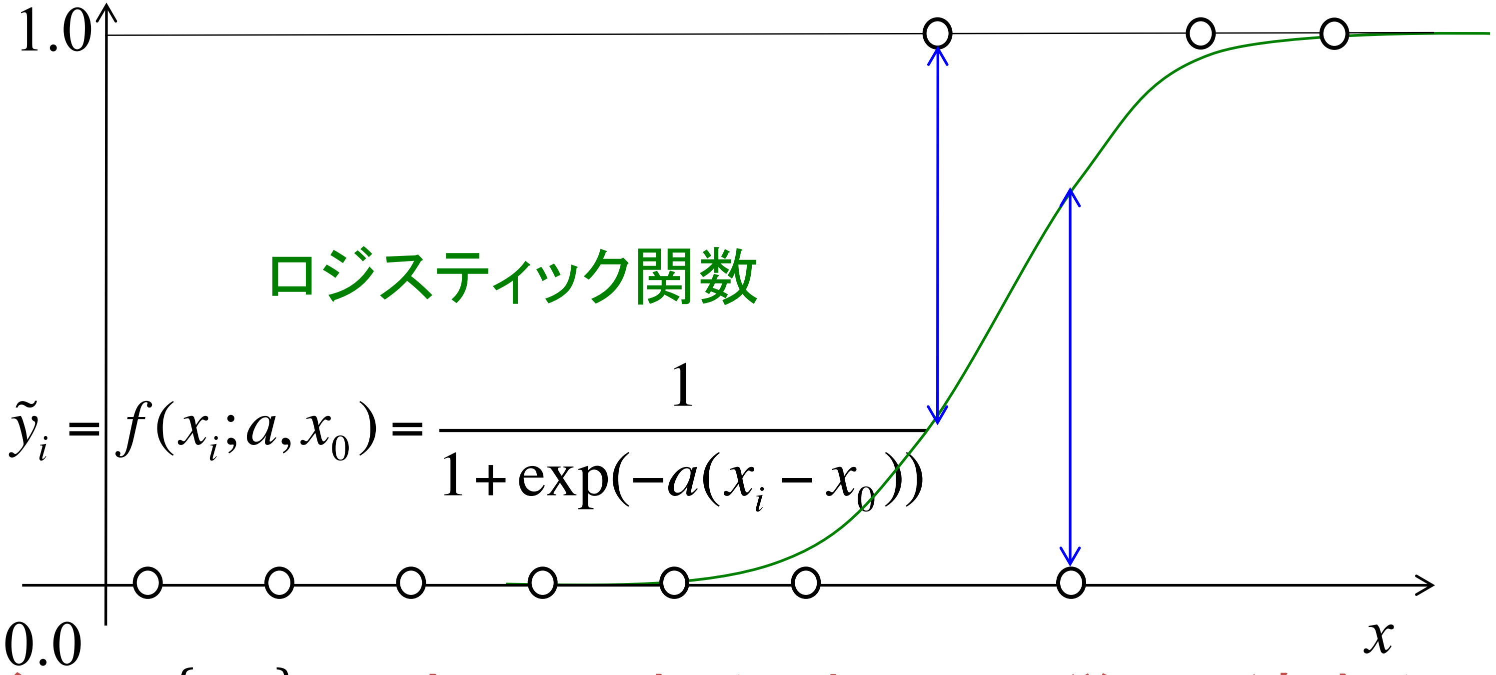
ロジスティック回帰

ロジスティック回帰

与えられたデータに対し，ロジスティック関数を当てはめる問題



$$\tilde{y}_i = f(x_i; a, x_0) = \frac{1}{1+\exp(-a(x_i - x_0))}$$

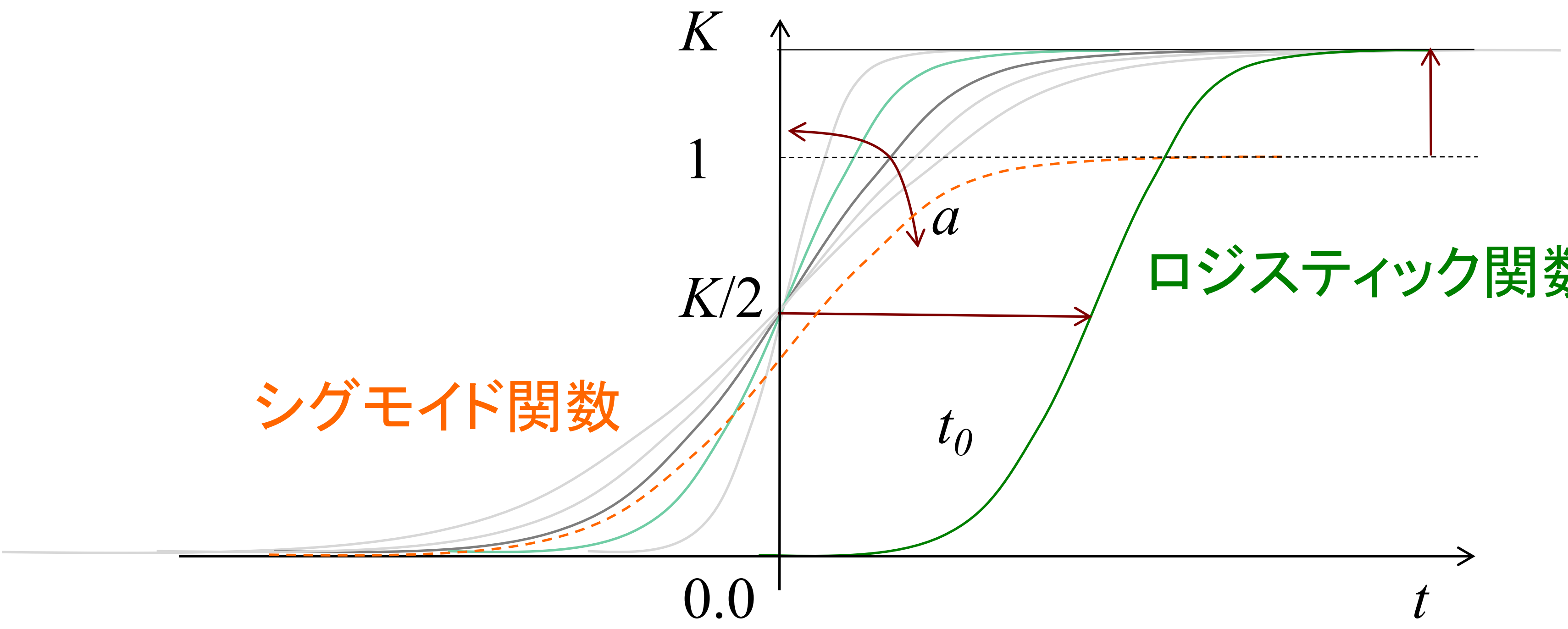
関数の値が $y_i = \{0,1\}$ のとき用いられる。識別器の学習に適する。

クラスA⇔y_i=0,クラスB⇔y_i=1;

if (f(x)<0.5) then クラスA; else クラスB

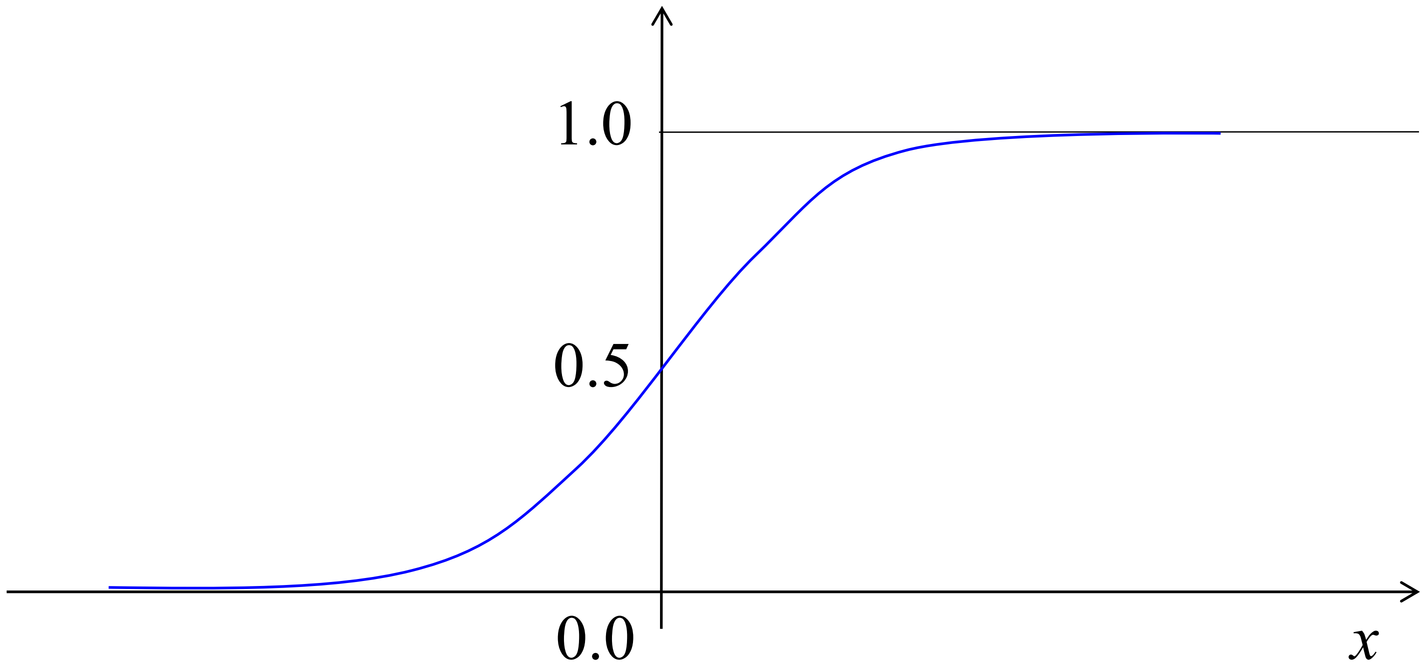
ロジスティック関数とは

$$\frac{dN}{dt} = a\left(\frac{K-N}{K}\right)N, \quad N(x) = \frac{K}{1+\exp(-aK(x-x_0))}$$



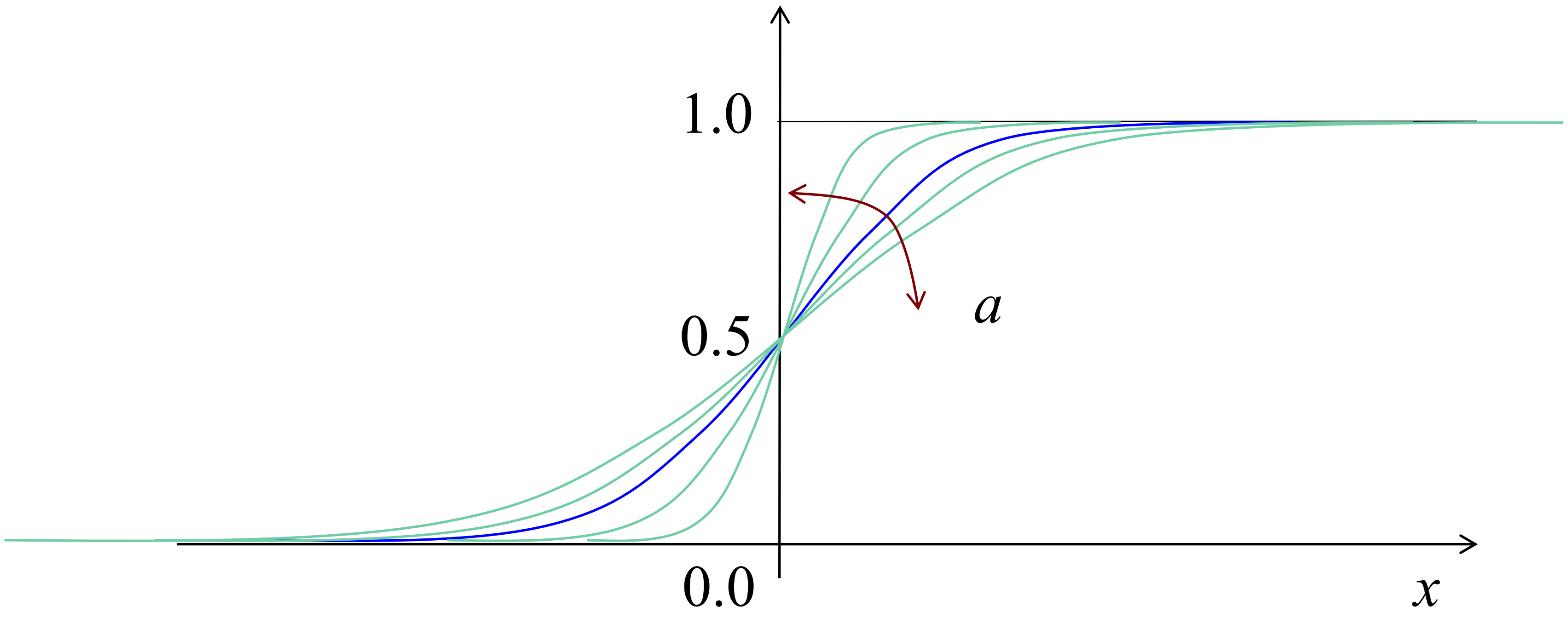
シグモイド関数は，ロジスティック関数のK=1，x_0=0の特殊な場合に相当する

（標準）シグモイド関数



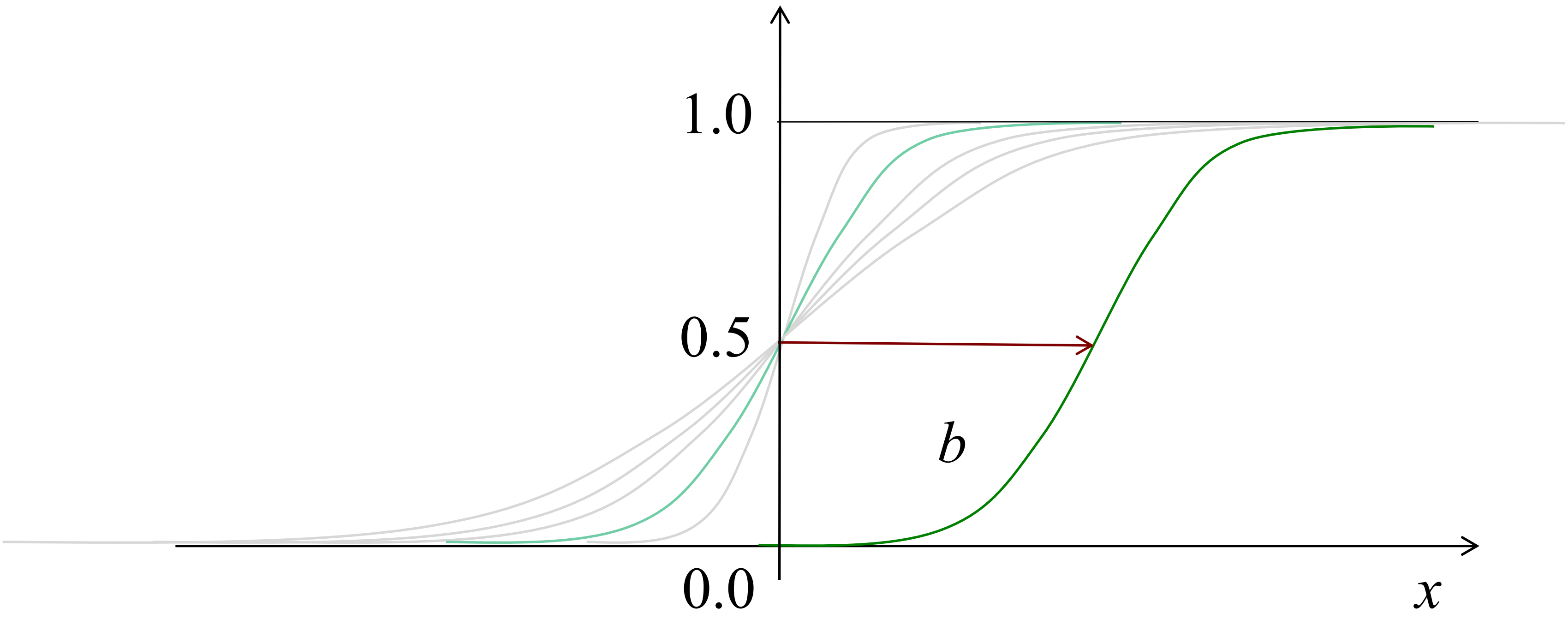
$$f(x) = \frac{1}{1 + \exp(-x)}$$

シグモイド関数



$$f(x) = \frac{1}{1+\exp(-ax)}$$

シグモイド関数の x 軸シフト

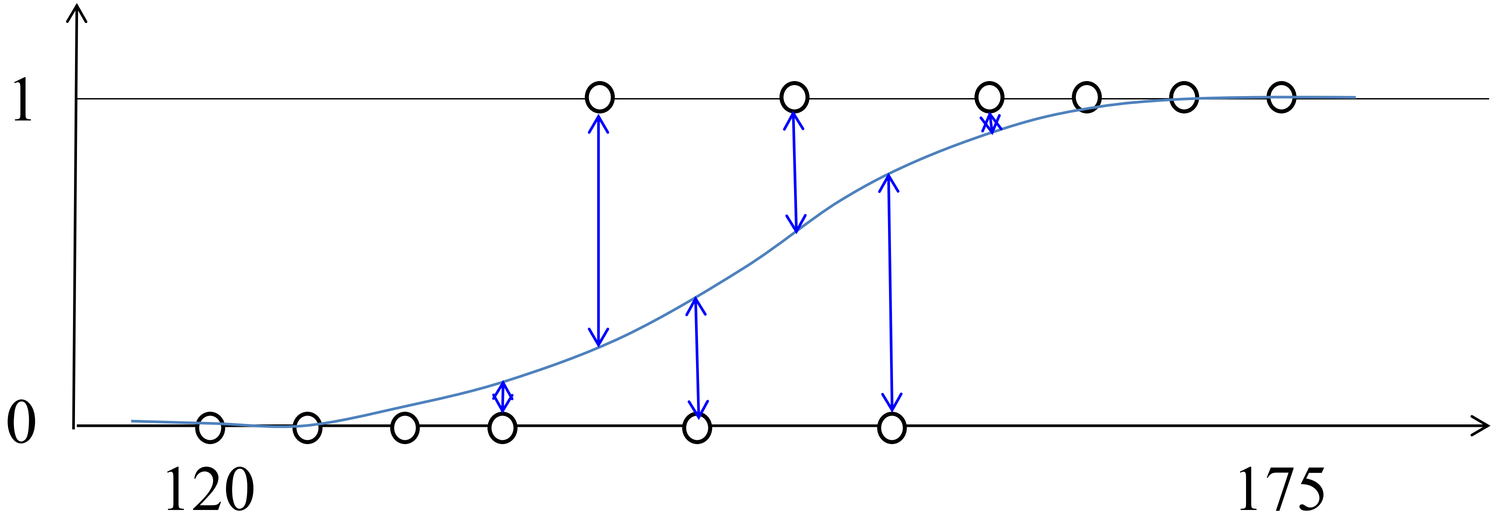


$$f(x) = \frac{1}{1 + \exp(-a(x-b))}$$

例題

下記の学習データに対し，ロジスティック関数（シグモイド関数）を当てはめる。

（1.20，0）（1.25，0）（1.30，0）（1.35，0）
（1.40，1）（1.45，0）（1.50，1）（1.55，0）
（1.60，1）（1.65，1）（1.70，1）（1.75，1）



$(x_n \;\; y_n)$ ： n番目の学習データ

$\boldsymbol{x}_n = (x_n \;\; 1)^T$ ： x_nを次元拡張してベクトル化

$\boldsymbol{w} = (w_1 \; w_2)^T$ ： パラメタ

$\tilde{y}_n = \dfrac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$ ： 予測式

$$e_n{}^2 = \frac{1}{2}(\tilde{y}_n - y_n)^2 = \frac{1}{2}\left(\frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})} - y_n\right)^2$$

$$E = \frac{1}{2}\sum_{n=1}^{N}(\tilde{y}_n - y_n)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(\frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})} - y_n\right)^2$$

- ロジスティック回帰の場合，解は解析的に求まらない。

⇒　反復法の必要性

勾配法：

　　近似解を与え，

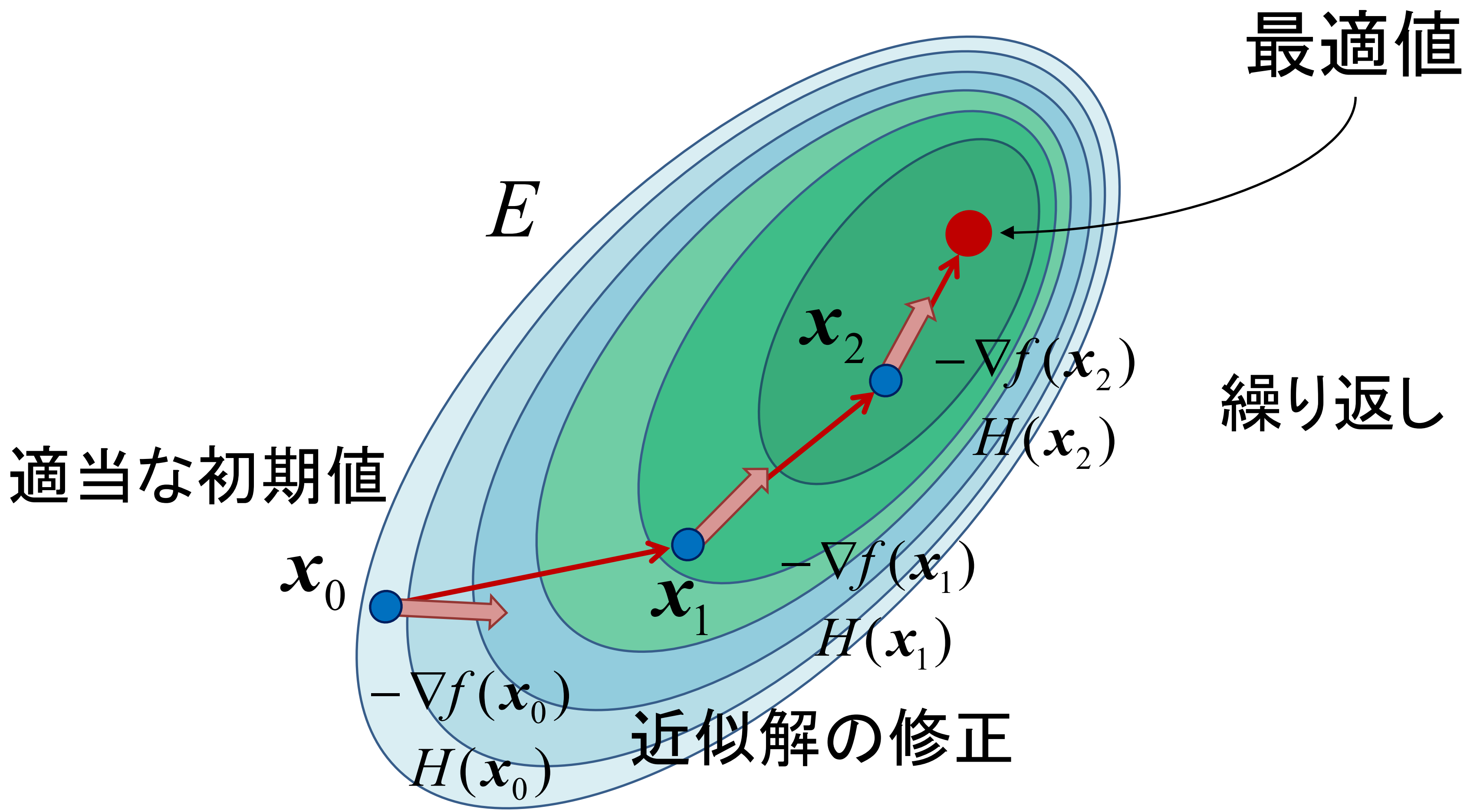
　　近似解の周辺の傾き等を用いて，

　　近似解を漸次改良する

　　　　この一連の処理を繰り返す。

最急降下法，共役勾配法，ニュートン法

勾配法

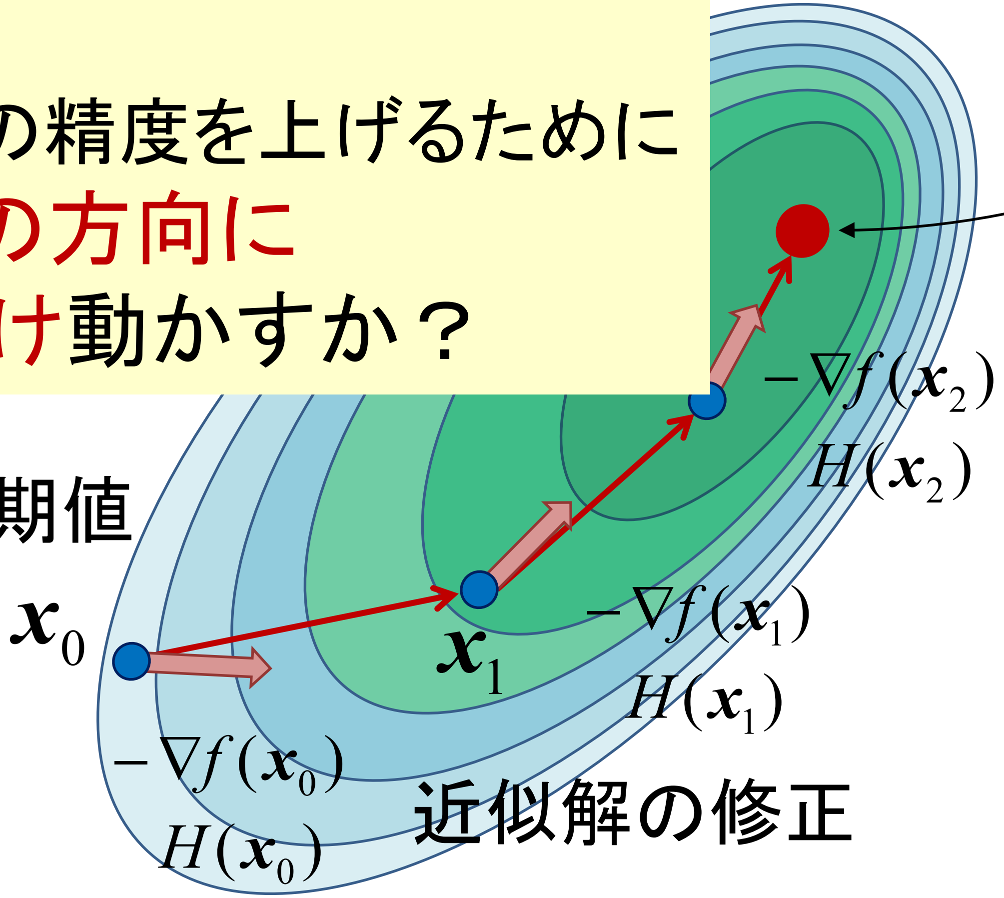


勾配（，ヘシアン）（Eのx_0における1次（，２次）の微分）

勾配法

問題：

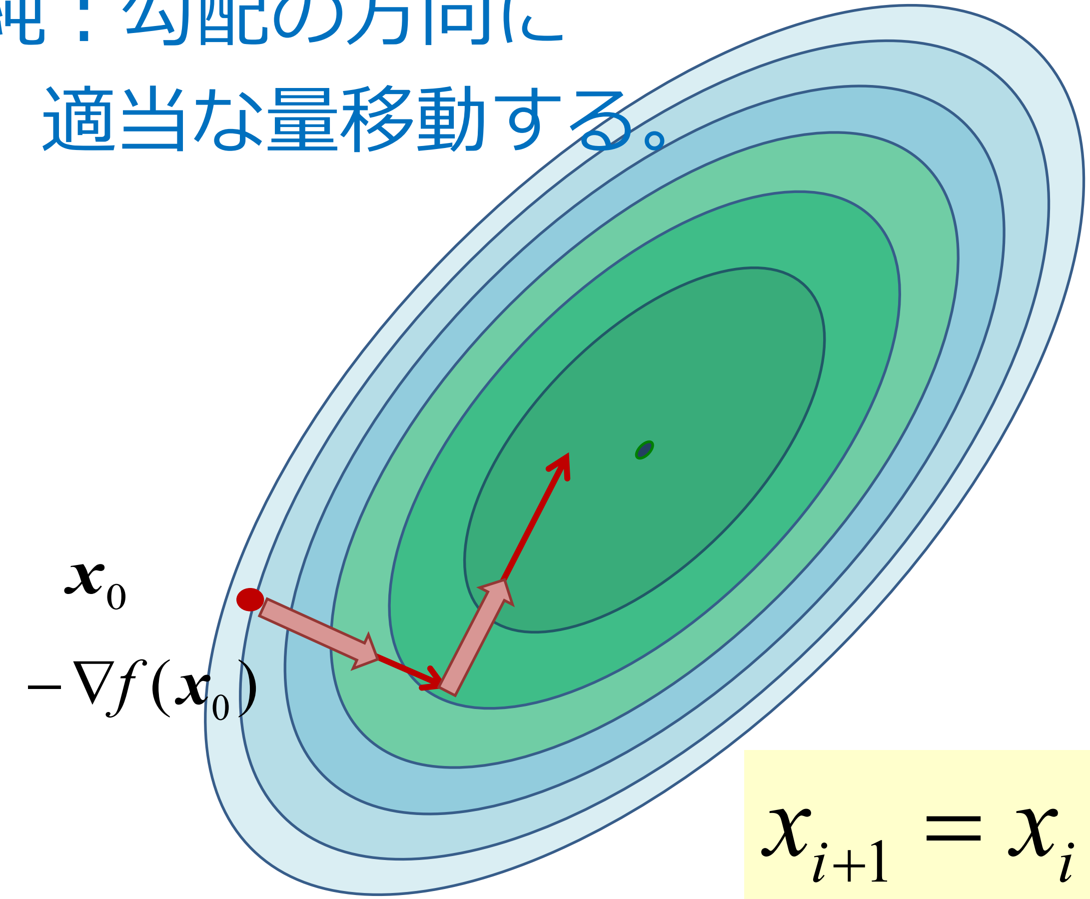
近似解の精度を上げるために
どちらの方向に
どれだけ動かすか？



勾配（，ヘシアン）（Eのx_0における1次（，２次）の微分）

最急降下法

- 最急降下法：
  - 最も単純：勾配の方向に適当な量移動する。



αの決め方は様々
- 決まった量移動
- 過去の勾配の大きさの履歴に応じて

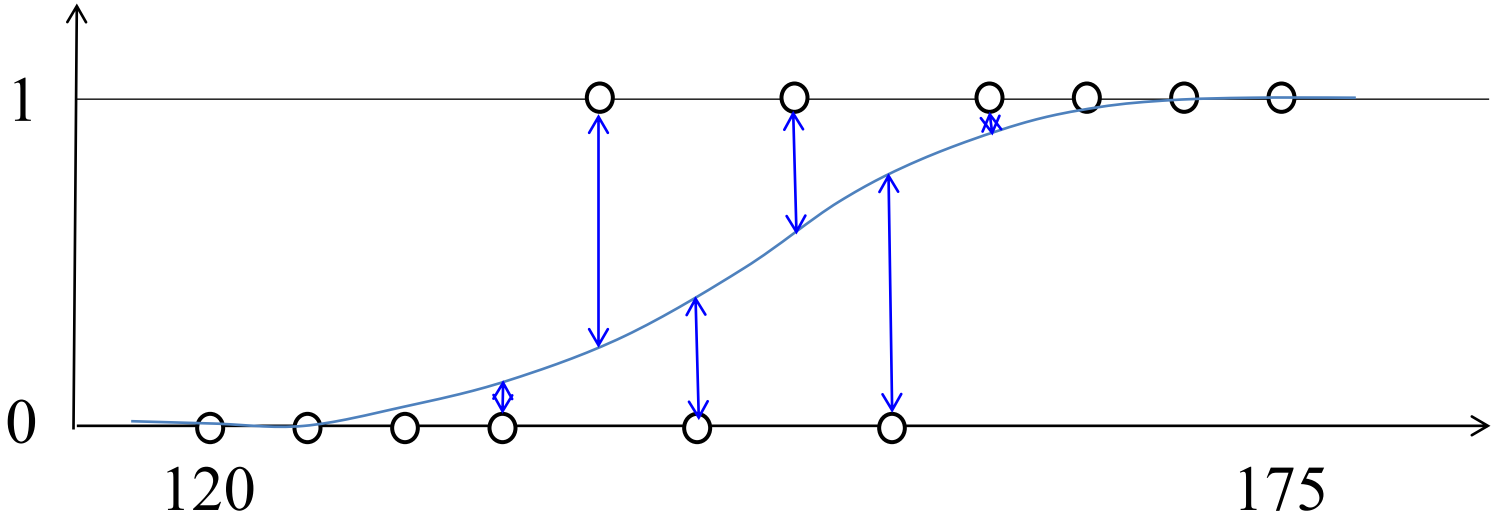
$$x_{i+1} = x_i - \alpha \nabla f(\boldsymbol{x}_i)$$

例題

再掲

下記の学習データに対し，ロジスティック関数（シグモイド関数）を当てはめる。

（1.20，0）	（1.25，0）	（1.30，0）	（1.35，0）
（1.40，1）	（1.45，0）	（1.50，1）	（1.55，0）
（1.60，1）	（1.65，1）	（1.70，1）	（1.75，1）



$$e_n{}^2 = \frac{1}{2}(\tilde{y}_n - y_n)^2 = \frac{1}{2}\left(\frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})} - y_n\right)^2$$

$$\frac{\partial e_n{}^2}{\partial \boldsymbol{w}} = (\tilde{y}_n - y_n)\tilde{y}_n(1-\tilde{y}_n)\boldsymbol{x}$$

$$\frac{\partial}{\partial \boldsymbol{w}}E = \sum_{n=1}^{N}(\tilde{y}_n - y_n)\tilde{y}_n(1-\tilde{y}_n)\boldsymbol{x}$$

- On-line 学習： サンプル毎にパラメタを更新

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{prv} - \alpha\frac{\partial e_n{}^2}{\partial \boldsymbol{w}}$$

- バッチ学習 ：学習データ一括でパラメタを更新

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{prv} - \alpha\frac{\partial E}{\partial \boldsymbol{w}}$$

疑似コーディングの例

```
パラメタの初期値を w に与える。
収束するまで繰り返し {
        j=1 から順に，学習データの数 N まで繰り返し {
                grad = データ xj における傾き;          // 修正の方向
                学習率 α を決める;                       // 修正の量
                w = w – α  * grad ;                      // wの更新
        }
        収束していたら， wを解として終了;
}
```

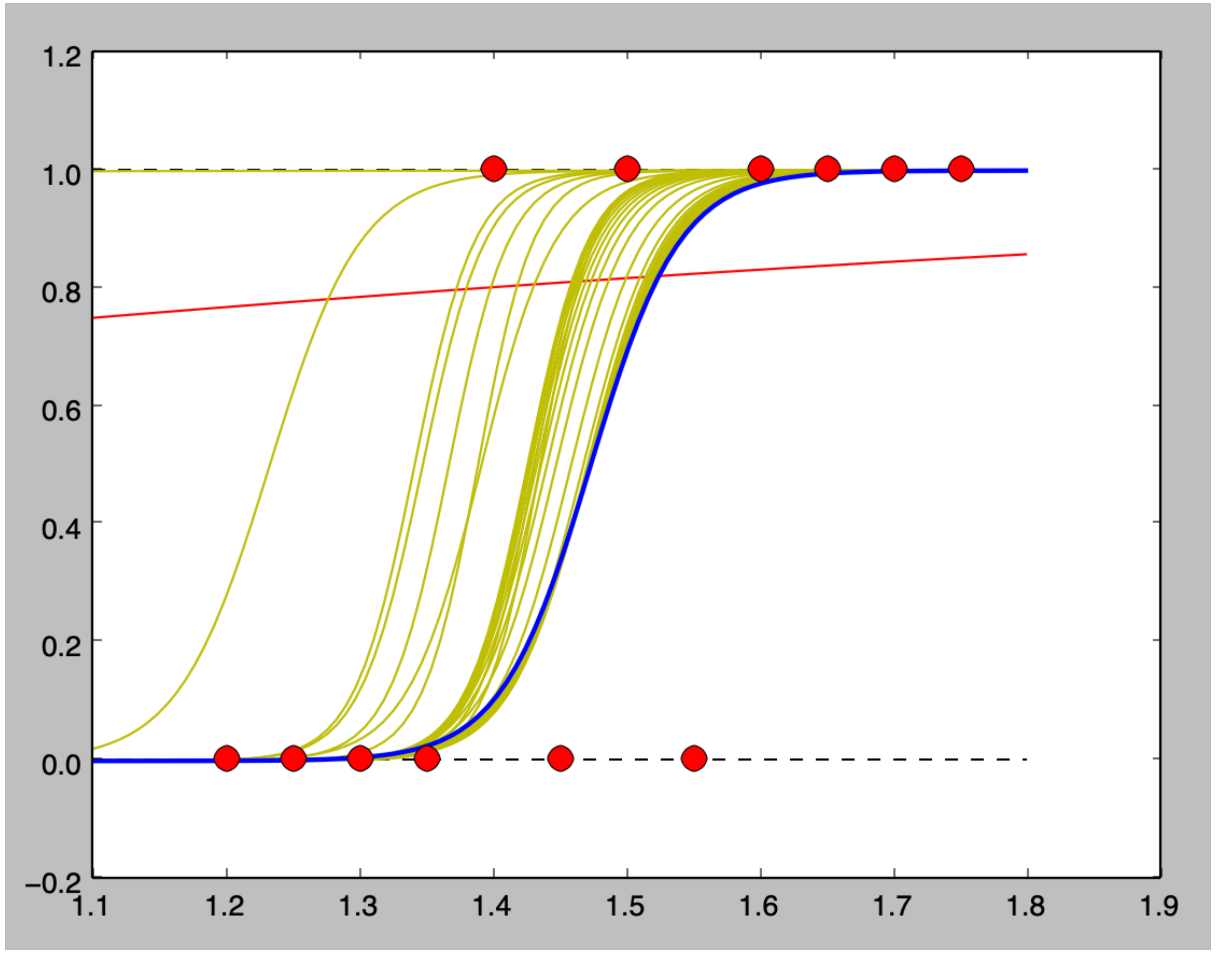
プログラムの例 1

```
import numpy as np
from scipy import linalg as la
import matplotlib.pyplot as plt
import copy as cp

def sigm(z):
    return 1/(1+np.exp(-z))

trainingData =
np.array([[1.20,0],[1.25,0],[1.30,0],[1.35,0],[1.40,1],[1.45,0],[1.50,1
],[1.55,0],[1.60,1],[1.65,1],[1.70,1],[1.75,1]])

N,nd = trainingData.shape      # N : number of samples
                                # nd: dimension
```

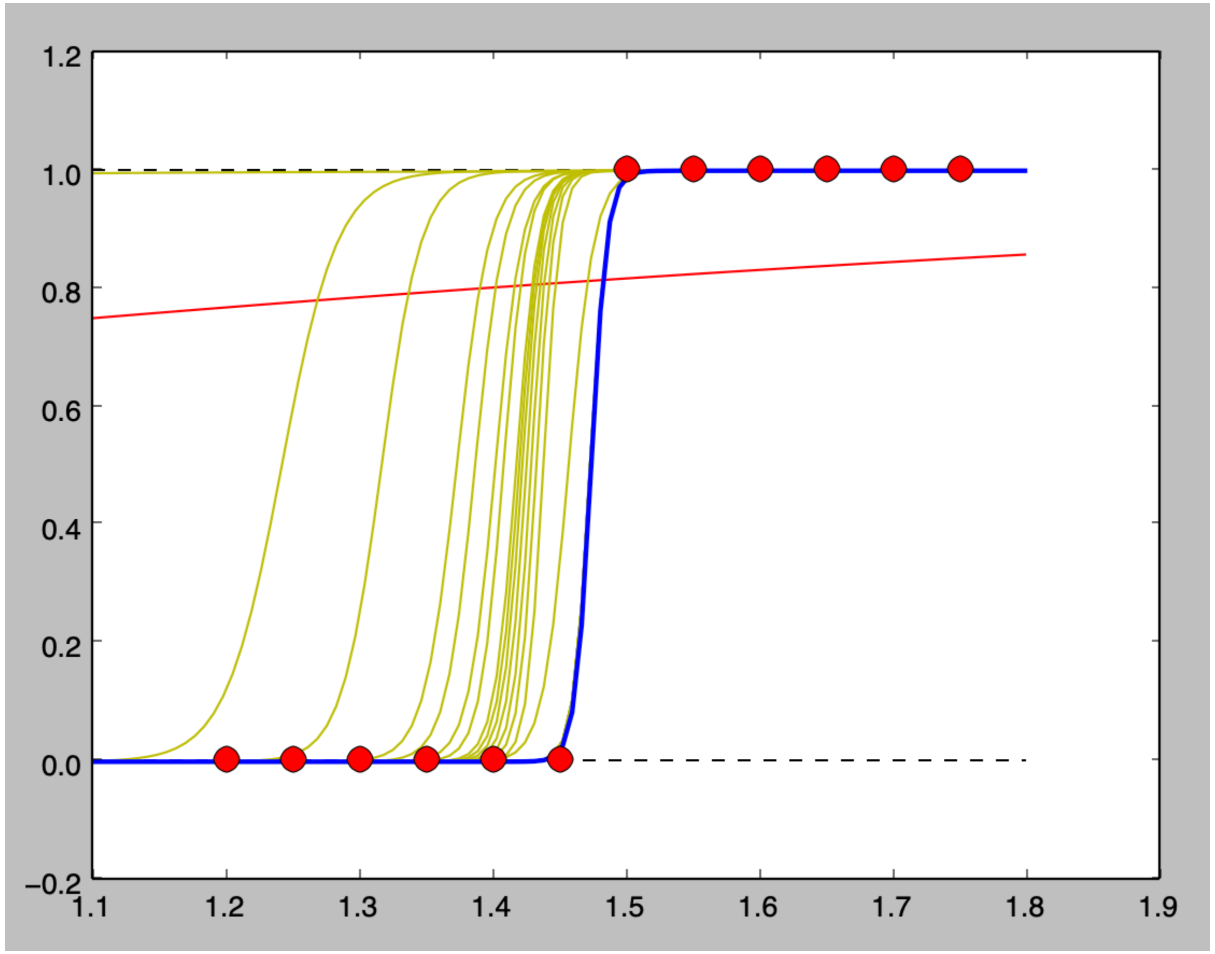
```
true    = trainingData[:,1]        # true value
D = cp.copy(trainingData)
D[:,nd-1] = np.ones(N)             # Data Matrix

TH = 1.0e-7          # threshold for iteration
alpha = 5.0          # learning rate
alpDec = 0.9999      # parameter for alpha decay

nPlot = 100                              # plot resolution
pltXaxis = np.linspace(1.1,1.8,nPlot)    # x-data for plot
xx = np.ones((nPlot,nd))                 # data for plot
xx[:,0] = pltXaxis                       # data for plot

w = np.array([1.0,0.0])     # initial model parameters
plt.plot(pltXaxis, sigm(xx.dot(w)),color="r",linewidth=1)
```

```
errPrv = 10.0
for i in range(100000):
    for j in range(N):
        zj = sigm(D[j,:].dot(w))
        grad = (zj-true[j])*zj*(1-zj)*D[j,:]
        w = w - alpha*grad
        alpha = alpha*alpDec

    estimate = sigm(D.dot(w))
    errNew = np.sqrt((true-estimate).dot((true-estimate)))
    if (abs(errNew-errPrv) < TH): break
    errPrv = errNew

    if (i%100 == 0):
        plt.plot(pltXaxis, sigm(xx.dot(w)),color="y")
        print i,w,errNew
```

```
plt.plot(pltXaxis, sigm(xx.dot(w)),color="b",linewidth=2)

plt.ylim(-0.2, 1.2)
plt.hlines([0, 1], 1.1, 1.8, linestyles="dashed")
plt.plot(trainingData[:,0],true,"ro",markersize=10)
plt.show()
```



iteration:6901, error: 1.35668



iteration:6162, error: 0.17588

クロスエントロピー基準での最適化

$(x_n \ \ y_n)$: n番目の学習データ

$\boldsymbol{x}_n = (x_n \ \ 1)^T$: x_nを次元拡張してベクトル化

$\boldsymbol{w} = (w_1 \ w_2)^T$: パラメタ

$$\tilde{y}_n = \frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}$$: 予測式

最小二乗基準

$$e_n = \frac{1}{2}(\tilde{y}_n - y_n)^2 = \frac{1}{2}\left(\frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})} - y_n\right)^2$$

$$\frac{\partial e_n}{\partial \boldsymbol{w}} = (\tilde{y}_n - y_n)\tilde{y}_n(1-\tilde{y}_n)\boldsymbol{x}$$

クロスエントロピー基準での最適化

- クロスエントロピー基準

n番目のデータ x_n がクラス 0,1 に属する確率を p_0, p_1,
その推定値をq_0, q_1とする。
ここで，

$$p_1 = y_n, p_0 = 1 - y_n,$$

$$q_1 = \tilde{y}_n = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}, q_0 = 1 - \tilde{y}_n$$

とすると，クロスエントロピーは，以下のように表される。

$$e_n = -\sum_{k=0}^{1} p_k \log q_k = -y_n \log \tilde{y}_n - (1 - y_n) \log(1 - \tilde{y}_n)$$

最小二乗基準とクロスエントロピー基準

$e_n = -y_n \log \tilde{y}_n - (1-y_n)\log(1-\tilde{y}_n)$ より

$$\frac{\partial e_n}{\partial \boldsymbol{w}} = -y_n \frac{1}{\tilde{y}_n}\frac{\partial \tilde{y}_n}{\partial \boldsymbol{w}} - (1-y_n)\frac{1}{1-\tilde{y}_n}\left(-\frac{\partial \tilde{y}_n}{\partial \boldsymbol{w}}\right) \quad (1)$$

$\tilde{y}_n = \dfrac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})}$ より

$$\frac{\partial \tilde{y}_n}{\partial \boldsymbol{w}} = \tilde{y}_n(1-\tilde{y}_n)\boldsymbol{x} \quad (2)$$

(2)を(1)に代入して,

$$\frac{\partial e_n}{\partial \boldsymbol{w}} = (\tilde{y}_n - y_n)\boldsymbol{x}, \qquad \boldsymbol{w}_{new} = \boldsymbol{w}_{prv} - \alpha\frac{\partial e_n}{\partial \boldsymbol{w}}$$

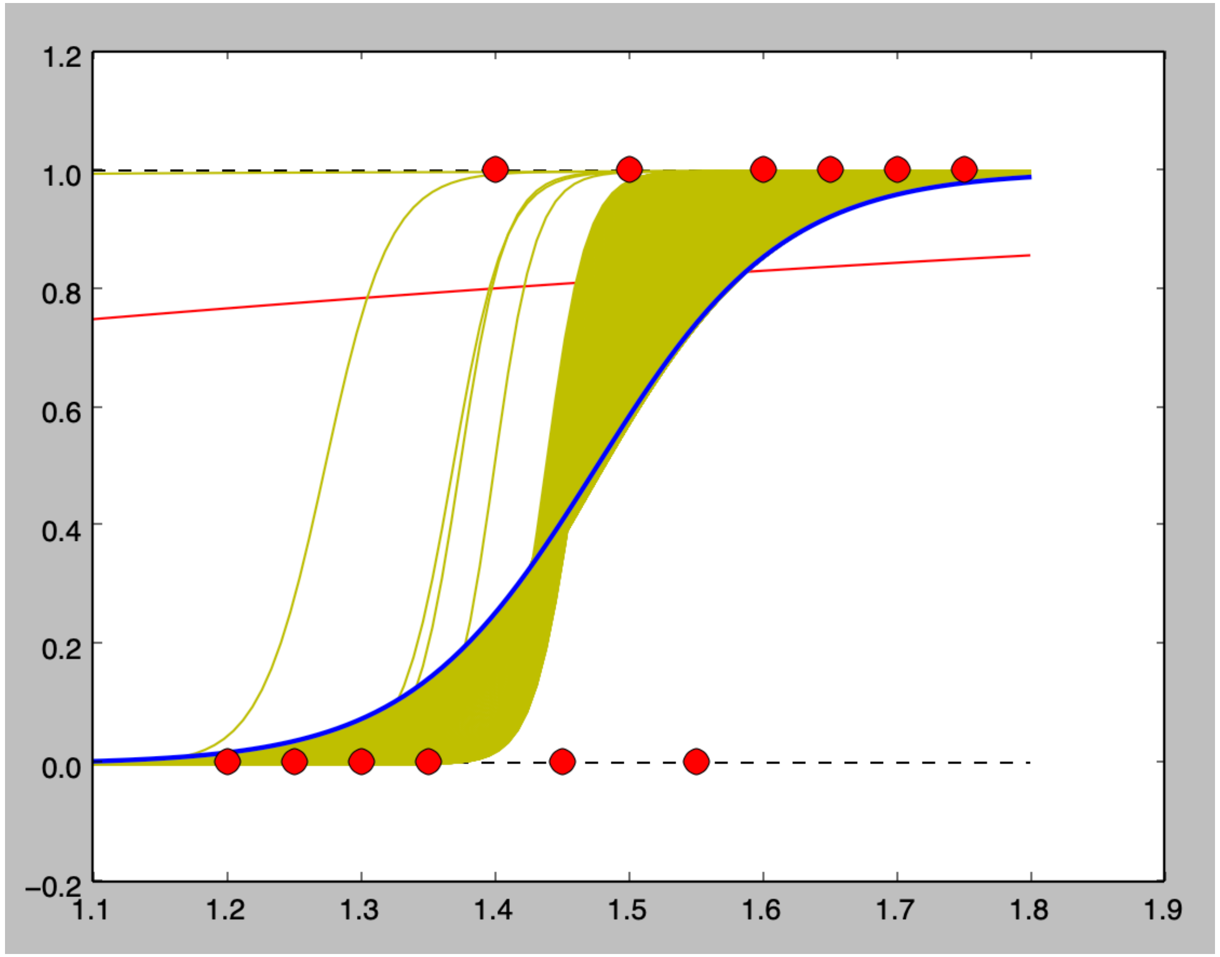
プログラムの例 3

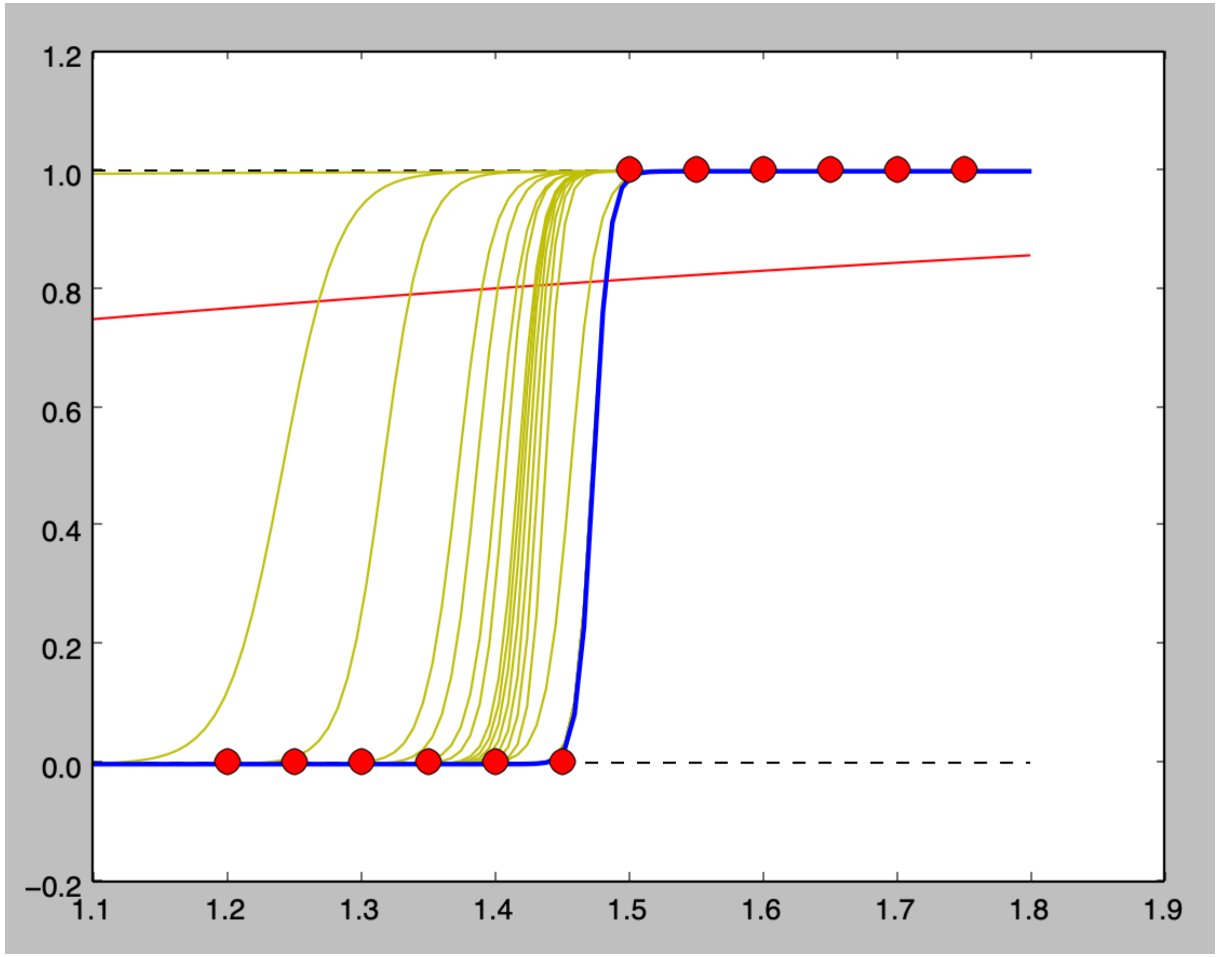
```
import numpy as np
from scipy import linalg as la
import matplotlib.pyplot as plt
import copy as cp

def sigm(z):
    return 1/(1+np.exp(-z))

trainingData = 
np.array([[1.20,0],[1.25,0],[1.30,0],[1.35,0],[1.40,1],[1.45,0],[1.50,1
],[1.55,0],[1.60,1],[1.65,1],[1.70,1],[1.75,1]])

N,nd = trainingData.shape       # N : number of samples
                                # nd: dimension
```

```
true    = trainingData[:,1]          # true value
D = cp.copy(trainingData)
D[:,nd-1] = np.ones(N)               # Data Matrix

TH = 1.0e-7                          # threshold for iteration
alpha = 5                            # learning rat
alpDec = 0.99999                     # parameter for alpha decay
epcilon = 1.0e-7

nPlot = 100                          # plot resolution
pltXaxis = np.linspace(1.1,1.8,nPlot) # x-data for plot
xx = np.ones((nPlot,nd))             # data for plot
xx[:,0] = pltXaxis                   # data for plot

errPrv =100.0
w = np.array([1.0,0.0])              # model parameters
```

```
pltData = sigm(xx.dot(w))
plt.plot(pltXaxis,pltData,color="r",linewidth=1)

for i in range(100000):
    for j in range(N):
        zj = sigm(D[j,:].dot(w))
        grad = (zj-true[j])*(D[j,:])
        w = w - alpha*grad
        alpha = alpha * alpDec

    estmt = sigm(D.dot(w))
    errNew = -true.dot(np.log(estmt))-(1-true).dot(np.log(1-estmt))
    if (abs(errNew-errPrv) < TH): break
    errPrv = errNew
```

```
    if (i%100 == 0):
        pltData = sigm(xx.dot(w))
        plt.plot(pltXaxis,pltData,color="y")
        print i,w,errNew

print i,w,errNew,alpha
pltData = sigm(xx.dot(w))
plt.plot(pltXaxis,pltData,color="b",linewidth=2)

plt.ylim(-0.2, 1.2)
plt.hlines([0, 1], 1.1, 1.8, linestyles="dashed")
plt.plot(trainingData[:,0],true,"ro",markersize=10)
plt.show()
```





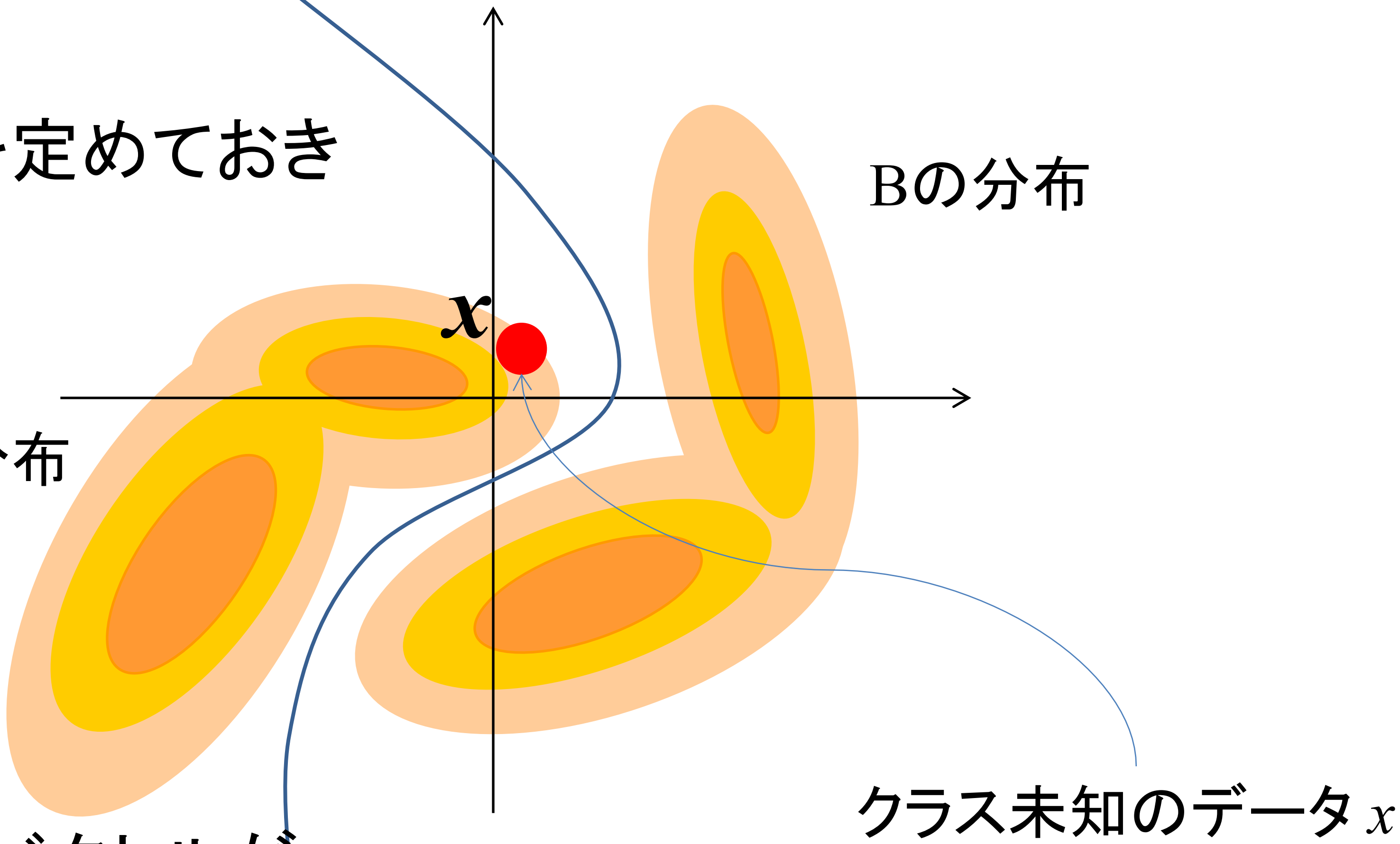
31837, 4.37879

18096, 0.02976

線形識別と
ロジスティック回帰

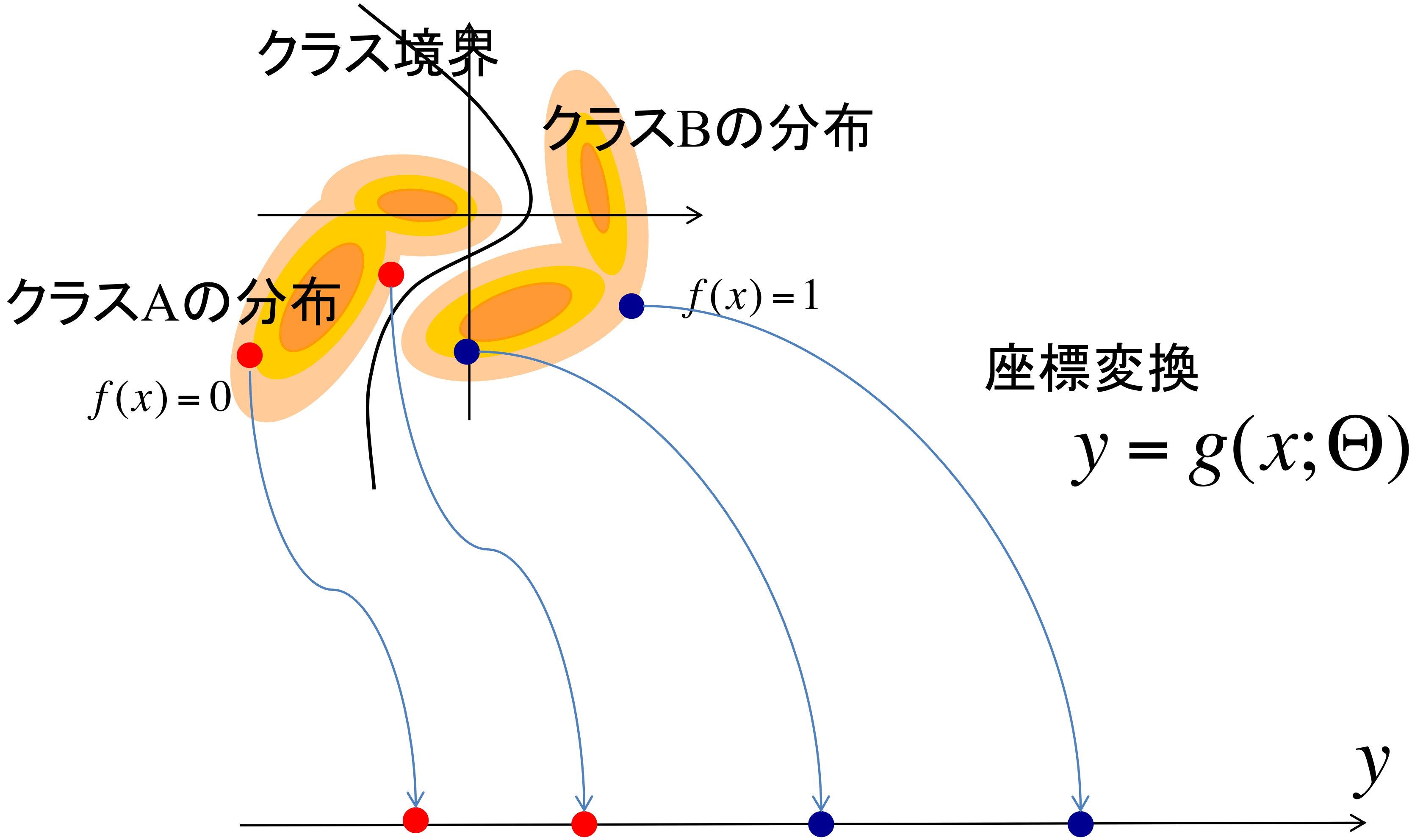
パターン認識の方法

1. 分布をもとに
   クラスの境界を定めておき

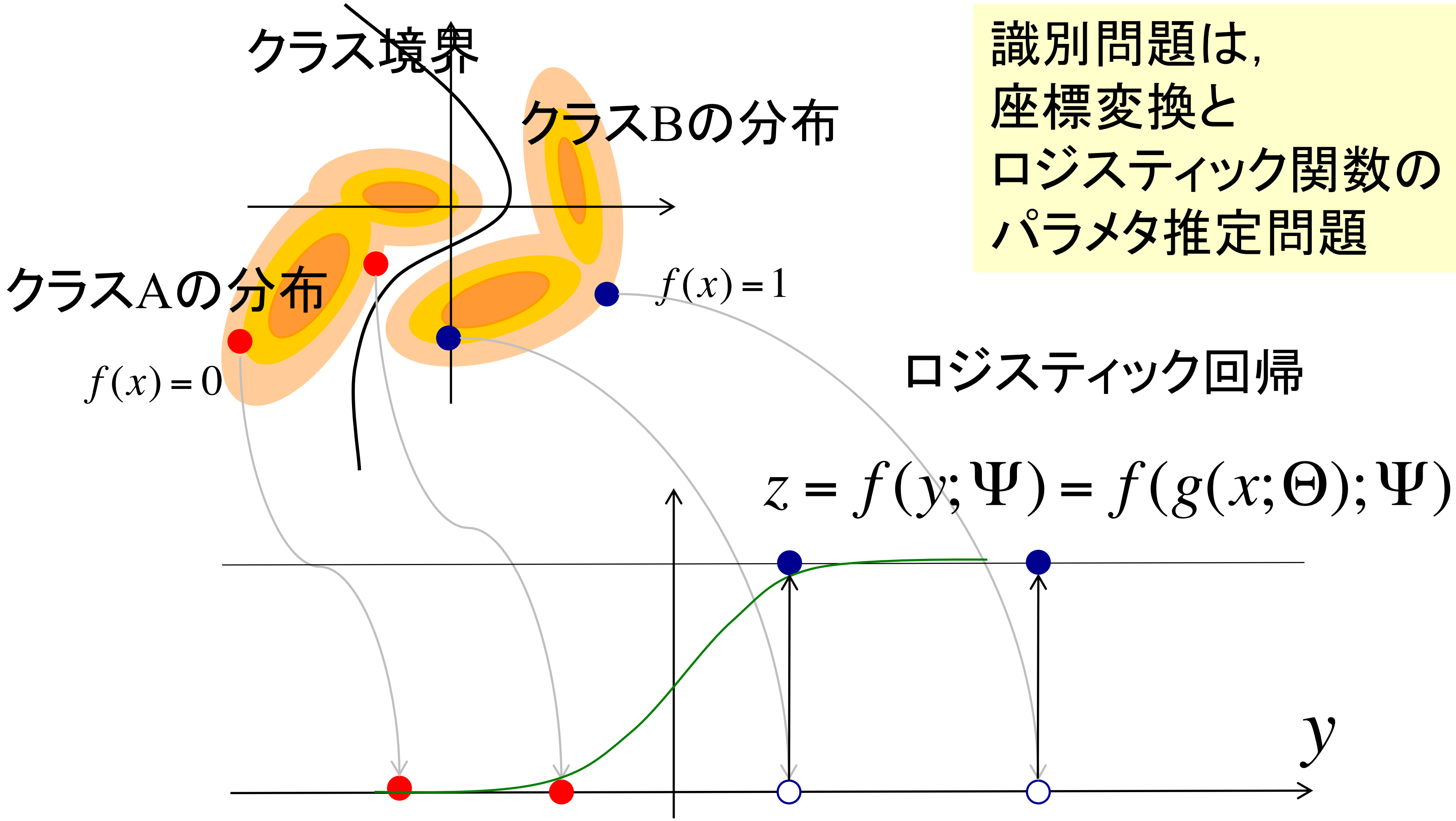


2. テストデータのベクトルが
   境界のどちら側にあるかを調べることで，
   テストデータのクラスを定める

典型的な識別器の設計法＝ロジスティック回帰

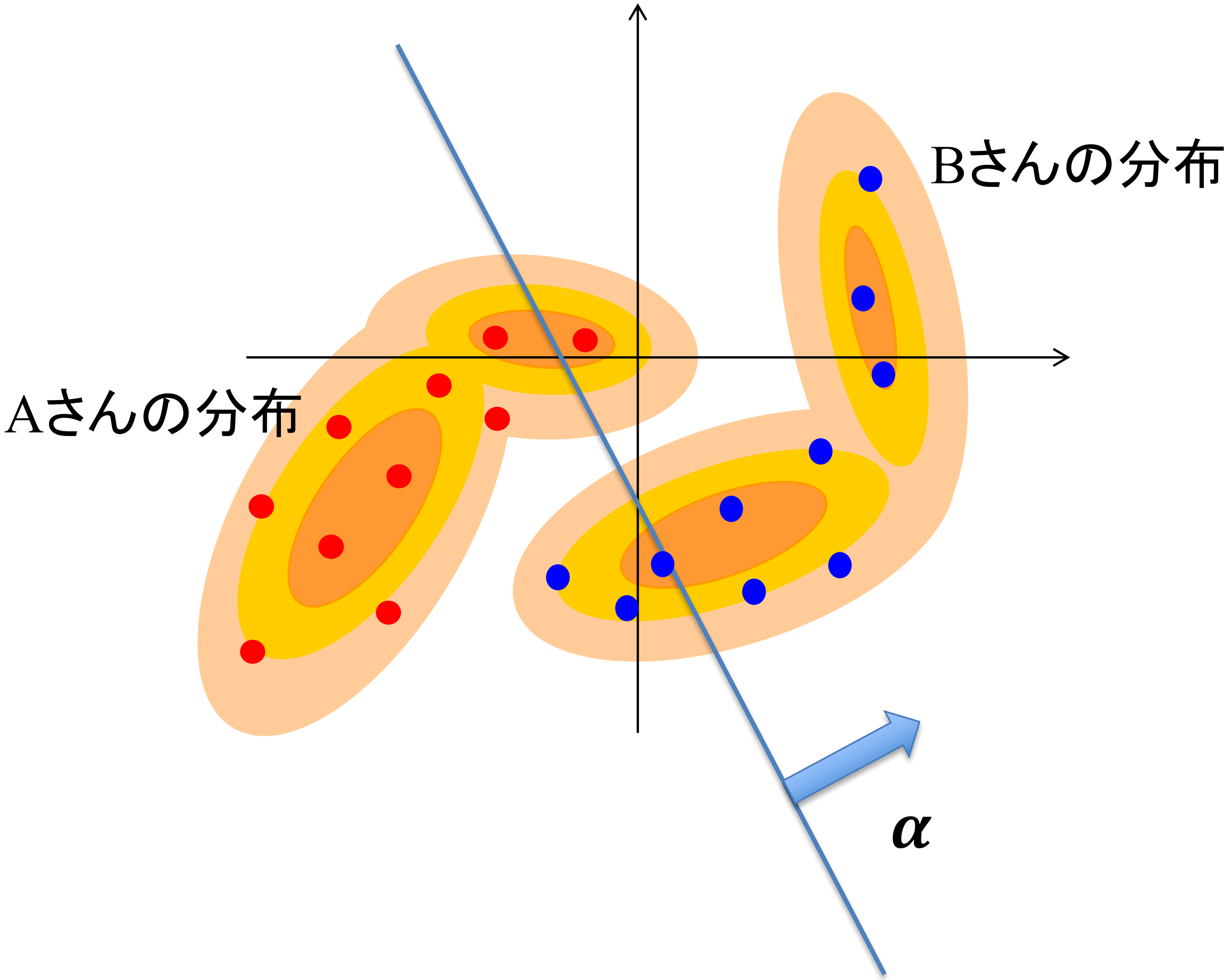


典型的な識別器の設計法＝ロジスティック回帰

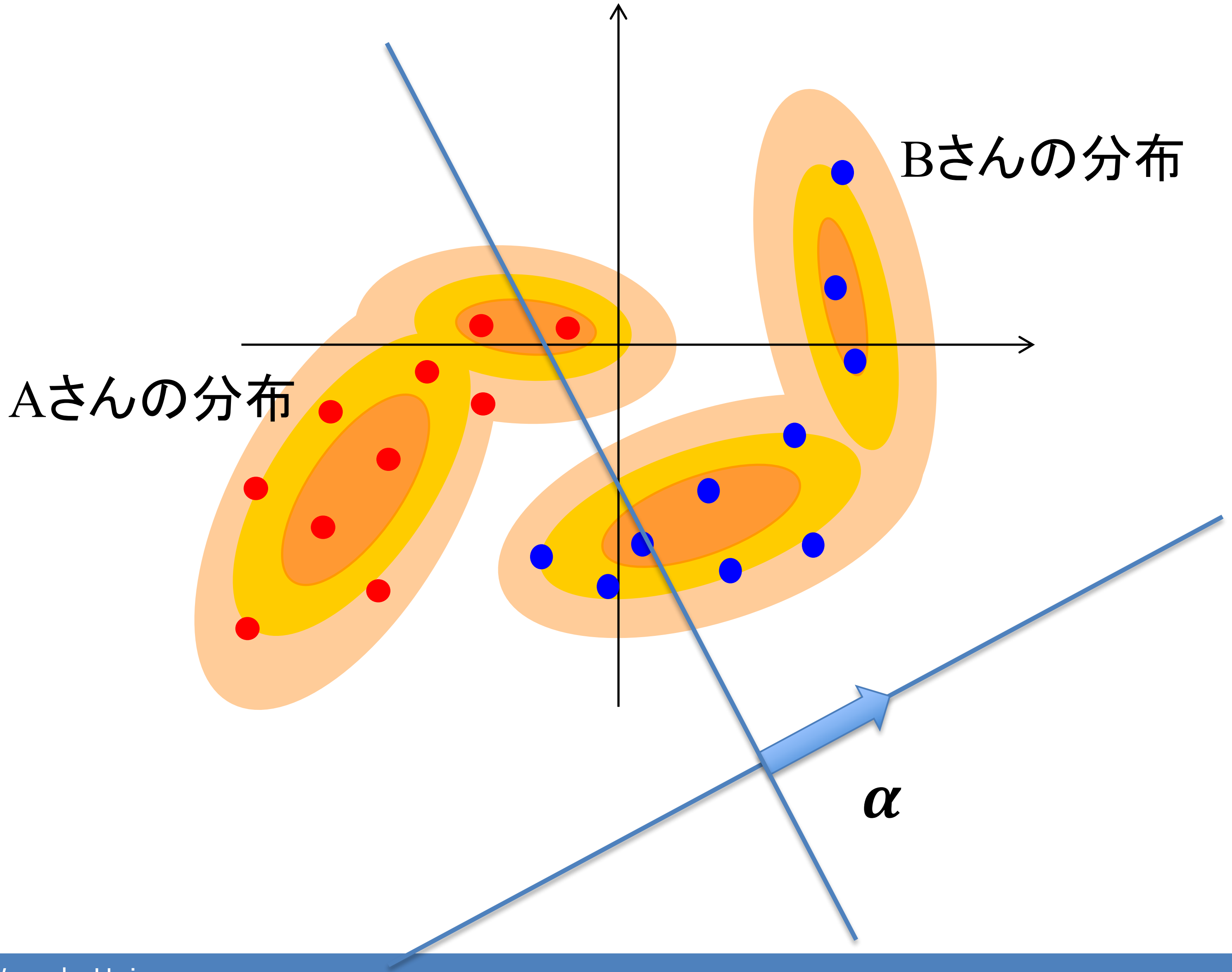


識別問題は，
座標変換と
ロジスティック関数の
パラメタ推定問題

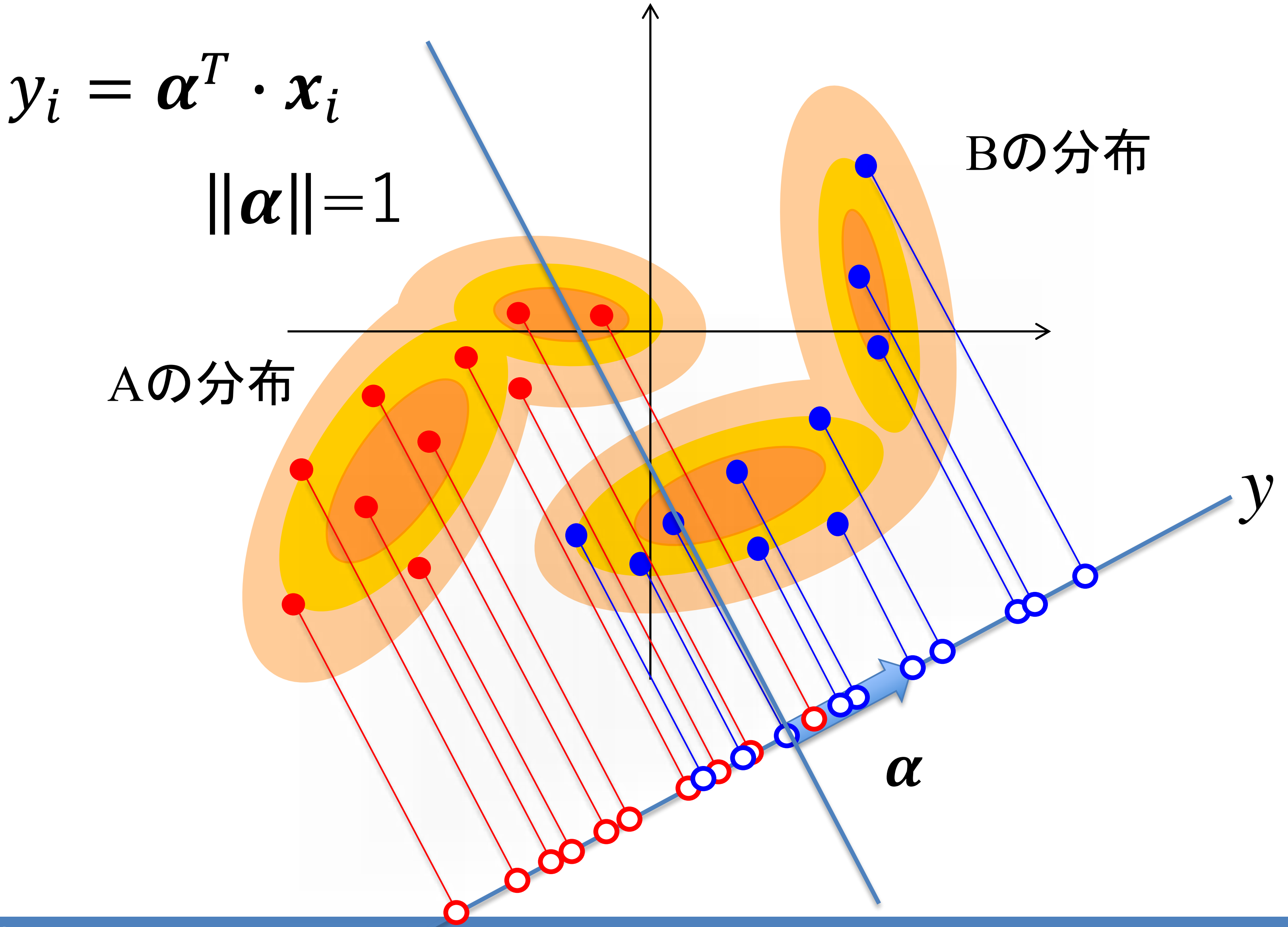
線形識別の場合



線形識別の場合



線形識別の場合

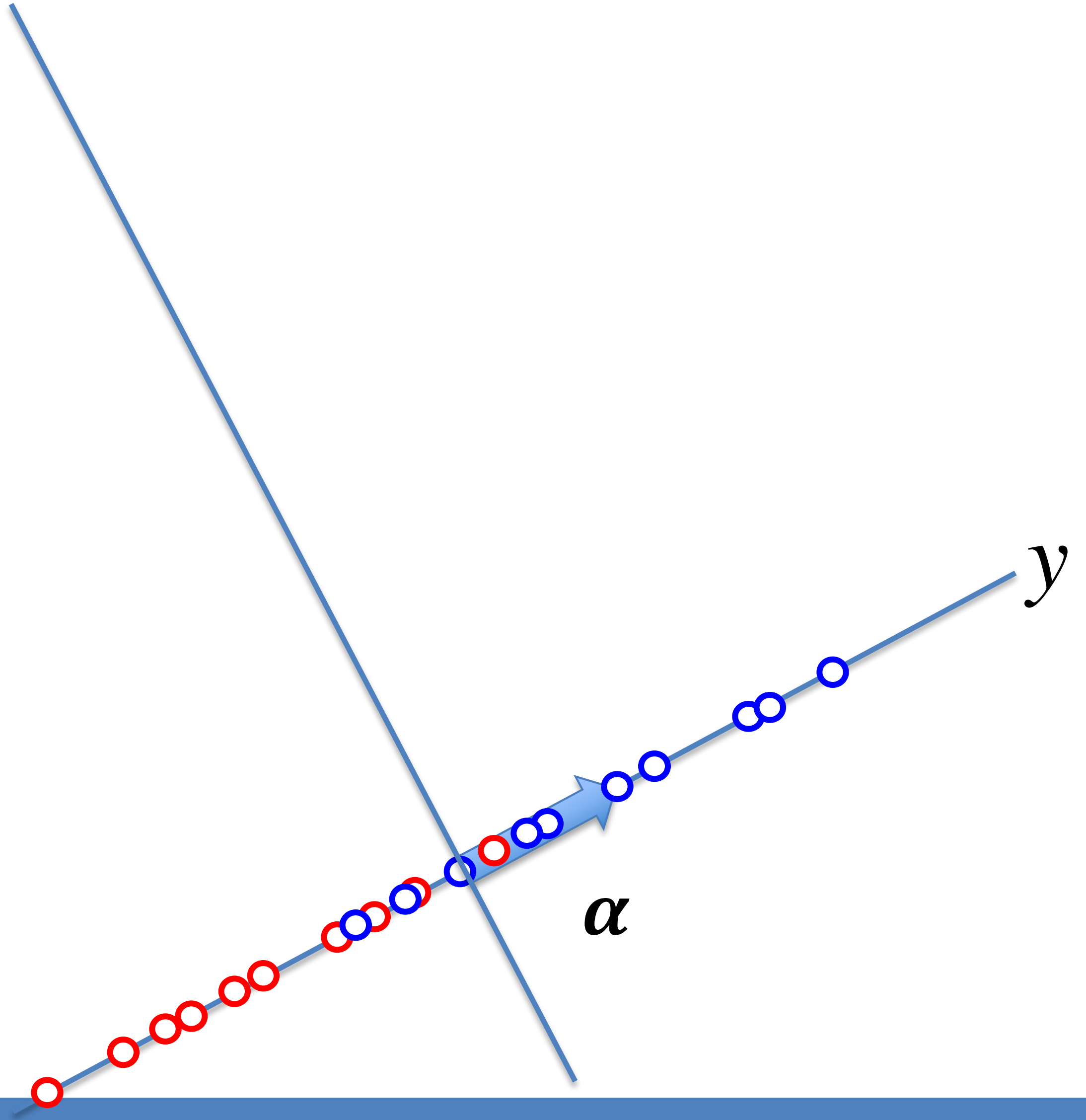


$y_i = \boldsymbol{\alpha}^T \cdot \boldsymbol{x}_i$
$\|\boldsymbol{\alpha}\|=1$
Bの分布
Aの分布
y
$\boldsymbol{\alpha}$

線形識別の場合

$$y_i = \boldsymbol{\alpha}^T \cdot \boldsymbol{x}_i$$

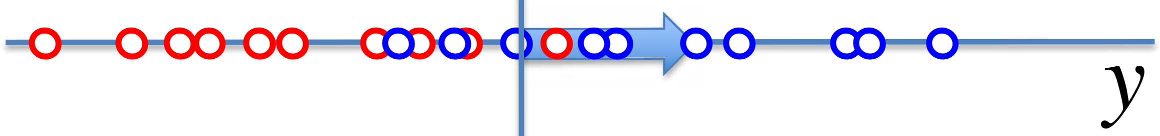
$$\|\boldsymbol{\alpha}\| = 1$$



線形識別の場合

$$y_i = \boldsymbol{\alpha}^T \cdot \boldsymbol{x}_i$$

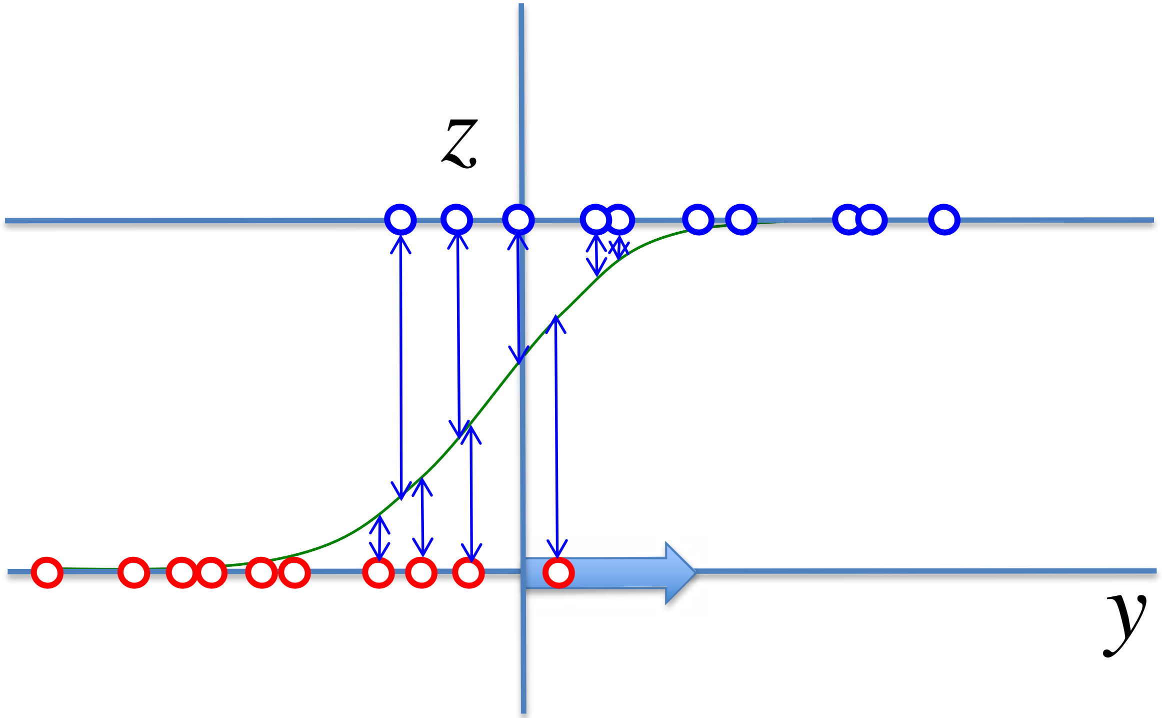
$$\|\boldsymbol{\alpha}\| = 1$$



線形識別の場合

$$y_i = \boldsymbol{\alpha}^T \cdot \boldsymbol{x}_i \qquad \|\boldsymbol{\alpha}\|=1$$

$$z_i = \frac{1}{1+\exp(-a \cdot y_i)} = \frac{1}{1+\exp(-a \cdot \boldsymbol{\alpha}^T \boldsymbol{x}_i)}$$

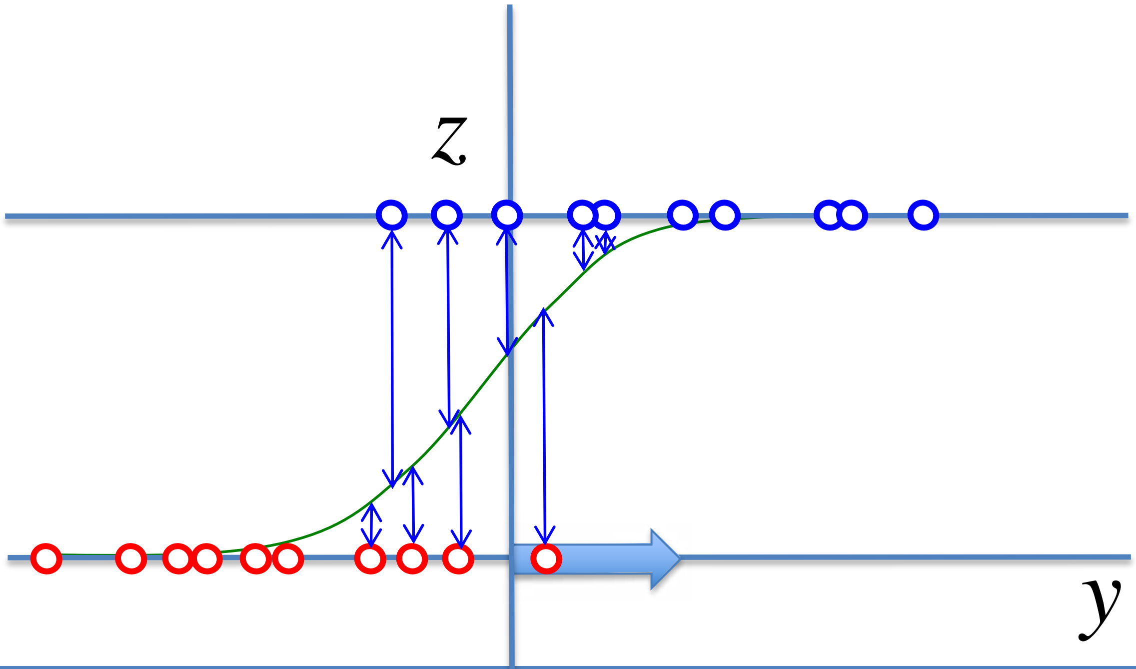


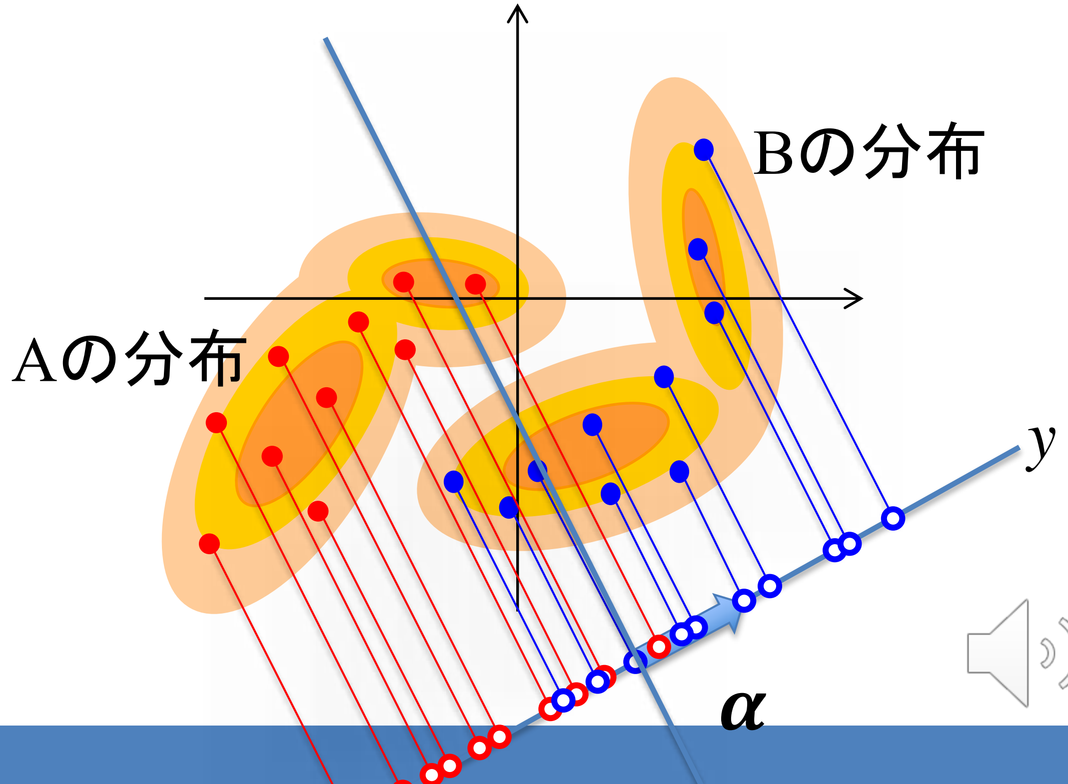
線形識別の場合

$$y_i = \boldsymbol{\alpha}^T \cdot \boldsymbol{x}_i \qquad \|\boldsymbol{\alpha}\|=1$$

$$z_i = \frac{1}{1+\exp(-a\cdot y_i)} = \frac{1}{1+\exp(-\boldsymbol{w}^T\cdot \boldsymbol{x}_i)} \quad \boldsymbol{w} = a\boldsymbol{\alpha}$$

$$\min(z_i - t_i)^2$$



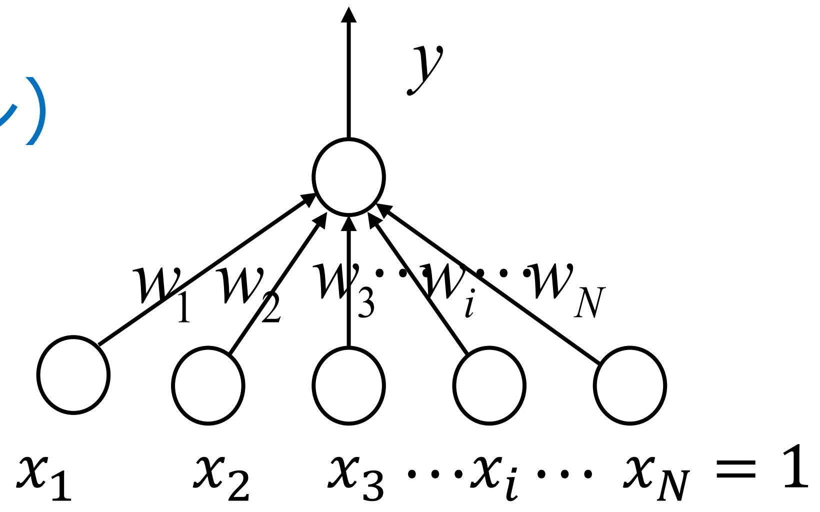


演習問題

2層のニューラルネット（パーセプトロン）を以下のように表現する。

$\boldsymbol{x}^{(k)} = (x_1^{(k)}\ x_2^{(k)} \cdots x_i^{(k)} \cdots x_N^{(k)})^{\mathrm{T}}$:第k入力ベクトル

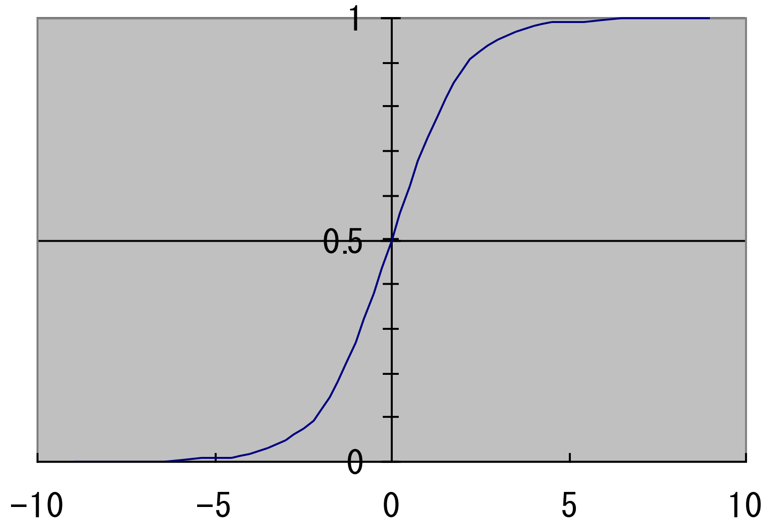
$y^{(k)}$:第k入力に対する出力



$$y^{(k)} = f\left(\sum_{i=1}^{N} w_i x_i^{(k)}\right)$$

$$f(z) = \frac{1}{1+\exp(-z)}$$

（カテゴリAであれば１，そうでなければ，0）



演習問題

$\boldsymbol{w} = (w_1\ w_2 \cdots w_i \cdots\ w_N)^{\mathrm{T}}$

: 重みベクトル（パーセプトロンのパラメータ）

$t^{(k)}$: 第k入力に対する正解カテゴリ（カテゴリAであれば 1 そうでなければ 0）

M : データの個数

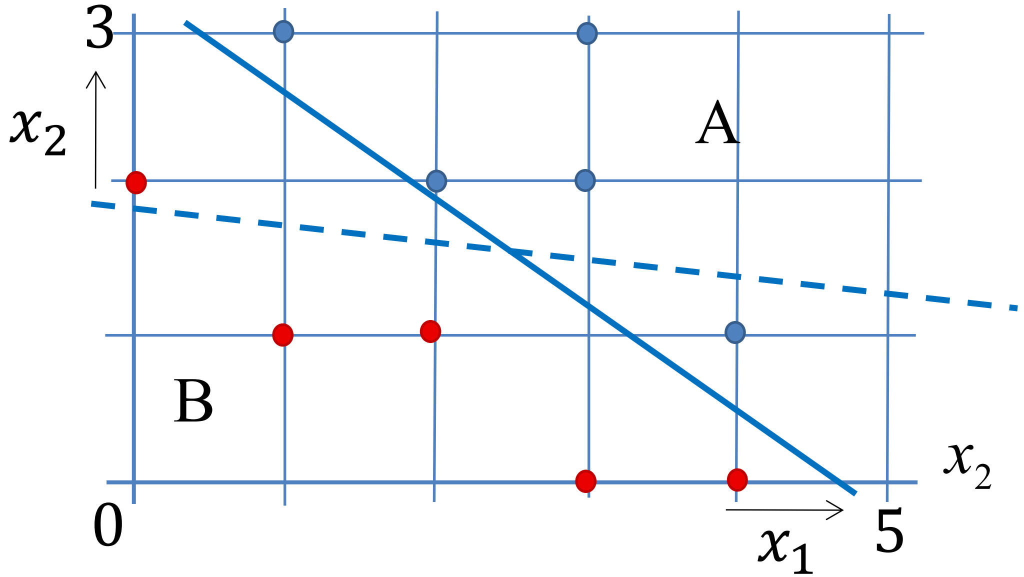
(1) 目的関数を

$$H(\boldsymbol{w}) = \sum_{j=1}^{M} \frac{1}{2}\left(y^{(j)} - t^{(j)}\right)^2$$

$$= \sum_{j=1}^{M} \frac{1}{2}\left(f\left(\sum_{i=1}^{N} w_i x_i^{(j)}\right) - t^{(j)}\right)^2, \quad f(z) = \frac{1}{1+e^{-z}}$$

として定義するとき， $\boldsymbol{w}$ を最急降下法を用いて求めるアルゴリズムを書け。

(2) $\boldsymbol{w}$ の初期値として $\boldsymbol{0}$，学習データとして以下のものが与えられるとき，$\boldsymbol{w}$を求めよ。データの分布図に重ねて，wが決めるカテゴリの境界線を描け。



$x_3 = 1$

$$X = \left(\boldsymbol{x}^{(k)}\right)^T = \begin{pmatrix} 1 & 2 & 3 & 3 & 4 & 0 & 1 & 2 & 3 & 4 \\ 3 & 2 & 2 & 3 & 1 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^T$$

$$T = \left(t^{(k)}\right)^T = (1 \ \ 1 \ \ 1 \ \ 1 \ \ 1 \ \ 0 \ \ 0 \ \ 0 \ \ 0 \ \ 0)^T$$