模式识别导论第二次作业

翁晨阳

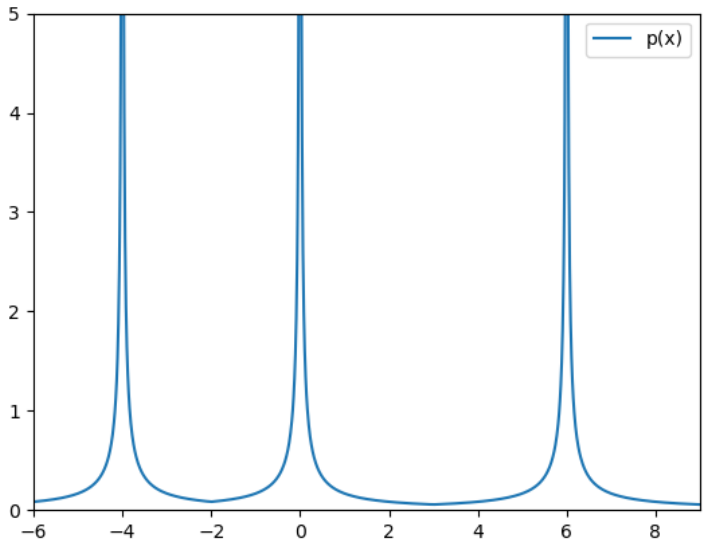
计算题

1. 概论密度函数$p(x)$的Parzen窗估计$p_n(x)$

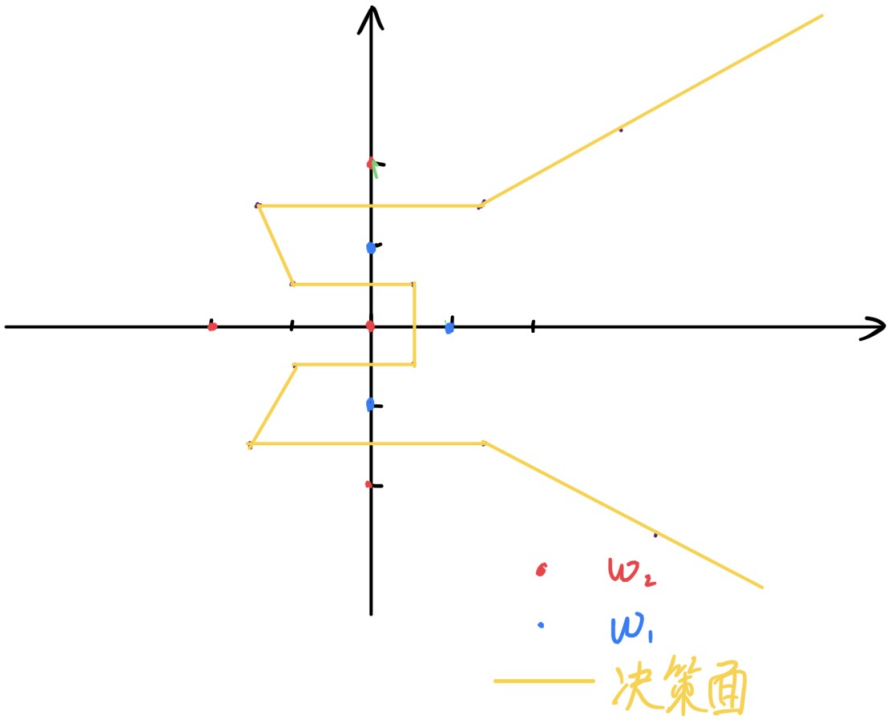
$$
\begin{aligned}
p_n(x) &= \frac{k_n}{nV_n} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\varphi\left(\frac{x-x_i}{h_n}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h_n}\left(\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{\frac{x-x_i}{h_n}}{2}\right)\right) \\
&= \frac{1}{nh_n\sqrt{2\pi}}\sum_{i=1}^{n}\left(\exp\left(-\frac{\frac{x-x_i}{h_n}}{2}\right)\right)
\end{aligned}
$$

2. $p(x)$的最近邻（1-NN）估计

$$
p_n(x) = \frac{k_n}{nV_n} = \begin{cases} \frac{1}{6|x+4|}, & \text{if } x < -2 \\ \frac{1}{6|x|}, & \text{if } -2 < x < 3 \\ \frac{1}{6|x-6|}, & \text{if } x > 3 \end{cases}
$$



3. 最近邻决策面



4. K近邻分类器的优点和缺点

1. **优点：**
    1. 理论成熟，思想简单，既可以用来做分类又可以做回归
    2. 可以用于非线性分类
    3. 训练时间复杂度比支持向量机之类的算法低
    4. 和朴素贝叶斯之类的算法比，对数据没有假设，准确度高，对异常点不敏感
    5. 由于k近邻方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属的类别，因此对于类域的交叉或重叠较多的待分类样本集来说，k近邻方法较其他方法更为适合
    6. 该算法比较适用于样本容量比较大的类域的自动分类，而那些样本容量比较小的类域采用这种算法比较容易产生误分类情况
2. **缺点：**
    1. 计算量大，尤其是特征数非常多的时候
    2. 样本不平衡的时候，对稀有类别的预测准确率低
    3. 相比决策树模型，k近邻模型的可解释性不强

5. 利用批处理感知准则函数方法求解线性判别函数的权向量a

规范化增广样本：

$$\mathbf{y}_1^1 = (1,4,1)^T, \mathbf{y}_1^2 = (2,3,1)^T$$
$$\mathbf{y}_2^1 = (-4,-1,-1)^T, \mathbf{y}_2^2 = (-3,-2,-1)^T$$

计算1：

$$g_1^1(x)=\mathbf{a}^T\mathbf{y}_1^1=(0,1,0)\begin{pmatrix}1\\4\\1\end{pmatrix}=4>0$$

$$g_1^2(x)=\mathbf{a}^T\mathbf{y}_1^2=(0,1,0)\begin{pmatrix}2\\3\\1\end{pmatrix}=3>0$$

$$g_2^1(x)=\mathbf{a}^T\mathbf{y}_2^1=(0,1,0)\begin{pmatrix}-4\\-1\\-1\end{pmatrix}=-1<0$$

$$g_2^2(x)=\mathbf{a}^T\mathbf{y}_2^2=(0,1,0)\begin{pmatrix}-3\\-2\\-1\end{pmatrix}=-2<0$$

梯度下降1：

$$\mathbf{a}=\mathbf{a}+\eta_k\sum_{\mathbf{y}\in Y_k}\mathbf{y}=(0,1,0)^T+(-4,-1,-1)^T+(-3,-2,-1)^T=(-7,-2,-2)^T$$

计算2：

$$g_1^1(x)\quad=\mathbf{a}^T\mathbf{y}_1^1=(-7,-2,-2)\begin{pmatrix}1\\4\\1\end{pmatrix}=-17<0$$

$$g_1^2(x)\quad=\mathbf{a}^T\mathbf{y}_1^2=(-7,-2,-2)\begin{pmatrix}2\\3\\1\end{pmatrix}=-22<0$$

$$g_2^1(x)\quad=\mathbf{a}^T\mathbf{y}_2^1=(-7,-2,-2)\begin{pmatrix}-4\\-1\\-1\end{pmatrix}=32>0$$

$$g_2^2(x)\quad=\mathbf{a}^T\mathbf{y}_2^2=(-7,-2,-2)\begin{pmatrix}-3\\-2\\-1\end{pmatrix}=27>0$$

梯度下降2：

$$\mathbf{a}=\mathbf{a}+\eta_k\sum_{\mathbf{y}\in Y_k}\mathbf{y}=(-7,-2,-2)^T+(1,4,1)^T+(2,3,1)^T=(-4,5,0)^T$$

计算3：

$$g_1^1(x)\quad=\mathbf{a}^T\mathbf{y}_1^1=(-4,5,0)\begin{pmatrix}1\\4\\1\end{pmatrix}=16>0$$

$$g_1^2(x)\quad=\mathbf{a}^T\mathbf{y}_1^2=(-4,5,0)\begin{pmatrix}2\\3\\1\end{pmatrix}=7>0$$

$$g_2^1(x)\quad=\mathbf{a}^T\mathbf{y}_2^1=(-4,5,0)\begin{pmatrix}-4\\-1\\-1\end{pmatrix}=1>0$$

$$g_2^2(x)\quad=\mathbf{a}^T\mathbf{y}_2^2=(-4,5,0)\begin{pmatrix}-3\\-2\\-1\end{pmatrix}=2>0$$

所有样本都分正确，此时权向量为$\mathbf{a}=(-4,5,0)^T$

编程题

1. Parzen算法

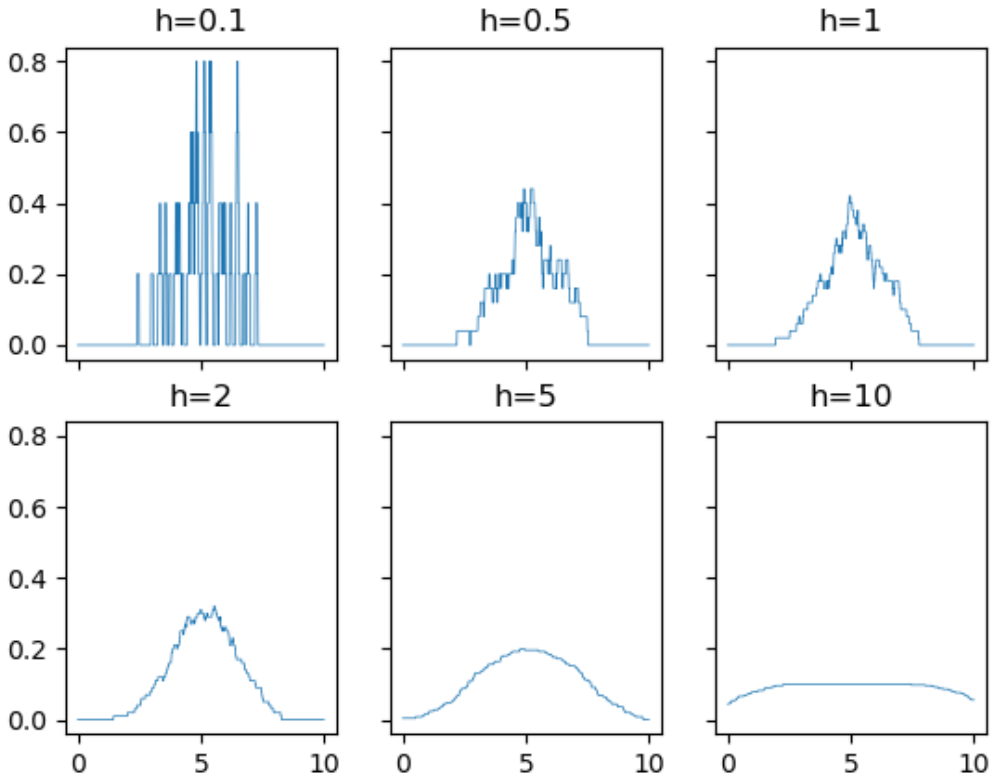
方窗：

```
#生成样本点和采样间隔
sampleNo = 50
mu = 5
std_var = 1
np.random.seed(0)
s = np.random.normal(mu, std_var, sampleNo )
x = np.linspace(0,10,1000)

#方窗
def Kernel(x, xi, h):
    delta_x = abs(x-xi)
    delta_x[delta_x<=h/2]=0
    delta_x[delta_x>=h/2]=2
    delta_x[delta_x==0]=1
    delta_x[delta_x==2]=0
    return delta_x

#p(x)方窗
def get_px(x, y, h):
    n = len(y)
    K = 0
    for i in range(n):
        K += Kernel(x, y[i], h)
    print(K.shape)
    px = 1/sampleNo*K/h
    return px

get_px(x, s, 0.5)
```



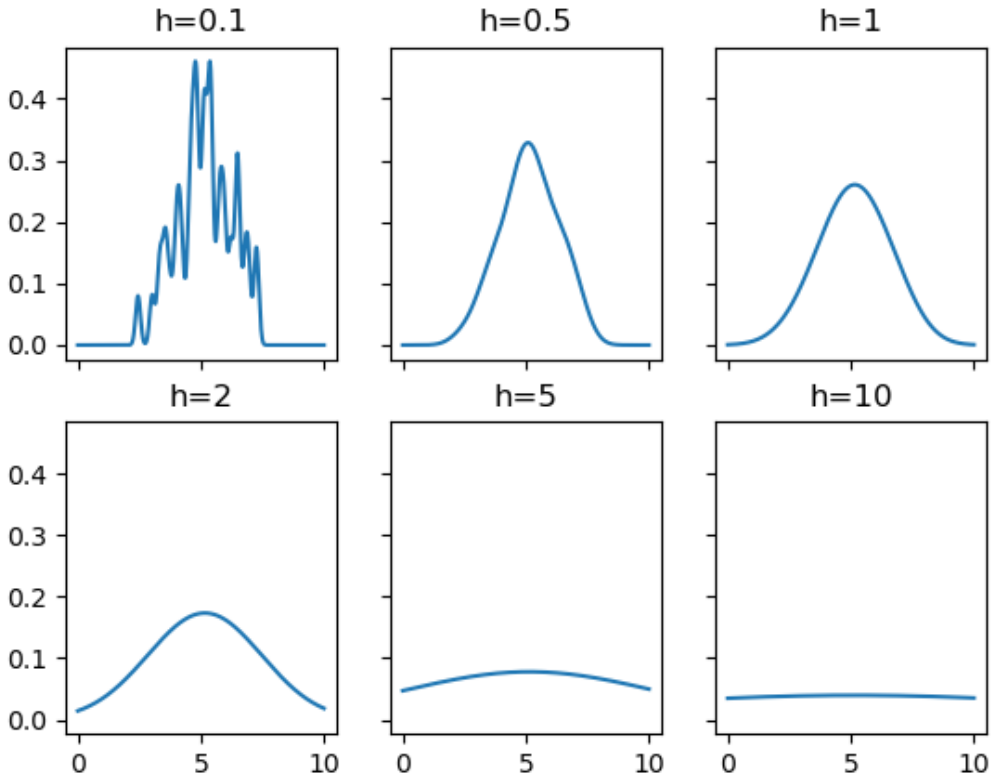
高斯窗:

```python
#生成样本点和采样间隔
sampleNo = 50;
mu = 5
std_var = 1
np.random.seed(0)
s = np.random.normal(mu, std_var, sampleNo )
x = np.linspace(0,10,1000)

#高斯
def get_phi_Gauss(u):
    phi = np.exp(-u*u.T/2)
    return phi

#p(x)高斯
def get_px_Gauss(x, y, h):
    phi = 0
    n = len(y)
    for i in range(n):
        u = (x-y[i])/h
        phi += get_phi_Gauss(u)
    px = 1/sampleNo/h*phi/math.sqrt(2*np.pi)
    return px

get_px_Gauss(x, s, 0.5)
```



窗宽的影响：

窗越宽会导致得到的$p(x)$越平滑数值越平均，窗越窄会导致得到的$p(x)$波动越大，当窗过窄的时候可能会导致采样缺失，尤其是方窗的时候。

2. 感知准则函数算法&MSE多类扩展算法

1. **感知准则函数算法**

```
#规范化增广样本
def samples_trans(w,key):
    # 复制样本,防止后续操作改变原始样本
    ww = copy.deepcopy(w)

    for i in ww: i.append(1)
    ww = np.array(ww)
    if key == 1:
        ww = -ww
    return ww

#计算权重a
def batch_perception(w1, w2, eta, theta):
    w1 = samples_trans(w1,0)
    w2 = samples_trans(w2,1)
    w = np.concatenate([w1,w2])
    a = np.zeros_like(w[1])

    count = 0
    while True:
        y = np.zeros_like(w[1])
        for i in w:
            if a.T.dot(i).T<=theta:
                y+=i
```

```
        #输出每一次迭代的解向量a和sum(y)
        print(count,'\t',a,'\t',y)
        if all(y == 0)or count>2000:
            break

        a += eta*y
        count += 1


batch_perception(w_2,w_3,1,0.01)
```

输出：

```
0    [0. 0. 0.]          [ 55.2 -43.7   0. ]
1    [ 55.2 -43.7   0. ]    [-2.9 -2.1 -1. ]
2    [ 52.3 -45.8  -1. ]    [-2.9 -2.1 -1. ]
3    [ 49.4 -47.9  -2. ]    [-2.9 -2.1 -1. ]
4    [ 46.5 -50.   -3. ]    [ 0.1  0.8 -2. ]
5    [ 46.6 -49.2  -5. ]    [-2.9 -2.1 -1. ]
6    [ 43.7 -51.3  -6. ]    [ 0.1  0.8 -2. ]
7    [ 43.8 -50.5  -8. ]    [ 0.1  0.8 -2. ]
8    [ 43.9 -49.7 -10. ]    [ 0.1  0.8 -2. ]
9    [ 44.  -48.9 -12. ]    [-2.9 -2.1 -1. ]
10    [ 41.1 -51.  -13. ]    [ 3.   2.9 -1. ]
11    [ 44.1 -48.1 -14. ]    [-2.9 -2.1 -1. ]
12    [ 41.2 -50.2 -15. ]    [ 3.   2.9 -1. ]
13    [ 44.2 -47.3 -16. ]    [-2.9 -2.1 -1. ]
14    [ 41.3 -49.4 -17. ]    [ 3.   2.9 -1. ]
15    [ 44.3 -46.5 -18. ]    [-2.9 -2.1 -1. ]
16    [ 41.4 -48.6 -19. ]    [0. 0. 0.]
```

问题：当 $\theta = 0$，初始化解向量 $a = 0$ 时，无论 η 改变迭代次数将不会改变，最终得到的解向量 a 将成比例变化，此时 η 的大小改变没有意义。

2. **MSE多类扩展算法**

```
def MSE_multi(wi):
    w_i = copy.deepcopy(wi)

    #回归值矩阵y,增广X矩阵
    y = np.zeros((len(w_i), len(w_i)*len(w_i[0])))
    X = []
    for idx, i in enumerate(w_i):
        for j in i:
            j.append(1)
            X.append(j)
        y[idx, idx*len(w_i[0]):(idx+1)*len(w_i[0])] = 1

    X = np.array(X).T
    il = X.shape[0]
    lad = 1e-10
    # 计算w
    W = np.matmul(np.matmul(np.linalg.inv(np.matmul(X,
X.T))+lad*np.identity(il), X), y.T)

    return W
```

```
20
21  #测试
22  def test(w_test, W):
23
24      w_t = copy.deepcopy(w_test)
25      f_cnt = 0
26      for idx_i, i in enumerate(w_t):
27          for idx_j, j in enumerate(i):
28              j.append(1)
29              j = np.array(j)
30              #决策
31              if np.argmax(np.matmul(W.T, j)) != idx_i: f_cnt += 1
32
33      f_ratio = f_cnt / ((idx_i+1)*(idx_j+1))
34
35      return 1-f_ratio
```

输出：

```
[[ 0.02049668  0.06810971 -0.04087307 -0.04773333]
 [ 0.01626151 -0.03603827  0.05969134 -0.03991458]
 [ 0.26747287  0.27372075  0.25027714  0.20852924]]
1.0
```