第三章作业

谈昊　2020E8013282037

Question

一、 在一个 10 类的模式识别问题中，有 3 类单独满足多类情况 1，其余的类别满足多类情况 2。问该模式识别问题所需判别函数的最少数目是多少？

二、 一个三类问题，其判别函数如下：

- * $d_1(x) = -x_1$
  * $d_2(x) = x_1 + x_2 - 1$
  * $d_3(x) = x_1 - x_2 - 1$

(1) 设这些函数是在多类情况 1 条件下确定的，绘出其判别界面和每一个模式类别的区域。

(2) 设为多类情况 2，并使：$d_{12}(x) = d_1(x), d_{13}(x) = d_2(x), d_{23}(x) = d_3(x)$。绘出其判别界面和多类情况 2 的区域。

(3) 设 $d_1(x)$, $d_2(x)$ 和 $d_3(x)$ 是在多类情况 3 的条件下确定的，绘出其判别界面和每类的区域。

三、 两类模式，每类包括 5 个 3 维不同的模式向量，且良好分布。如果它们是线性可分的，问权向量至少需要几个系数分量？假如要建立二次的多项式判别函数，又至少需要几个系数分量？（设模式的良好分布不因模式变化而改变。）

四、 用感知器算法求下列模式分类的解向量 w:

- $\omega_1 : (0,0,0)^T, (1,0,0)^T, (1,0,1)^T, (1,1,0)^T$
- $\omega_2 : (0,0,1)^T, (0,1,1)^T, (0,1,0)^T, (1,1,1)^T$

编写求解上述问题的感知器算法程序。

五、 用多类感知器算法求下列模式的判别函数：

- $\omega_1 : (-1,-1)^T$
- $\omega_2 : (0,0)^T$

- $\omega_3 : (1,1)^T$

六、 编写求解上述问题的感知器算法程序，求下列模式分类的解向量 w:

- $\omega_1 : (0,0,0)^T, (1,0,0)^T, (1,0,1)^T, (1,1,0)^T$
- $\omega_2 : (0,0,1)^T, (0,1,1)^T, (0,1,0)^T, (1,1,1)^T$

尝试不同的初始值尝试不同的迭代顺序

七、 采用梯度法和准则函数

$$J(w,x,b) = \frac{1}{8\|x\|^2}\left[\left(w^T x - b\right) - \left|w^T x - b\right|\right]^2$$

式中实数 b>0，试导出两类模式的分类算法。

八、 用二次埃尔米特多项式的势函数算法求解以下模式的分类问题

- $\omega_1 : (0,1)^T, (0,-1)^T$
- $\omega_2 : (1,0)^T, (-1,0)^T$

九、 用下列势函数

$$K\left(x, x_k\right) = e^{-\alpha\|x-x_k\|^2}$$

求解以下模式的分类问题

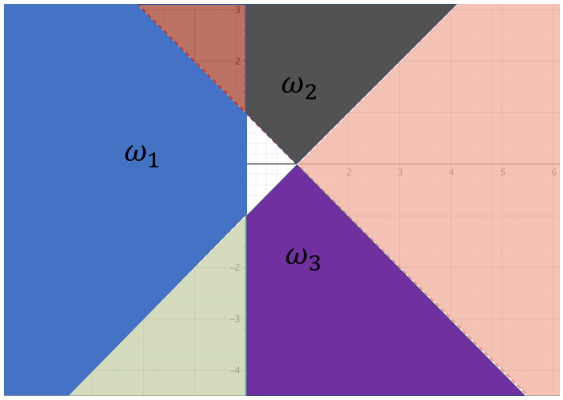
- $\omega_1 : (0,1)^T, (0,-1)^T$
- $\omega_2 : (1,0)^T, (-1,0)^T$

编写求解上述问题的感知器算法程序

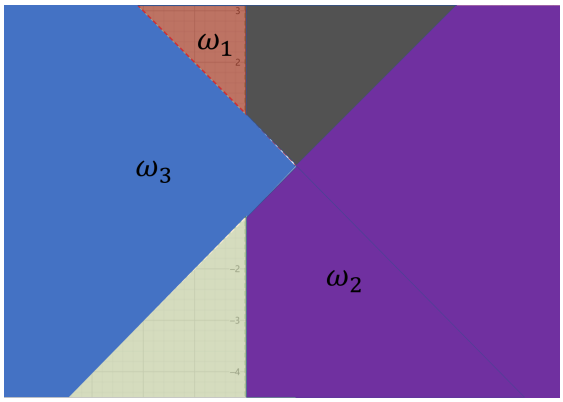
Answer

一、 多类情况 1：M 类需要 M 个判别函数，因此需要 3 个判别函数
多类情况 2：M 类需要 $M(M-1)/2$, 因此需要 $7*(7-1)/2 = 21$ 个判别函数
因此共需要 $3+21 = 24$ 个判别函数

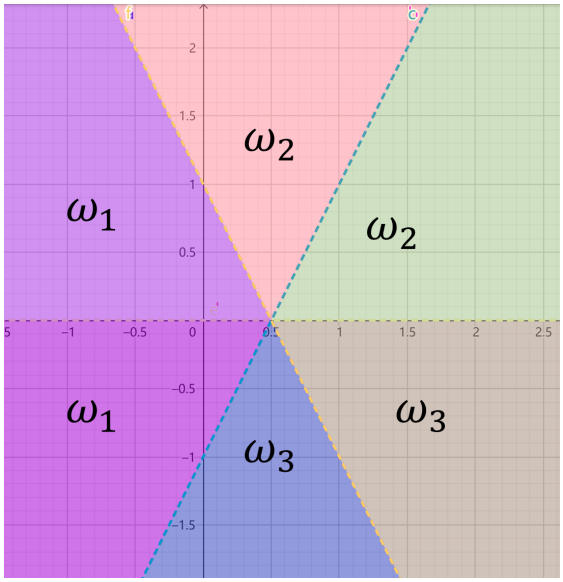
二、 (1) 如下图：



(2) 如下图：



(3) 如下图：



三、 根据公式

$$N_w = C_{n+r}^r = \frac{(n+r)!}{r!n!}$$

因为线性可分，因此仅需 3 个权向量系数；若不可分，则需要 $\frac{5!}{3!2!} = 10$ 个权向量系数

四、 – 解向量 w 为：$(4, -3, -2)$

– 代码：

```
import numpy as np

if __name__ == '__main__':
    w_1 = np.array([[0, 0, 0], [1, 0, 0], [1, 0,
        1], [1, 1, 0]])
    w_2 = np.array([[0, 0, 1], [0, 1, 1], [0, 1,
        0], [1, 1, 1]])
    c = 1
    w_2 *= -1
    w_all = np.vstack((w_1, w_2))
    w = np.array([0, 0, 0])
    flag = [-1] * (w_all.shape[0])
    i = 0
    while -1 in flag:
        i %= len(flag)
```

```
                    x = w_all.data.obj[i]
                    res = np.dot(x, w)
                    if res > 0:
                        flag[i] = 0
                    else:
                        w += x
                        flag[i] = -1
                    i += 1
                print(w)
```

五、 – 判别函数：$d_1(x) = -2x_1 - 2x_2 - 2x_3, d_2(x) = 0, d_3(x) = 2x_1 + 2x_2 - 2x_3$

– 代码为:

```
import numpy as np


def mat_max_index(x, w):
    res = [np.dot(x, value) for value in w.data.obj]
    max_item = max(res)
    max_count = res.count(max_item)
    max_index = res.index(max(res)) if max_count == 1 else
        -1
    return max_index


if __name__ == '__main__':
    O = np.array([[-1, -1], [0, 0], [1, 1]])
    O = np.insert(O, O.shape[1], values=1, axis=1)

    W = np.array([[0, 0, 0], [0, 0, 0], [0, 0, 0]])

    flag = [-1] * O.shape[0]
    i = 0
    while -1 in flag:
        flag = [-1] * O.shape[0]
        for i, o in enumerate(O):
            if i == mat_max_index(o, W):
```

```
                    flag[i] = 0
                else:
                    for j, w in enumerate(W):
                        if j == i:
                            w += o
                        else:
                            w -= o
    print(W)
```

六、 无变化，同四

七、 对准则函数求导可得

$$\frac{\partial J}{\partial w} = \frac{1}{4\|x\|^2}\left[\left(w^t x - b\right) - \left|w^t x - b\right|\right] * \left[x - x * \mathrm{sgn}\left(w^t x - b\right)\right]$$

其中，

$$\mathrm{sgn}\left(w^t x - b\right) = \begin{cases} 1, w^t x - b > 0 \\ -1, w^t x - b \leq 0 \end{cases}$$

得迭代式：

$$w(k+1) = w(k) + \frac{C}{4\|x\|^2}\left[\left(w(k)^t x - b\right) - \left|w(k)^t x - b\right|\right] * \left[x - x * \mathrm{sgn}\left(w(k)^t x - b\right)\right]$$

故得

$$w(k+1) = w(k) + C\begin{cases} 0 & w'x - b > 0 \\ \frac{\left(b - w^t x\right)}{\|x\|^2} x & w^t x - b \leq 0 \end{cases}$$

八、

- 判别函数：$K(x) = 4 * x_2 + 1, 1 - 4 * x_1$
- 代码：

```
import numpy as np
import sympy

if __name__ == '__main__':
    w_1 = np.array([[0, 1], [0, -1]])
    w_2 = np.array([[1, 0], [-1, 0]])

    w_all = np.vstack((w_1, w_2))
    flag = [-1] * w_all.shape[0]

```

```
11              x_1, x_2, y_1, y_2 = sympy.symbols('x_1 x_2
                    y_1 y_2')
12              K = 1 + 4 * x_1 * y_1 + 4 * x_2 * y_2 + 16 *
                    x_1 * x_2 * y_1 * y_2
13              i = 0
14              flag_first = True
15              while -1 in flag:
16                  i %= len(flag)
17                  if i > w_1.shape[0]-1:
18                      isW_1 = -1
19                  else:
20                      isW_1 = 1
21                  w = w_all.data.obj[i]
22                  if flag_first:
23                      K_ = K.subs(y_1, w.data.obj[0])
24                      K_ = K_.subs(y_2, w.data.obj[1])
25                      K_res = K_
26                      flag_first = False
27                  else:
28                      K_res = K_.subs(x_1, w.data.obj[0]).
                            subs(x_2, w.data.obj[1])
29                  if not K_res.is_Add:
30                      if (K_res > 0 and isW_1 == 1) or (
                            K_res < 0 and isW_1 == -1):
31                          flag[i] = 0
32                      else:
33                          K_ += isW_1 * K.subs(y_1, w.data.
                                obj[0]).subs(y_2, w.data.obj
                                [1])
34                          flag[i] = -1
35                  i += 1
36
37                  print(K_)
```

九、 – 判别函数：$K(x) = exp(-x_1^2 - (x_2 - 1)^2)$

– 代码：

```
import numpy as np
import sympy

if __name__ == '__main__':
    w_1 = np.array([[0, 1], [0, -1]])
    w_2 = np.array([[1, 0], [-1, 0]])

    w_all = np.vstack((w_1, w_2))
    flag = [-1] * w_all.shape[0]

    x_1, x_2, y_1, y_2 = sympy.symbols('x_1 x_2 y_1 y_2')
    # K = 1 + 4 * x_1 * y_1 + 4 * x_2 * y_2 + 16 * x_1 *
        x_2 * y_1 * y_2
    K = sympy.exp(-((x_1-y_1)**2+(x_2-y_2)**2))
    i = 0
    flag_first = True
    while -1 in flag:
        i %= len(flag)
        if i > w_1.shape[0]-1:
            isW_1 = -1
        else:
            isW_1 = 1
        w = w_all.data.obj[i]
        if flag_first:
            K_ = K.subs(y_1, w.data.obj[0])
            K_ = K_.subs(y_2, w.data.obj[1])
            K_res = K_
            flag_first = False
        else:
            K_res = K_.subs(x_1, w.data.obj[0]).subs(x_2,
                w.data.obj[1])
        if not K_res.is_Function:
            if (K_res > 0 and isW_1 == 1) or (K_res < 0
                and isW_1 == -1):
                flag[i] = 0
            else:
```

```
34                      K_ += isW_1 * K.subs(y_1, w.data.obj[0]).
                            subs(y_2, w.data.obj[1])
35                      flag[i] = -1
36              i += 1
37
38              print(K_)
```