机器学习实验报告 (三)

2018/11/25

目录

1 实验环境

操作系统	win10
编程语言	python3
编程环境	Jupyter Notebook
报告编写	latex

2 题目描述与分析

2.1 概率密度的估计

在看具体问题之前，我们先来看一下概率密度的估计。我们知道一个向量 **x** 落在区域 R 中的概率

$$P = \int_R p(x')\,dx' \tag{1}$$

假设 n 个样本 $\boldsymbol{x_1} \ldots \boldsymbol{x_n}$ 都是根据概率密度函数 $p(\boldsymbol{x})$ 独立同分布的抽取而得到的，则 k 个样本落在区域 R 中的概率服从二项式定理：

$$P_k = \binom{n}{k} P^k (1-P)^{n-k} \tag{2}$$

容易得到 k 的期望值为

$$\varepsilon[k] = nP \tag{3}$$

当 n 较大时，比值 k/n 将会是 P 的一个很好的估计。假设 $p(\boldsymbol{x})$ 是连续的，则当区域 R 足够小时，我们可以用 R 所包含的体积 V 与 $p(\boldsymbol{x})$ 的乘积作为该点的概率，即：

$$P = p(\boldsymbol{x})\,V \tag{4}$$

将 $P = k/n$ 代入 (4) 可得到 $p(\boldsymbol{x})$ 的估计为

$$p(\boldsymbol{x}) \approx \frac{k/n}{V} \tag{5}$$

且在 V 相对较小的情况下，样本个数 n 越大时，得到的估计越准确

2.2 第一题——Parzen 窗

题目描述: 对所给数据进行 *Parzen* 窗估计和设计分类器

暂时假设区间 R_n 是一个 d 维的超立方体，h_n 为超立方体的一条边的长度，则体积为

$$V_n = h_n^d \tag{6}$$

定义窗函数为:

$$\varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h_n}\right) = \begin{cases} 1 & |\boldsymbol{x}-\boldsymbol{x}'| \leq h_n/2 \\ 0 & other \end{cases} \tag{7}$$

即当 $\boldsymbol{x}'$ 落在以 $\boldsymbol{x}$ 为中心，边长为 h_n 的超立方体中时 $\varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{h_n}\right) = 1$ 否则便为 0. 于是便可得到超立方体中的样本个数

$$k_n = \sum_{i=1}^{n} \varphi_1\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right), \tag{8}$$

联立 (5) 式，可推得：

$$p_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{V_n}\varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right) \tag{9}$$

更一般的，我们可以把上式理解为：对样本集中每一个样本依据它离 $\mathbf{x}$ 的远近不同所做的贡献作平均，因此区间的选取并不一定为超立方体，只要保证有窗函数满足 $\int \varphi(\boldsymbol{x})\, d\boldsymbol{x} \geq 0$ 且

$$\int \varphi(\boldsymbol{u})\, d\boldsymbol{u} = 1 \tag{10}$$

即可保证 $p_n(\boldsymbol{x})$ 是一个合理的概率密度函数。在此题中，窗函数为一个球形的高斯函数，如下所示：

$$\varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right) \propto exp[\frac{-(\mathbf{x}-\boldsymbol{x}_i)^t(\mathbf{x}-\boldsymbol{x}_i)}{2h^2}]$$

为了满足上式，经过简单计算，可以得到窗函数的具体表达：

$$\varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right) = exp[\frac{-(\mathbf{x}-\boldsymbol{x}_i)^t(\mathbf{x}-\boldsymbol{x}_i)}{2h^2}]/\sqrt{2\pi} \tag{11}$$

观察 (9)(11) 两式即可得到最终表达式

$$p_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}exp[\frac{-(\mathbf{x}-\boldsymbol{x}_i)^t(\mathbf{x}-\boldsymbol{x}_i)}{2h^2}]/\sqrt{2\pi} \tag{12}$$

2.3 第二题——k-近邻

问题描述: 在不同维数的空间中，使用 k 近邻方法对概率密度进行估计

不同于 parzen 窗方法，k-近邻不用选择窗函数，而是让体积成为训练样本的函数。如，为了从 n 个训练样本中估计 $p_n(\boldsymbol{x})$，我们能够以点 $\mathbf{x}$ 为中心，让体积扩张，直到包含进 k_n 个样本为止，其中 k_n 为关于 n 的某个特定函数。令

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n} \tag{13}$$

很容易看出，当某个区域较密集时，很小的 V 即可满足包含 k 个样本的要求，此时对应的概率密度 $p_n(\boldsymbol{x})$ 就很大；相反，当某个区域比较稀

疏，满足包含 k 个样本的要求所需要的 V 相对较大，此时对应的概率密度 $p_n(\boldsymbol{x})$ 就相对很小。

根据不同的维数，以不同的方式计算体积，代入公式即可算出某点的概率密度

3 具体实现

3.1 库的导入及数据准备

```
import numpy as np #矩阵运算
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

lists1=np.array([[0.28,1.31,-6.2],
                 [0.07,0.58,-0.78],
                 [1.54,2.01,-1.63],
                 [-0.44,1.18,-4.32],
                 [-0.81,0.21,5.73],
                 [1.52,3.16,2.77],
                 [2.20,2.42,-0.19],
                 [0.91,1.94,6.21],
                 [0.65,1.93,4.38],
                 [-0.26,0.82,-0.96]])

lists2=np.array([[0.011,1.03,-0.21],
                 [1.27,1.28,0.08],
                 [0.13,3.12,0.16],
```

```
                    [-0.21,1.23,-0.11],
                    [-2.18,1.39,-0.19],
                    [0.34,1.96,-0.16],
                    [-1.38,0.94,0.45],
                    [-0.12,0.82,0.17],
                    [-1.44,2.31,0.14],
                    [0.26,1.94,0.08]])

lists3=np.array([[1.36,2.17,0.14],
                    [1.41,1.45,-0.38],
                    [1.22,0.99,0.69],
                    [2.46,2.19,1.31],
                    [0.68,0.79,0.87],
                    [2.51,3.22,1.35],
                    [0.60,2.44,0.92],
                    [0.64,0.13,0.97],
                    [0.85,0.58,0.99],
                    [0.66,0.51,0.88]])
list_point = [np.array([0.5,1.0,0.0]),np.array([0.31,1.51,-0.50]),
               np.array([-0.3,0.44,-0.1])]
list_group = [lists1,lists2,lists3]
list_point2 = [np.array([-0.41,0.82,0.88]),np.array([0.14,0.72,4.1]),
               np.array([-0.81,0.61,-0.38])]
```

3.2 Parzen 窗

求解第一题，我们将分类样本点与训练样本集带入公式 (12) 即可。首先，我们定义了一个 $getpxitem(point_aim, point_item, h)$ 函数, 用于窗函数的计算。有三个参数，其中 $point_aim$ 代表测试点的坐标，$point_item$ 代表样本

点的坐标，h 为我们传进去的参数。具体代码如下:

```
def getpxitem(point_aim,point_item,h):
    temp = point_aim-point_item
    mici = np.dot(temp,temp.T)/(2*pow(h, 2))
    return np.exp(-mici)/((2*np.pi)**0.5)
```

然后我们定义 $getpx(points, point, h)$ 函数求解在训练集为 ω_i 的某一点 x 处的概率密度 $p(x)$，其中 $points$ 为训练样本集，$point$ 为我们要求的样本点。调用了我们实现好的 $getpxitem$ 函数。具体代码如下:

```
def getpx(points,point,h):
    point = np.matrix(point)
    n = len(points)
    m_points = np.matrix(points)#10x3
    sigma = 0
    for i in m_points:
        sigma = sigma + getpxitem(point,i,h)
    return float(sigma/(n*h*h*h))
```

随后编写调用函数:

```
def descript(list_group,list_point,h):
    for p in list_point:
        print("测试点",p)
        pos = 1
        i = 0
        maxpx = 0
        for points in list_group:
            i = i+1
            temp = getpx(points,p,h)
            if(temp>maxpx):
```

```
                maxpx = temp
                pos = i
            print('在样本集w'+str(i)+"下的概率密度为",temp)
        print("因此我们将",p,"归为w"+str(pos))
        print("---------------------------------")
```

对 (a) 小问进行求解：

```
descript(list_group,list_point,1)
```

所得结果如下所示:

测试点 $(0.5, 1.0, 0.0)^t$
在样本集 *w1* 下的概率密度为 *0.050234405015983905*
在样本集 *w2* 下的概率密度为 *0.18793313079292684*
在样本集 *w3* 下的概率密度为 *0.15878419484781175*
因此我们将 $(0.5, 1.0, 0.0)^t$ 归为 *w2*

测试点 $(0.31, 1.51, -0.5)^t$
在样本集 *w1* 下的概率密度为 *0.061213126923756335*
在样本集 *w2* 下的概率密度为 *0.19260786786365375*
在样本集 *w3* 下的概率密度为 *0.0901699142163708*
因此我们将 $(0.31, 1.51, -0.5)^t$ 归为 *w2*

测试点 $(-0.3, 0.44, -0.1)^t$
在样本集 *w1* 下的概率密度为 *0.0558152640204981*
在样本集 *w2* 下的概率密度为 *0.15090164416412882*
在样本集 *w3* 下的概率密度为 *0.072731855396044 24*
因此我们将 $(-0.3, 0.44, -0.1)^t$ 归为 *w2*

对 (b) 小问进行求解：

```
descript(list_group,list_point,0.1)
```

所得结果如下所示:

测试点 $(0.5, 1.0, 0.0)^t$
在样本集 *w1* 下的概率密度为 *3.5005969554957954e-20*
在样本集 *w2* 下的概率密度为 *2.700448085924117e-05*
在样本集 *w3* 下的概率密度为 *3.1916135837019276e-17*
因此我们将 $(0.5, 1.0, 0.0)^t$ 归为 *w2*

测试点 $(0.31, 1.51, -0.5)^t$
在样本集 *w1* 下的概率密度为 *1.1454097051470594e-20*
在样本集 *w2* 下的概率密度为 *4.788040708256211e-06*
在样本集 *w3* 下的概率密度为 *8.614476229511478e-26*
因此我们将 $(0.31, 1.51, -0.5)^t$ 归为 *w2*

测试点 $(-0.3, 0.44, -0.1)^t$
在样本集 *w1* 下的概率密度为 *1.45100204839023e-12*
在样本集 *w2* 下的概率密度为 *0.0001509236162806271*
在样本集 *w3* 下的概率密度为 *4.2448498919421 4e-40*
因此我们将 $(-0.3, 0.44, -0.1)^t$ 归为 *w2*

3.3 k-近邻

我们由公式 (11) 易知，在给定 k 与 n 的情况下，我们只需求出相应的 V 即可，在这点与点的距离我们采用欧氏距离，于是我们首先定义方法 $distance(vecA, vecB)$，两个参数分别为两个点的表示。具体代码如下:

```
# 计算欧几里得距离
def distance(vecA, vecB):
```

```
    return np.sqrt(np.sum(np.power(vecA - vecB, 2))) # 求两个向量之间的
    return D
```

然后我们定义方法 $getNearest(point, points, k)$ 来求解包含 k 个点时的最短半径，$point$ 为中心，$points$ 为样本集合，k 为所包含点的个数。具体代码如下:

```
#返回包含k个点的最短半径
def getNearest(point , points , k):
    m_dis={}
    pos =0
    for i in points:
        pos = pos+1
        m_dis[pos] = distance(point,i)
    sortlist=sorted(m_dis.items(),key = lambda x:x[1],reverse = False)
    return sortlist[k-1][1]
```

我们再定义函数 $getY(dis, dim, k)$ 来计算不同维数下 $p(\boldsymbol{x})$ 的取值。有三个参数，dis 为我们求出的最短半径，dim 代表维数，我们这里取值 1,2,3；k 为所包含点的个数。在计算体积时，一维我们选择以 x 为中心的长度为 $2dis$ 的线段；二维时我们选择以 x 为中心的半径为 dis 的圆；三维时我们选择以 x 为中心的半径为 dis 的球。具体代码如下:

```
#根据不同维数算px
def getY(dis,dim,k):
    total = 10
    if dim == 1:
        result = (k/total)/(2*dis)
    elif dim == 2:
        result = (k/total)/(np.pi*dis*dis)
    elif dim == 3 :
```

```
        result = (k/total)/((4/3)*(np.pi*dis*dis*dis))
    else:
        result = 0
    return result
```

3.3.1 一维情况

编写代码，绘制一维情况下概率密度估计结果

```
def drawPx1d(k):
    #最大2.51，最小0.60，于是去区间为[0.5,2.6]
    x = list(np.arange(0.5,2.6,0.01))
    y = []
    for i in x:
        dis = getNearest(np.matrix(i),np.matrix(lists3[:,0]).T,k)
        y.append(getY(dis,1,k))
    plt.figure()
    plt.plot(x,y)
```

当 $k=1$ 时，

```
drawPx1d(k=1)
```

得到如下结果

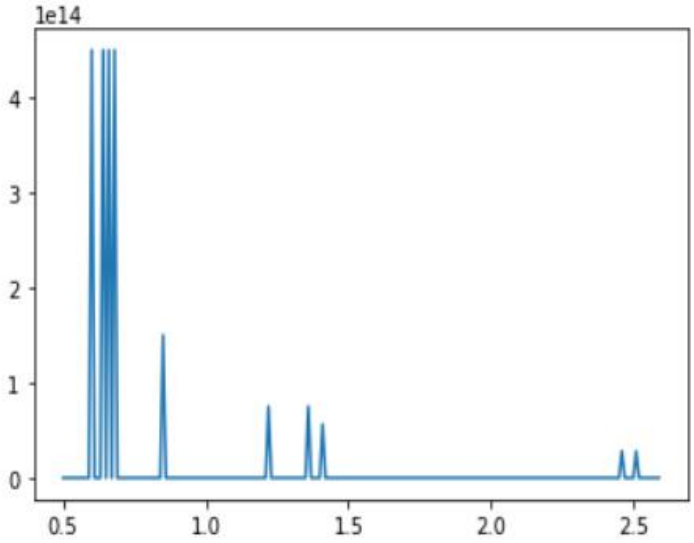

图 1　k=1 时概率密度估计结果

当 $k = 3$ 时，

```
drawPx1d(k=3)
```

得到如下结果

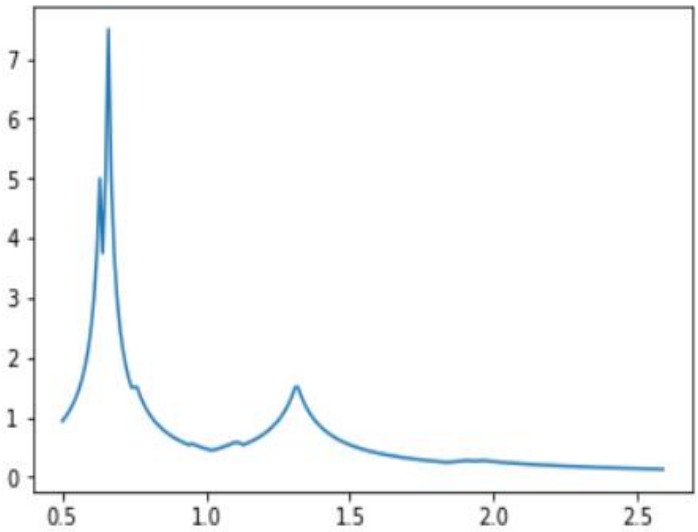

图 2　k=3 时概率密度估计结果

当 $k=5$ 时，

```
drawPx1d(k=5)
```

得到如下结果

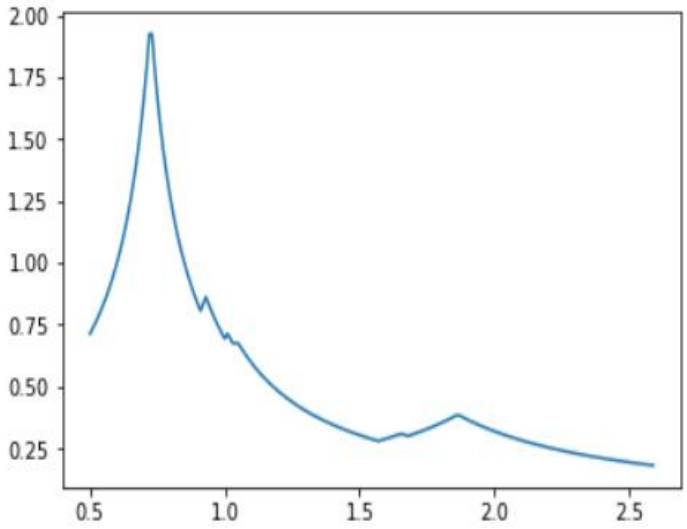

图 3　k=5 时概率密度估计结果

通过对比观察我们可以发现，当 k 较大时，$p_n(\boldsymbol{x})$ 分辨率较低，对个别样本敏感度不高，图像较为平滑；当 k 较小时，$p_n(\boldsymbol{x})$ 较为陡峭，噪声比较大，统计稳定性也不高

3.3.2　二维情况

编写如下程序，求解并绘制概率密度估计结果：

```
def drawPx2d(k):
    #x最小-2.18，最大1.27，于是去区间为[-2.20，1.30]
    #y最小0.82，最大3.12，于是去区间为[0.80，3.15]
    x = list(np.arange(-2.20,1.30,0.03))
    y = list(np.arange(0,3.15,0.03))
    X, Y = np.meshgrid(x, y)
```

```
    z = []
    for j in y:
        temp = []
        for i in x:
            dis = getNearest(np.matrix([i,j]),np.matrix(lists2[:,[0,1]
            temp.append(getY(dis,2,k))
        z.append(temp)
    Z = np.array(z)

    fig = plt.figure()
    ax = Axes3D(fig)
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='rainbow')
    plt.show()
```

当 $k=1$ 时，按如下调用函数

```
drawPx2d(1)
```

得到如下结果

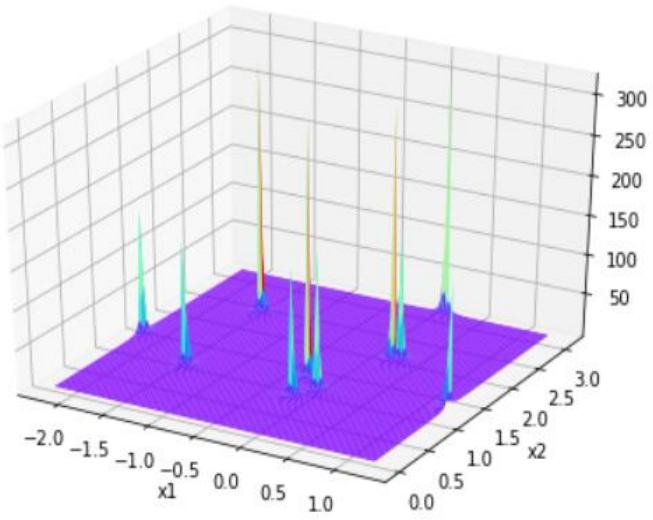

图 4　k=1 时概率密度估计结果

从另一角度来观察：

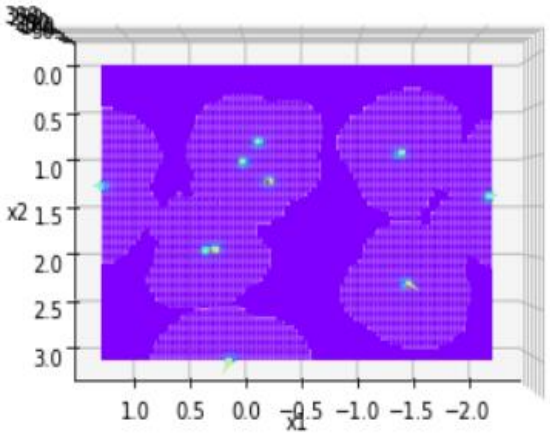

图 5　k=1 时概率密度估计结果

当 $k = 3$ 时，按如下调用函数

```
drawPx2d(3)
```

得到如下结果

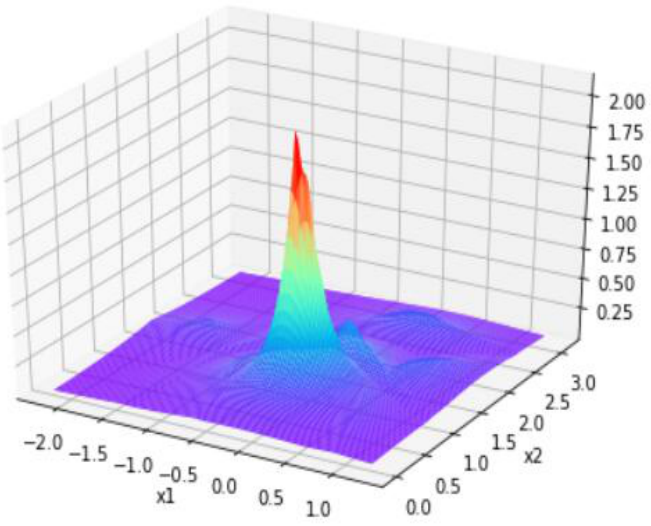

图 6　k=3 时概率密度估计结果

从另一角度来观察：

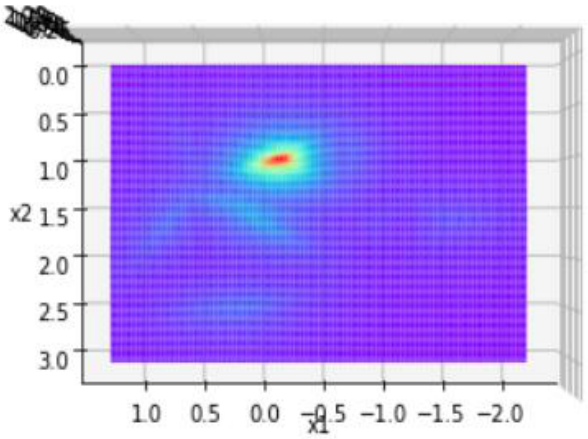

图 7　k=3 时概率密度估计结果

当 $k = 5$ 时，按如下调用函数

```
drawPx2d(5)
```

得到如下结果

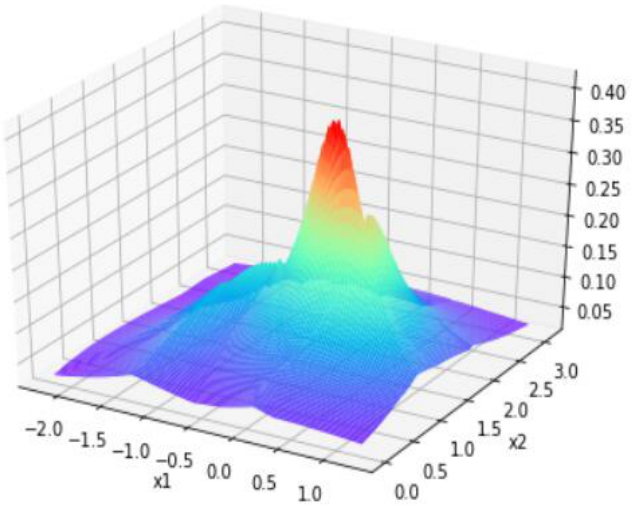

图 8　k=5 时概率密度估计结果

从另一角度来观察：

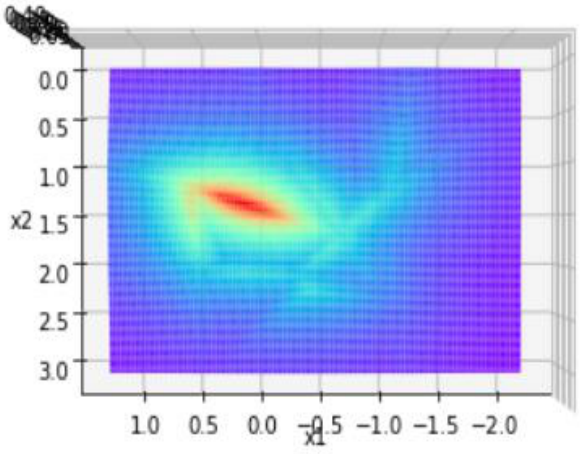

图 9　k=5 时概率密度估计结果

我们发现了与一维情况类似的结果，即 k 越大，图像相对越平滑；反之，图像噪点越多。

3.3.3 三维情况

根据公式编写如下程序：

```
def classification(testPoints,pointList,k):
    x=0
    for i in testPoints:
        print("测试点",i)
        x=x+1
        pos =0
        maxpx = 0
        maxpos = 0
        for j in pointList:
            pos = pos+1
            dis = getNearest(np.matrix(i),np.matrix(j),k)
            temp = getY(dis,3,k)
            if(temp>=maxpx):
                maxpos = pos
                maxpx = temp
            print('在样本集w'+str(pos)+"下的概率密度为",temp)
        print("---------------------------------")
```

所得结果如下所示:

测试点 $(-0.41, 0.82, 0.88)^t$

在样本集 *w1* 下的概率密度为 *0.002614944210445274*

在样本集 *w2* 下的概率密度为 *0.033276646733531694*

在样本集 *w3* 下的概率密度为 *0.026581112078461 73*

测试点 $(0.14, 0.72, 4.1)^t$

在样本集 *w1* 下的概率密度为 *0.0010255387557116637*

在样本集 *w2* 下的概率密度为 *0.00154925332155 8584*

在样本集 *w3* 下的概率密度为 *0.002586099273986083*

测试点 $(-0.81, 0.61, -0.38)^t$

在样本集 *w1* 下的概率密度为 *0.0018676527558116678*

在样本集 *w2* 下的概率密度为 *0.029813898753325307*

在样本集 *w3* 下的概率密度为 *0.0094855259161561*

可以发现，三个样本点分别在 ω_2、ω_3、ω_2 时的概率密度最大

4 总结与收获

通过这次实验，我增强了对“非参数化方法”的认识与理解。Parzen 窗估计方法和 K-近邻概率密度估计方法是其主要的两种技术，前者固定体积求 k 值，后者固定 k 值求体积。非参数化方法优点在于通用性，不必提前去了解假设分布的形式，用同样的方法便可求出结果。在样本足够的情况下，往往可以得到很好的结果。不过也有很大的缺点。相对知道参数的分布形式的情况下，要求的训练样本的个数多很多，同时，随着样本特征空间维数的提高，将会有很大的计算量。