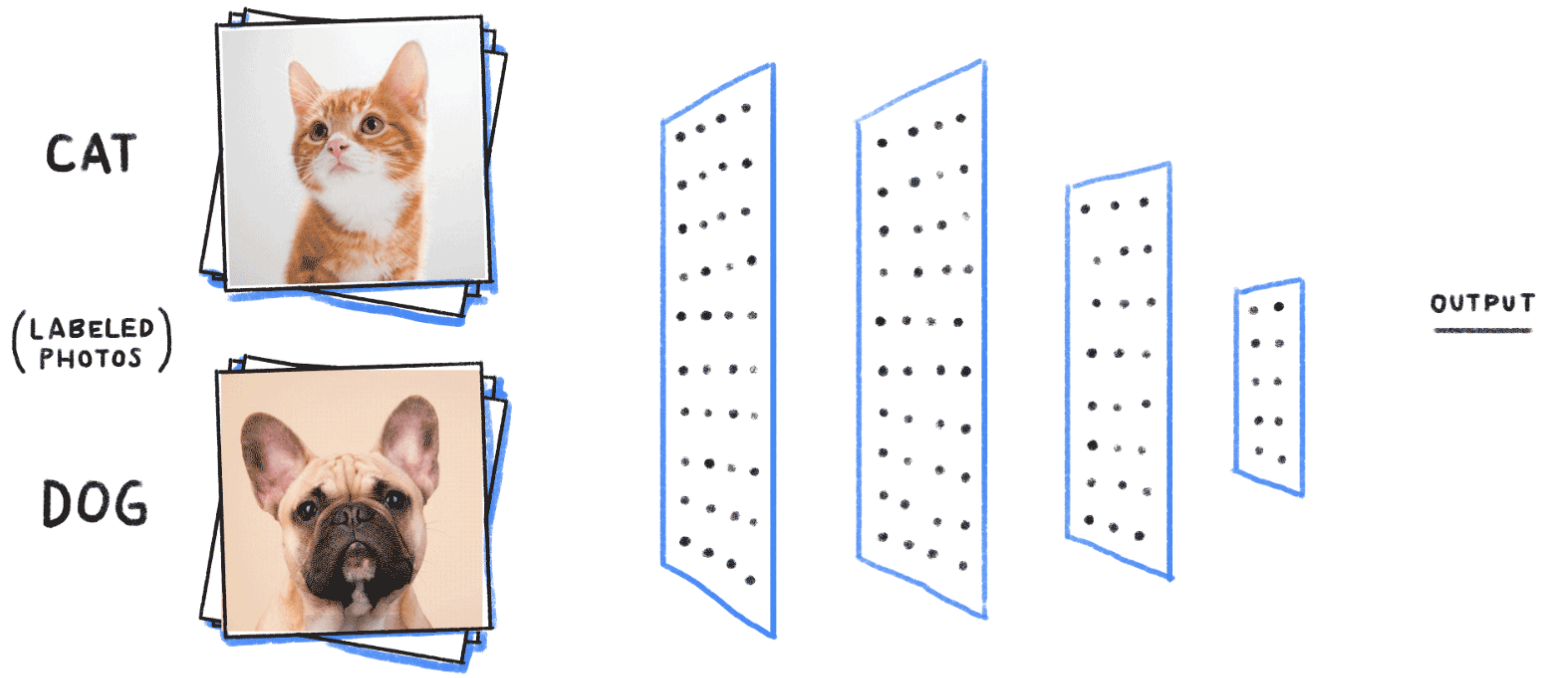
- The state of nerve cells depends on the amount of input signals received from other nerve cells and the strength of synapses (inhibition or enhancement).

- When the sum of the semaphores exceeds a certain threshold, the cell body will activate



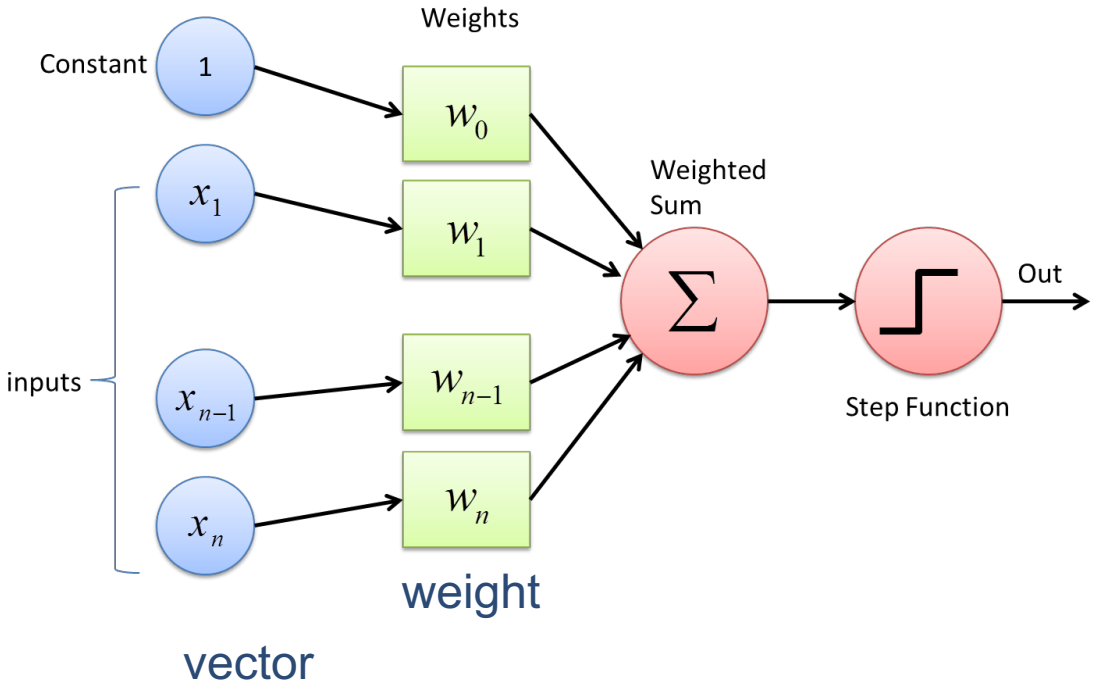
Perceptron

- The feedforward operation of a multilayer neural network is as follows:



Perceptron

- Perceptron is a two-class linear model, the input is feature vector, the output is a binary value



$$f(x) = \sum_{i=1}^{N} w_i \mathbf{x}_i + b \quad = sign(\mathbf{w}^T \mathbf{x} + \mathbf{b})$$

If $f(x) > threshold$, then y(tag) is set to 1

If $f(x) < threshold$, then y(tag) is set to 0

Perceptron

- perceptron has four parts
  - Input values or One input layer
  - Weights and Bias
  - Net sum
  - Activation Function

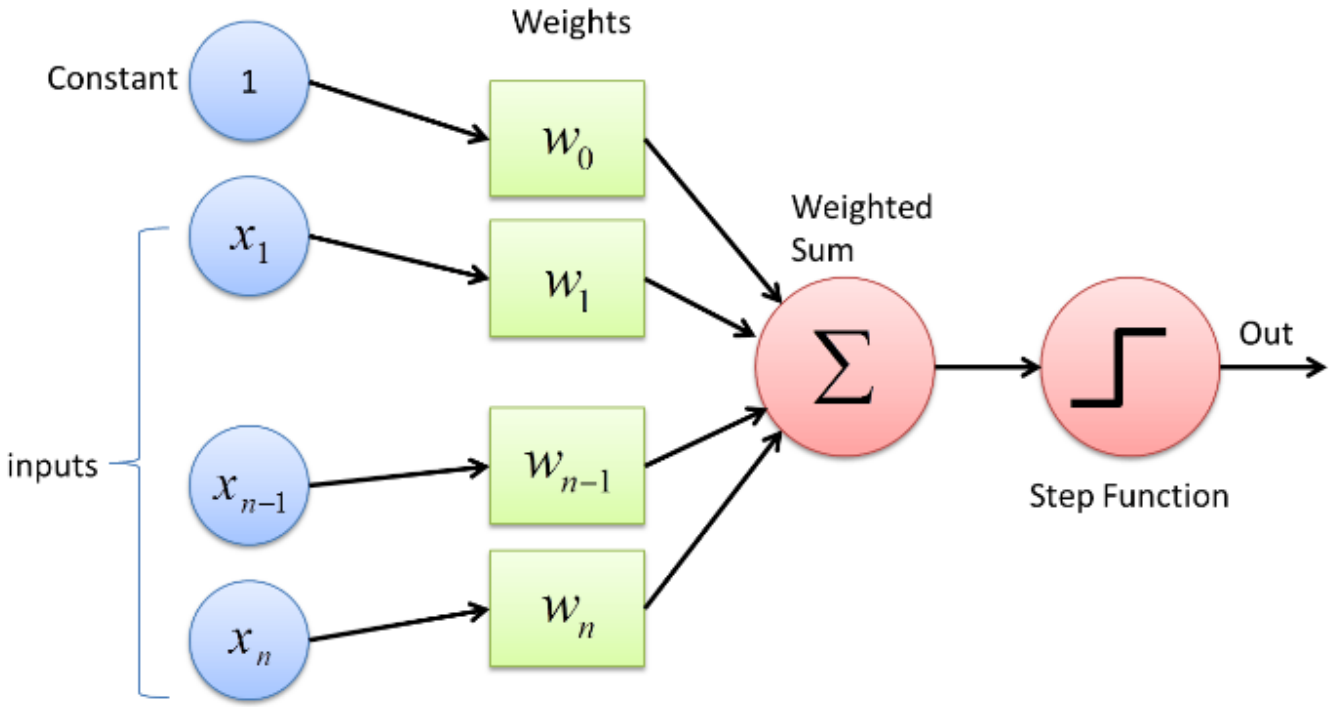


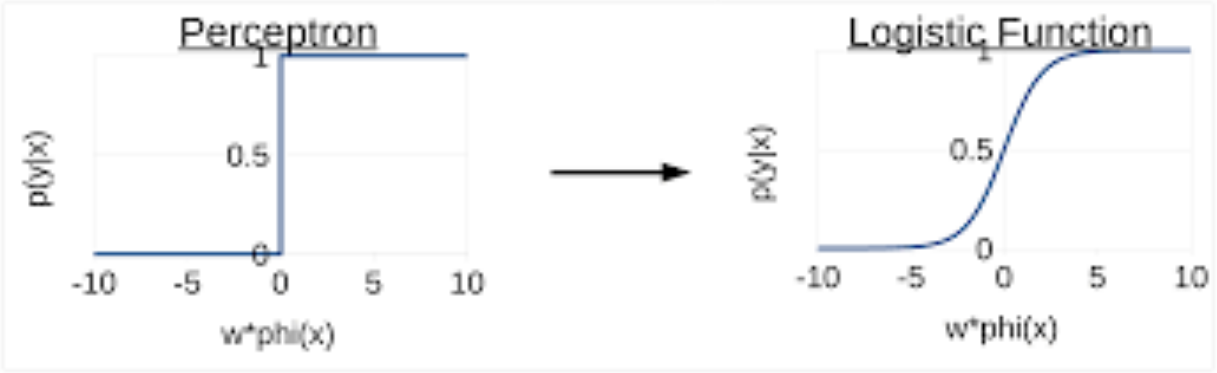
Fig : Perceptron

Perceptron vs. Logistic regression

- The perceptron model is simple and intuitive, but the problem is that this model is not smooth enough (Discontinuous)

- Logistic regression makes the mapping from $\omega^T x+b$ to y smoother. Input is $-\infty \rightarrow +\infty$, output is $0 \rightarrow 1$

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$



Perceptron learning algorithm

Input :

- Training set $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$;
- Leanring rate $\eta > 0$;
- Maximum number of iterations T ;

Initialisation :

- Initialize weights $\boldsymbol{w}^{(0)} = (\bar{\boldsymbol{w}}^{(0)}, w_0^{(0)})$;
- $t \leftarrow 0$; // Generally $\boldsymbol{w}^{(0)} = \mathbf{0}$

while $t \leqslant T$ **do**

  Choose an example at random $(\mathbf{x}, y) \in S$;

  if $y \times \left(\left\langle \bar{\boldsymbol{w}}^{(t)}, \mathbf{x} \right\rangle + w_0^{(t)} \right) \leqslant 0$ **then**

    $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \times y$;

    $\bar{\boldsymbol{w}}^{(t+1)} \leftarrow \bar{\boldsymbol{w}}^{(t)} + \eta \times y \times \mathbf{x}$;

  else

    $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)}$;

  $t \leftarrow t + 1$;

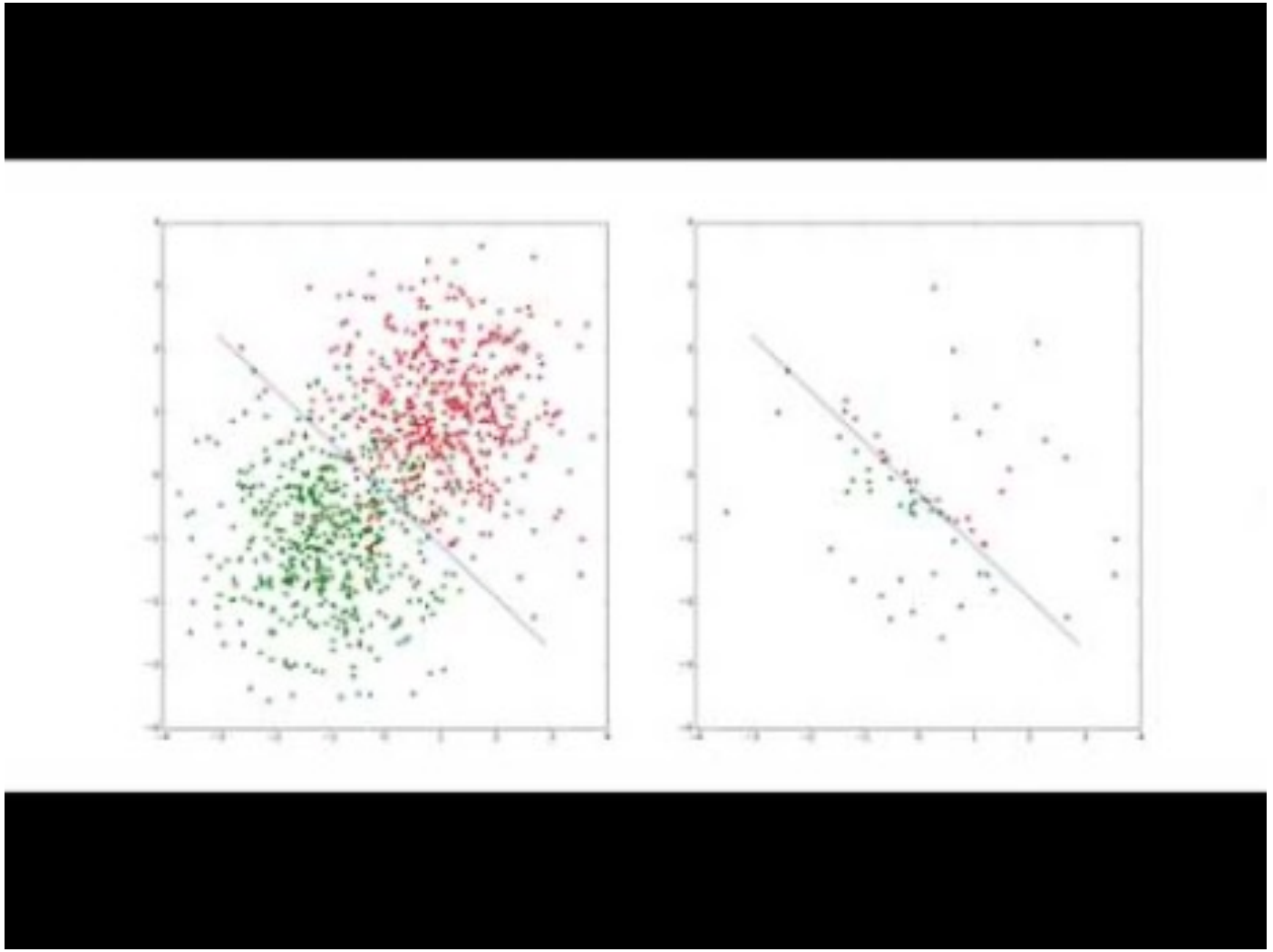
Output : Parameters of the linear model $\boldsymbol{w}^{(t)}$

Regression in perceptron

Logistic regression

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}^{(n)} \left(y^{(n)} - \hat{y}_{\boldsymbol{w}_t}^{(n)} \right)$$

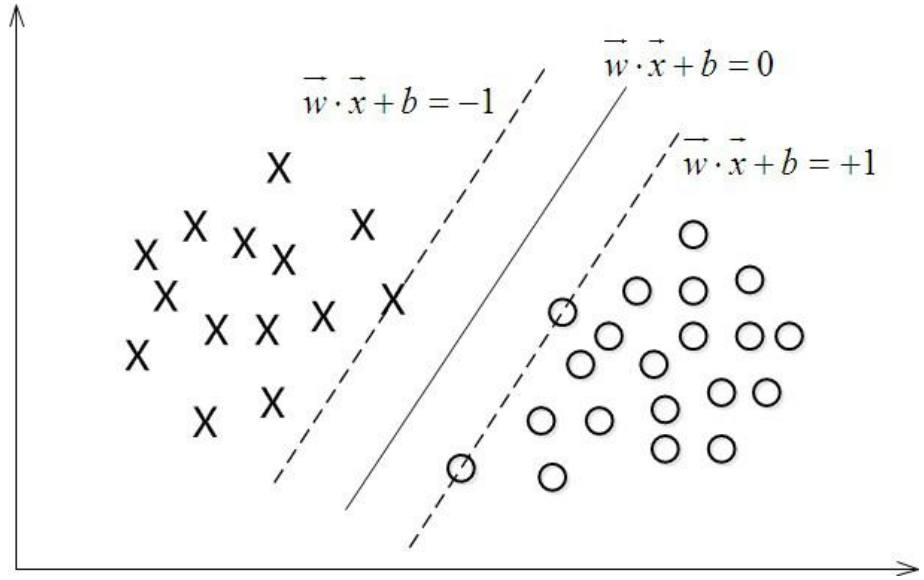
Perceptron learning algorithm



Support Vector Machine

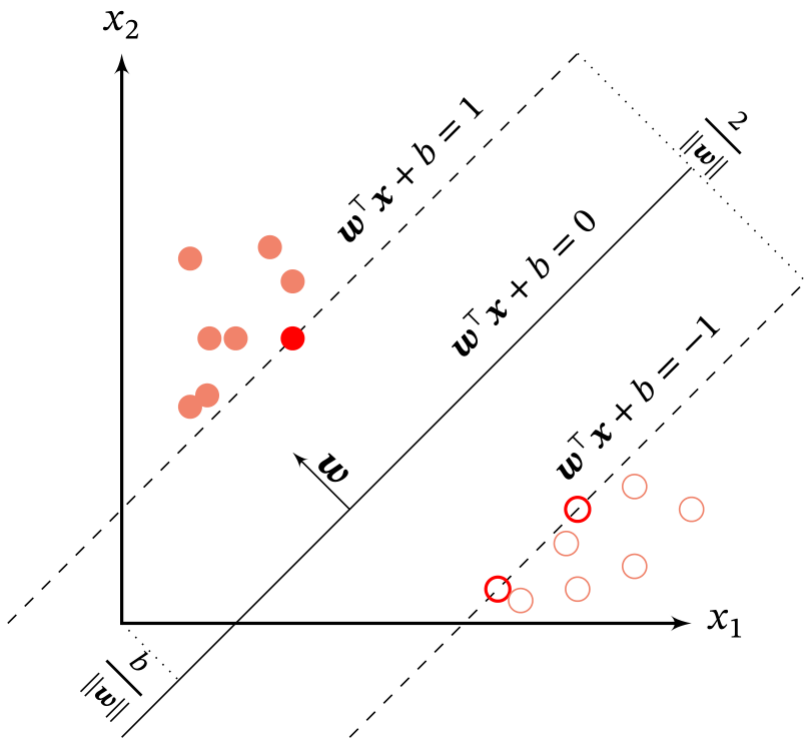
- Support vector machines (SVM) is a two-class model

- It defines the linear classifier with the largest interval in the feature space



For linearly separable data sets, there are infinitely many such hyperplanes (perceptrons), but the separated hyperplane with the largest geometric interval is the only one.

Support Vector Machine



The goal of the support vector machine is to find a hyperplane (w^*, b^*) that maximizes γ

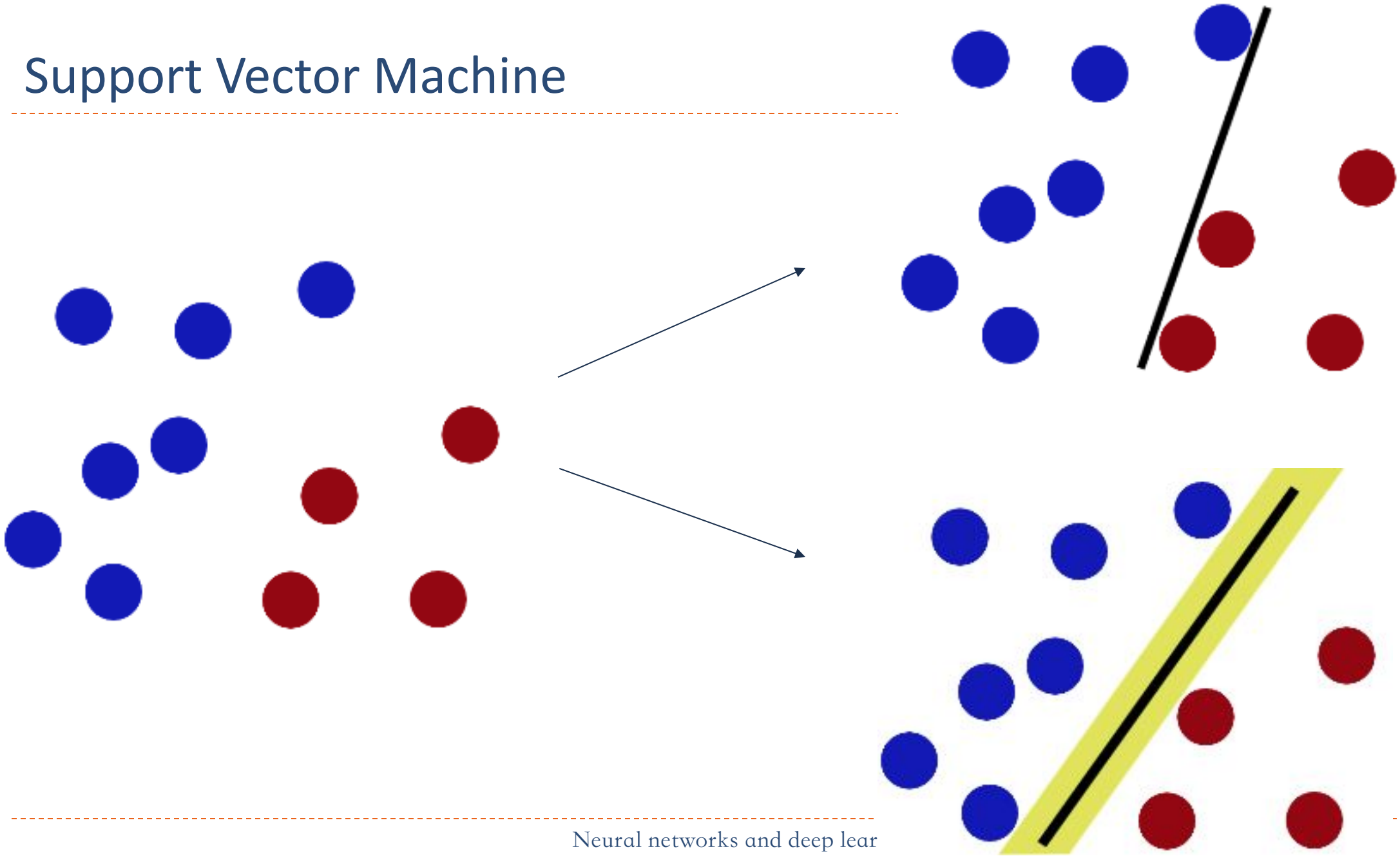
$$\max_{\boldsymbol{w},b} \quad \gamma$$

$$\text{s.t.} \quad \frac{y^{(n)}(\boldsymbol{w}^\top \boldsymbol{x}^{(n)} + b)}{\|\boldsymbol{w}\|} \geq \gamma, \forall n \in \{1, \cdots, N\}$$

$$\max_{\boldsymbol{w},b} \quad \frac{1}{\|\boldsymbol{w}\|^2}$$

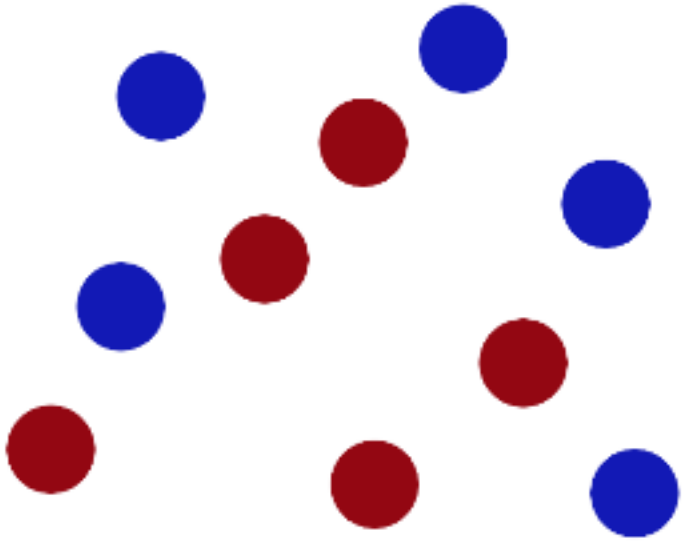
$$\text{s.t.} \quad y^{(n)}(\boldsymbol{w}^\top \boldsymbol{x}^{(n)} + b) \geq 1, \forall n \in \{1, \cdots, N\}$$

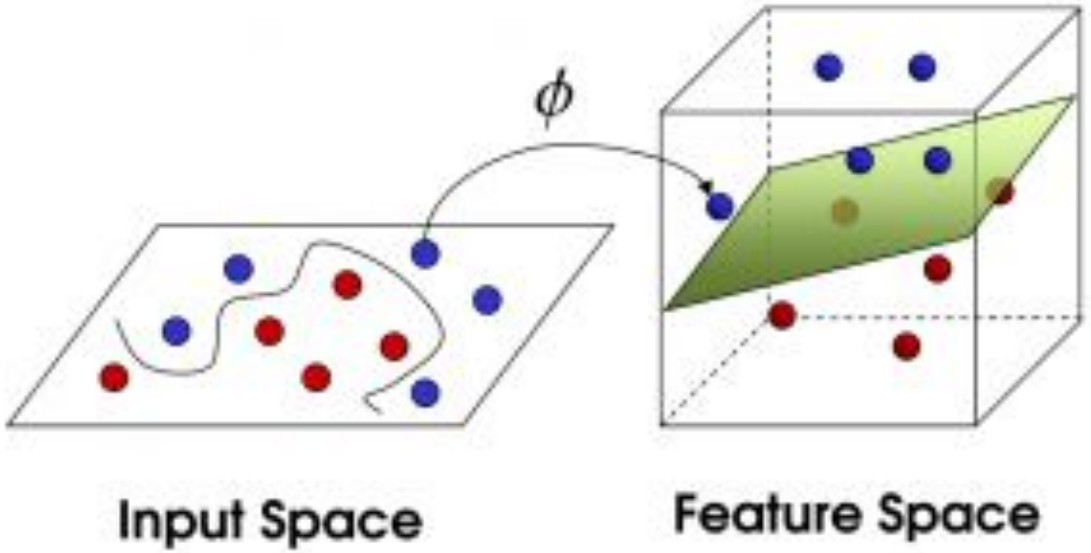
Support Vector Machine



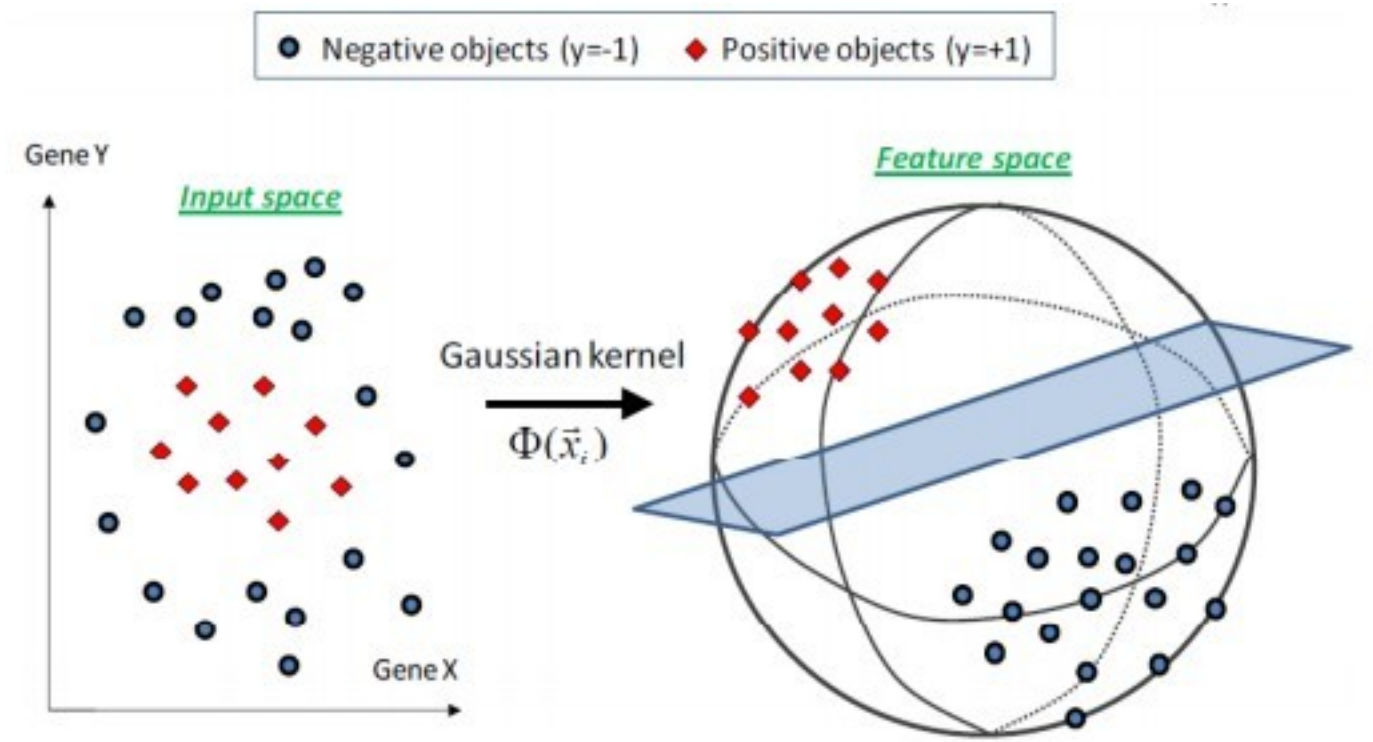
Support Vector Machine

The low-dimensional linearly inseparable data is mapped to the high-dimensional space through the Gaussian kernel function:



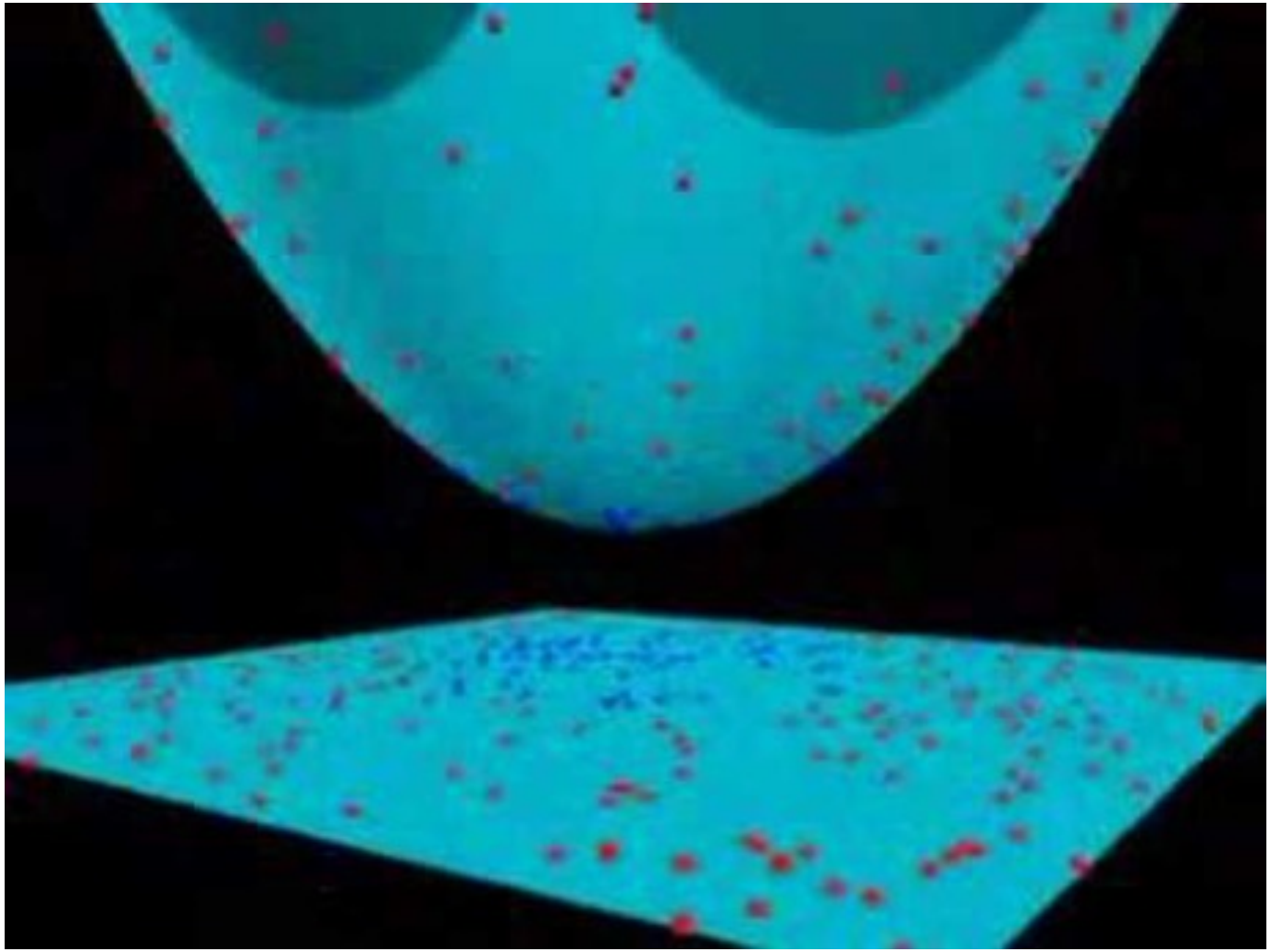


Support Vector Machine



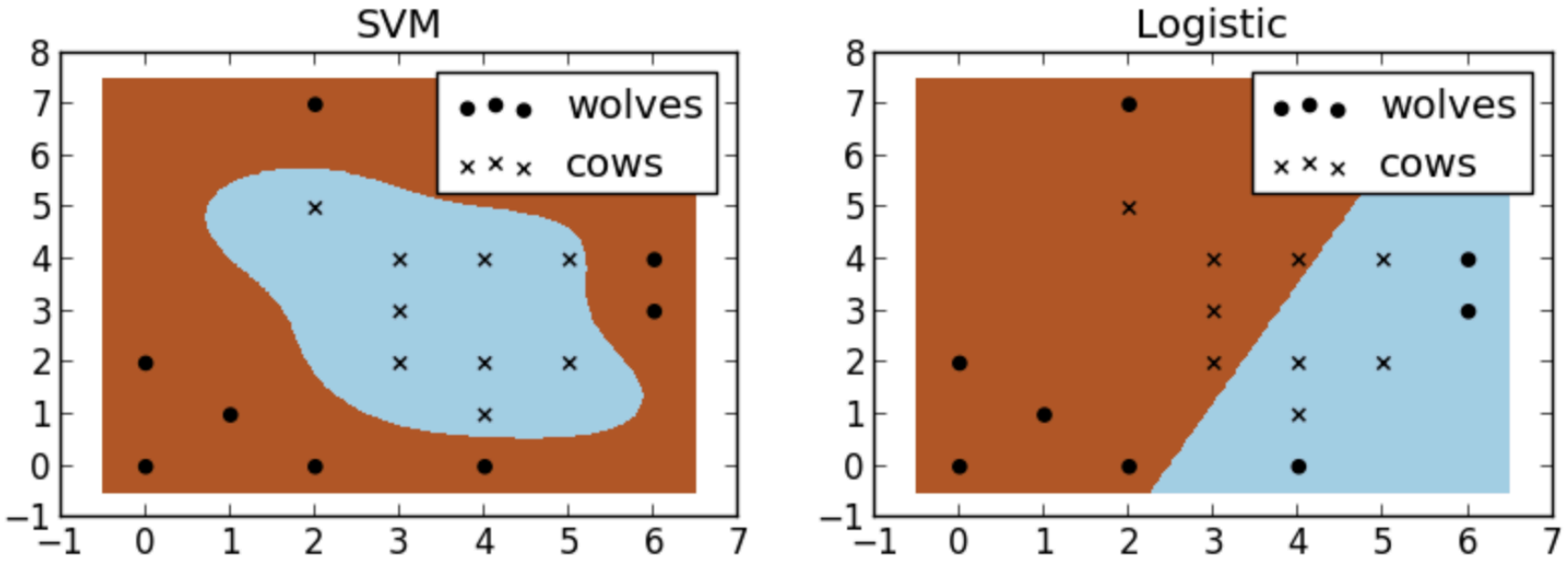
The low-dimensional linearly inseparable data is mapped to the high-dimensional space through the Gaussian kernel function

Support Vector Machine



Support Vector Machine

- Now you are a farmer and you have raised a group of sheep in captivity, but to prevent wolves from attacking the sheep, you need to build a fence to enclose the sheep. But where should the fence be built?



Conclusion

Summary of linear classification models

Model	Function	Loss function	Optimization
Linear regression	-	$(y - \boldsymbol{w}^\top \boldsymbol{x})^2$	Gradient descent
Logistic regression	$\sigma(\boldsymbol{w}^\top \boldsymbol{x})$	$\boldsymbol{y} \log \sigma(\boldsymbol{w}^\top \boldsymbol{x})$	Gradient descent
Softmax regression	$\text{softmax}(\boldsymbol{W}^\top \boldsymbol{x})$	$\boldsymbol{y} \log \text{softmax}(\boldsymbol{W}^\top \boldsymbol{x})$	Gradient descent
Perceptron	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$	$\max(0, -y\boldsymbol{w}^\top \boldsymbol{x})$	Random gradient descent
SVM	$\text{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$	$\max(0, 1 - y\boldsymbol{w}^\top \boldsymbol{x})$	SMO

Feature

http://playground.tensorflow.org

How to deal with non-linear separable problems?