

2.5 ノンパラメトリック法

平成 28 年 9 月 11 日

概要

PRML の「2.5 ノンパラメトリック法」についての実装と考察

目次

1 問題設定	2
2 ヒストグラム密度推定法	2
2.1 アルゴリズム	2
2.2 コード	2
2.3 結果	3
2.4 考察, ヒストグラム密度推定法の改良	3
2.4.1 点を直線で結ぶ	4
2.4.2 回帰	4
2.4.3 区間を限定	5
2.4.4 確率 0 についての工夫	6
3 カーネル密度推定法	7
3.1 アルゴリズム	7
3.2 コード	7
3.3 結果	8
3.3.1 Pazan 窓	8
3.3.2 ガウスカーネル	9
4 K 近傍法	9
4.1 アルゴリズム	9
4.2 コード	10
4.3 結果	10
4.4 考察, K 近傍法の改良	11
4.4.1 $K, K+1$ 個含む V_K, V_{K+1} の中間を用いる	11
5 まとめ	12

1 問題設定

分布のわからないデータについて、パラメータを用いる（例えば、ガウス分布と仮定し平均、分散の最適化を行う）のではなく、分布の形状についてわずかな仮定のみを用い、分布の形状を把握しようというものである。

2 ヒストグラム密度推定法

2.1 アルゴリズム

一次元で説明するが、区間を幅 Δ_i の別々の区間に区切り、 i 番目の区間にに入った x の観測地の数 n_i を数える。この計数を正規化された確率密度にするために、これらの計数を、単に観測地の総数 N と、区間の幅 Δ_i で割る。すると区間の確率密度は

$$p_i = \frac{n_i}{N\delta_i} \quad (2.241)$$

になる。

ここでは、 $\Delta_i = \Delta$ と一定にしている。

2.2 コード

ヒストグラムをプロットする (hist.py)

```
#coding: utf-8
import numpy as np
import numpy.random as rd
import scipy as sp
import matplotlib.pyplot as plt
from math import pi

for N in [100,1000,10000,100000]:
    for M in [10,50,100,500]:
        #データ
        dataset="./x%d.tsv" %N
        x=sp.genfromtxt(dataset,delimiter="\n")

        #区間
        p=np.zeros(M)
        MIN,MAX=np.amin(x),np.amax(x)+0.00001
        delta=(MAX-MIN)/M
        for n in range(N):
            p[int((x[n]-MIN)//delta)]+=1
        p/=(N*delta)

        PI,mu,sig=[0.3,0.7],[3,6],[0.5,1]
        def func(z):
            sum=0.0
            for i in [0,1]:
                sum+=PI[i]/np.sqrt(2*pi*(sig[i]**2))*\
                    np.exp(-((z-mu[i])**2)/(2*(sig[i]**2)))
            return sum
        fx=sp.linspace(0,10,100)
        plt.xlim((0,10))
        plt.ylim((0,0.5))
        plt.hist(x,bins=M,histtype="step",range=(0,10),normed=True)
        plt.plot(fx,func(fx),'r-',lw=3,alpha=0.7)
```

```

plt.title("%d to %d" %(N,M))
filename = "%d_%d.png" %(N,M)
plt.savefig(filename)
plt.show()

```

2.3 結果

以下ではデータ点数 $N = 100, 1000, 10000, 100000$, 区間数 $K = 10, 50, 100, 500$ としている。

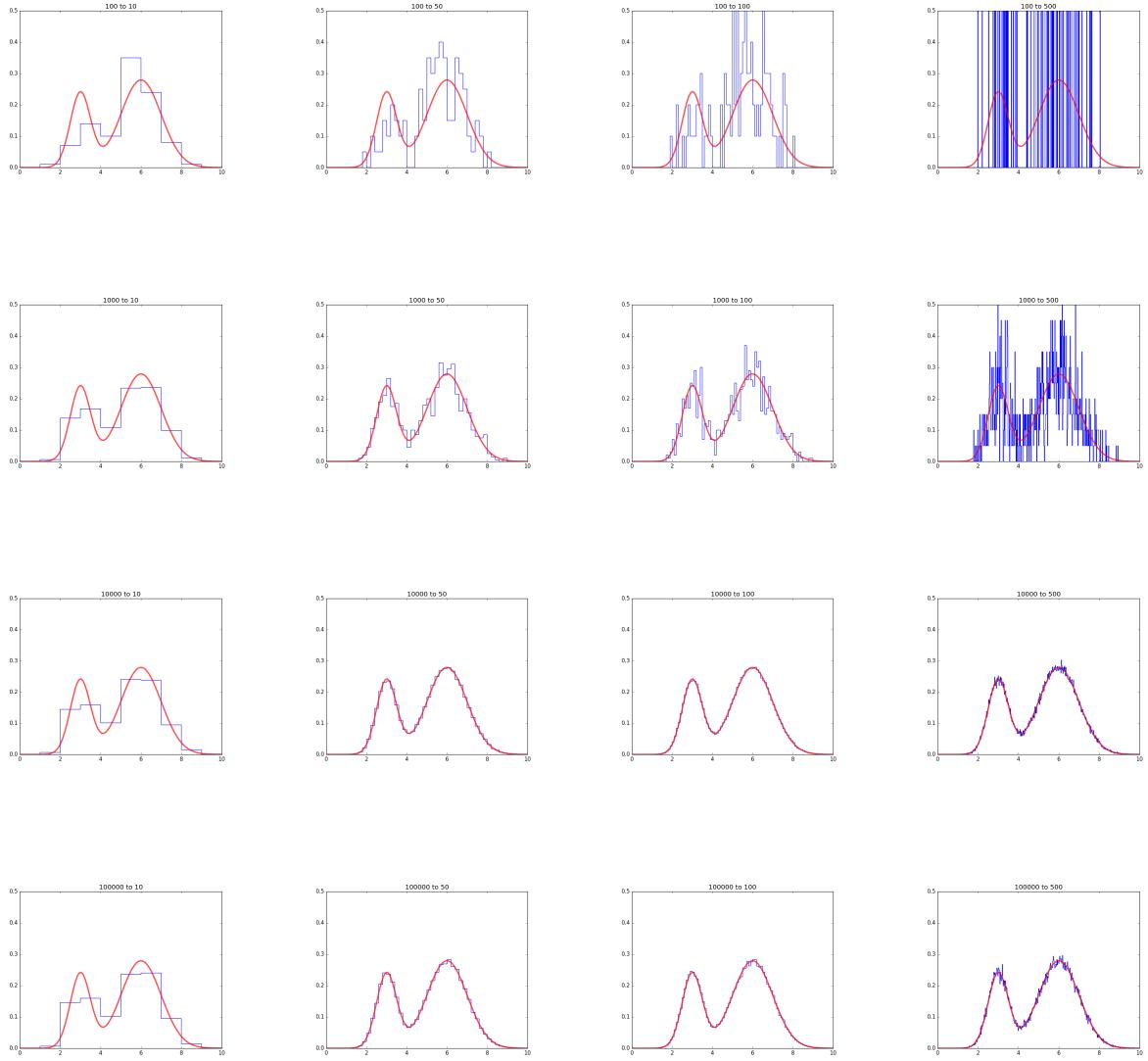


図 1: ヒストグラム密度推定法

2.4 考察, ヒストグラム密度推定法の改良

元のデータ数が少ないと、区間数を多くすると分布は大きく振動する。この問題を解決することを考える。

ヒストグラム密度推定法の改良として、ヒストグラム密度推定法による区間の中心を d_i , 区間の確率密度 p_i としたときに, (d_i, p_i) ($i = 0, \dots, K$) の点を確率分布の条件を満たすように回帰することが思いつく。

2.4.1 点を直線で結ぶ

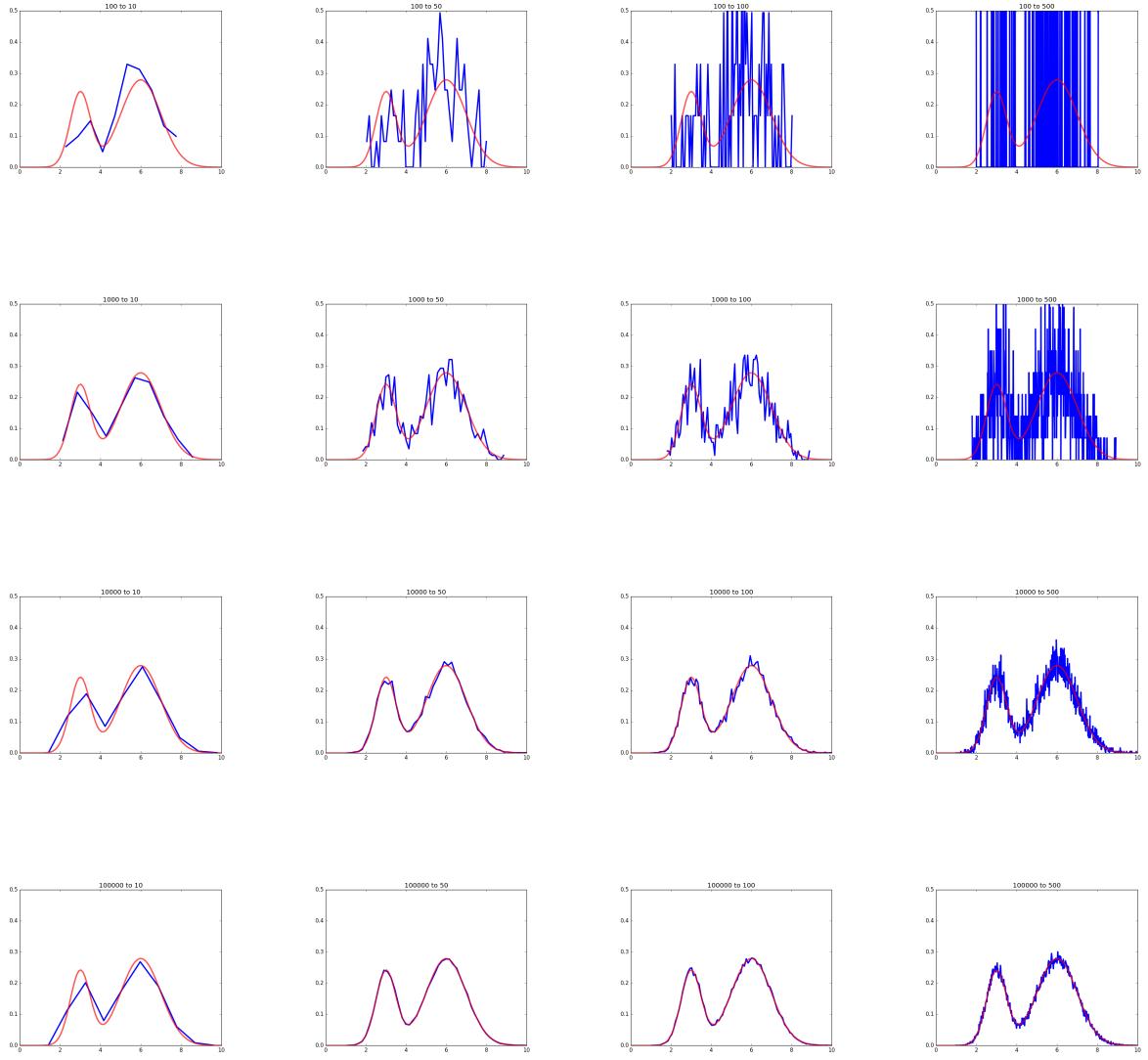
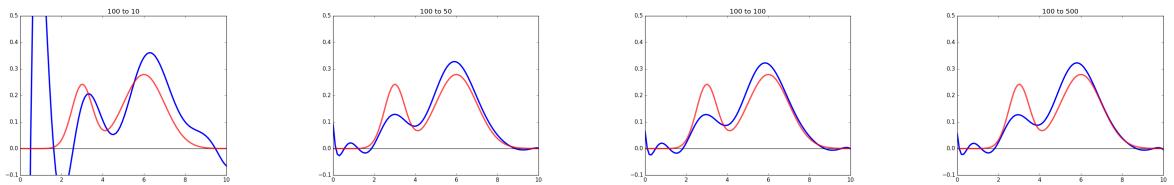


図 2: 点を直線で結ぶ

点を結ぶだけでは、区間数が増えたとき振動が激しくなる。ただ、確率分布の条件を満たすよう変形するのは容易なように思う。

2.4.2 回帰

まずは、 $(0, 10)$ の点を回帰した。



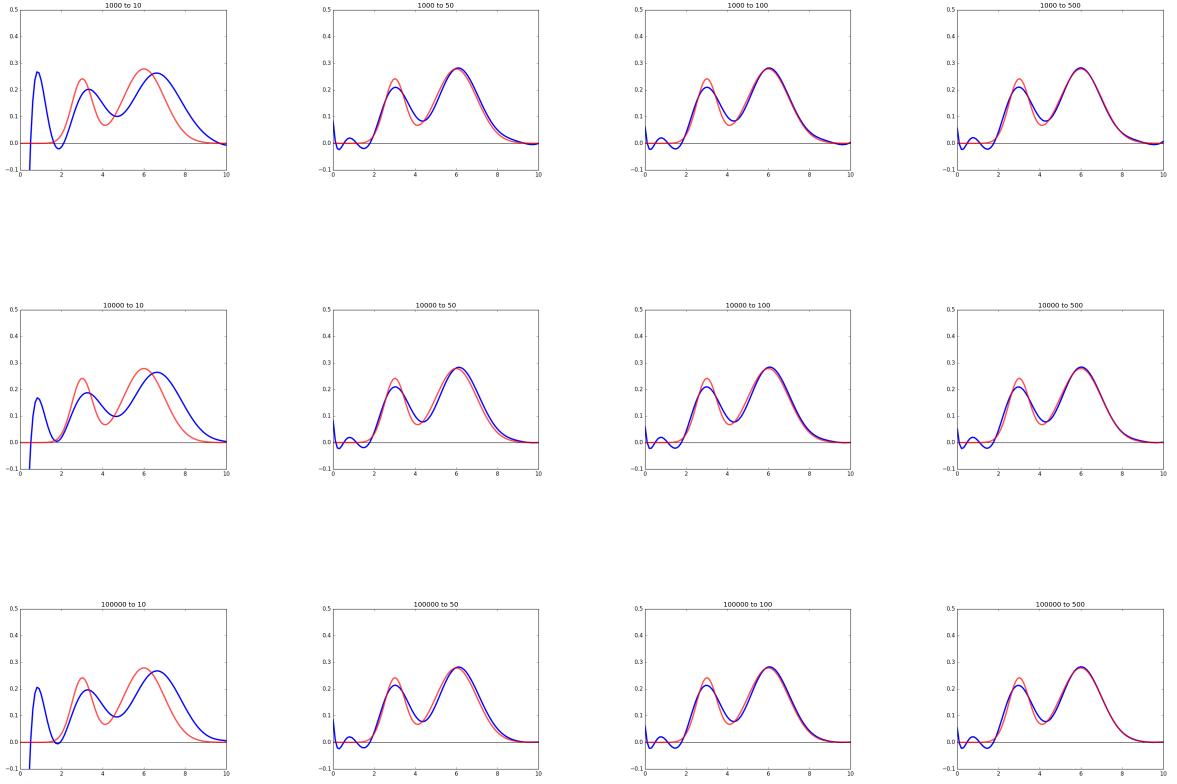
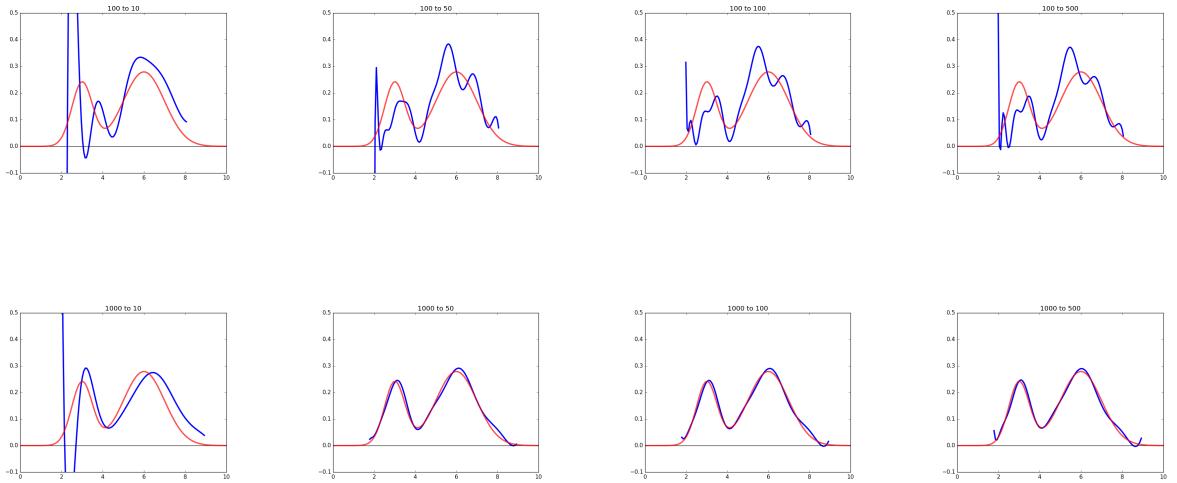


図 3: 回帰

2.4.3 区間を限定

上では、線形基底関数モデル (\tanh 基底) を用いて回帰を行ったが、確率 0 以下となってしまうことが発生し、この条件を満たすよう変形するのは少し難しいように思う。この対策としては、区間を (MIN, MAX) と 0 をあまり含めないようにすると良いように思う。



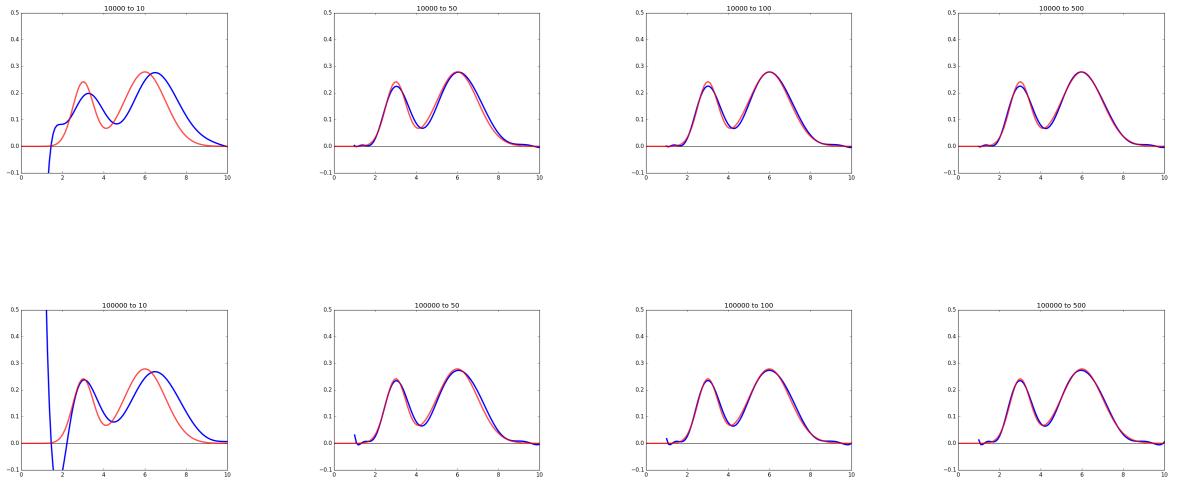


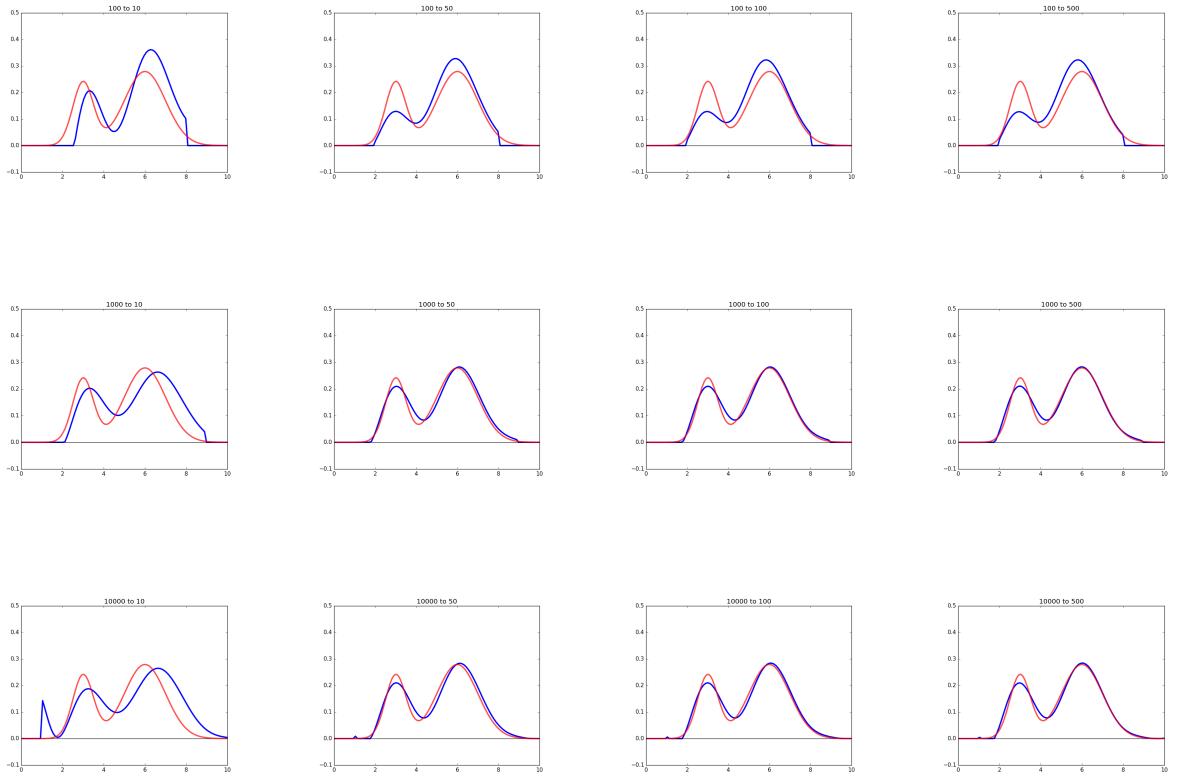
図 4: 回帰, 区間を限定

よくならなかった。

ただ, 回帰は元のデータ数が少ないと, 区間数を多くすると分布は大きく振動する, という問題はある程度解決したように思う. しかし, 区間数が少ないとには過学習を起こすという別の問題が生じる.

2.4.4 確率 0 についての工夫

確率 0 以下の箇所, 観測点の最小値以下, 最大値以上では確率を 0 としてもよいと思われる所以, 0にしてみた.



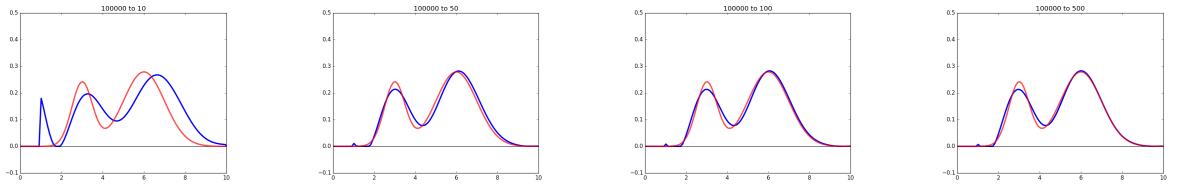


図 5: 回帰, 確率 0 についての工夫

3 カーネル密度推定法

一次元で説明するが, ヒストグラム密度推定法で用いた

$$p_i = \frac{n_i}{N\Delta} \quad (2.241)$$

であるが, ここで Δ を固定し, n_i をデータから推測しようというのがカーネル密度推定法である.

3.1 アルゴリズム

Pazen 窓

$$k(\mathbf{u}) = \begin{cases} 1, & |u_i| \leq 1/2, \quad i = 1, \dots, D \text{ のとき} \\ 0, & \text{それ以外} \end{cases} \quad (2.247)$$

を用いて, h を区間幅として,

$$p(\mathbf{z}) = \frac{1}{N} \sum_{n=1}^N \frac{1}{h^D} k\left(\frac{\mathbf{z} - \mathbf{x}_n}{h}\right) \quad (2.249)$$

とあらわされる.

これは, 基本的なカーネル密度推定法であるが, 人為的な不連続が生じてしまうことが多い. より滑らかなカーネル関数を選ぶことで, より滑らかな密度モデル得ることができそうである. ここでは, ガウスカーネルを用いたとき,

$$p(\mathbf{z}) = \frac{1}{N} \sum_{n=1}^N \frac{1}{(2\pi h^2)^{D/2}} \exp\left\{-\frac{\|\mathbf{z} - \mathbf{x}_n\|^2}{2h^2}\right\} \quad (2.250)$$

となる.

3.2 コード

アルゴリズムの本体だけ記す (kernel.py)

```
for N in [100, 1000, 10000, 100000]:
    for M in [10, 50, 100, 500]:
        #データ
        dataset = "./x%d.tsv" % N
        x = sp.genfromtxt(dataset, delimiter="\n")

        #区間
        MIN, MAX = 0, np.amax([10, np.amax(x)+0.00001])
        delta = (MAX-MIN)/M
```

```

def p_func(z):
    p=np.zeros(100)
    for m in range(100):
        for n in range(N):
            if abs((z[m]-x[n])/delta)<0.5:
                p[m]+=1
    p/=(N*delta)
    return p

```

3.3 結果

3.3.1 Pazan 窓

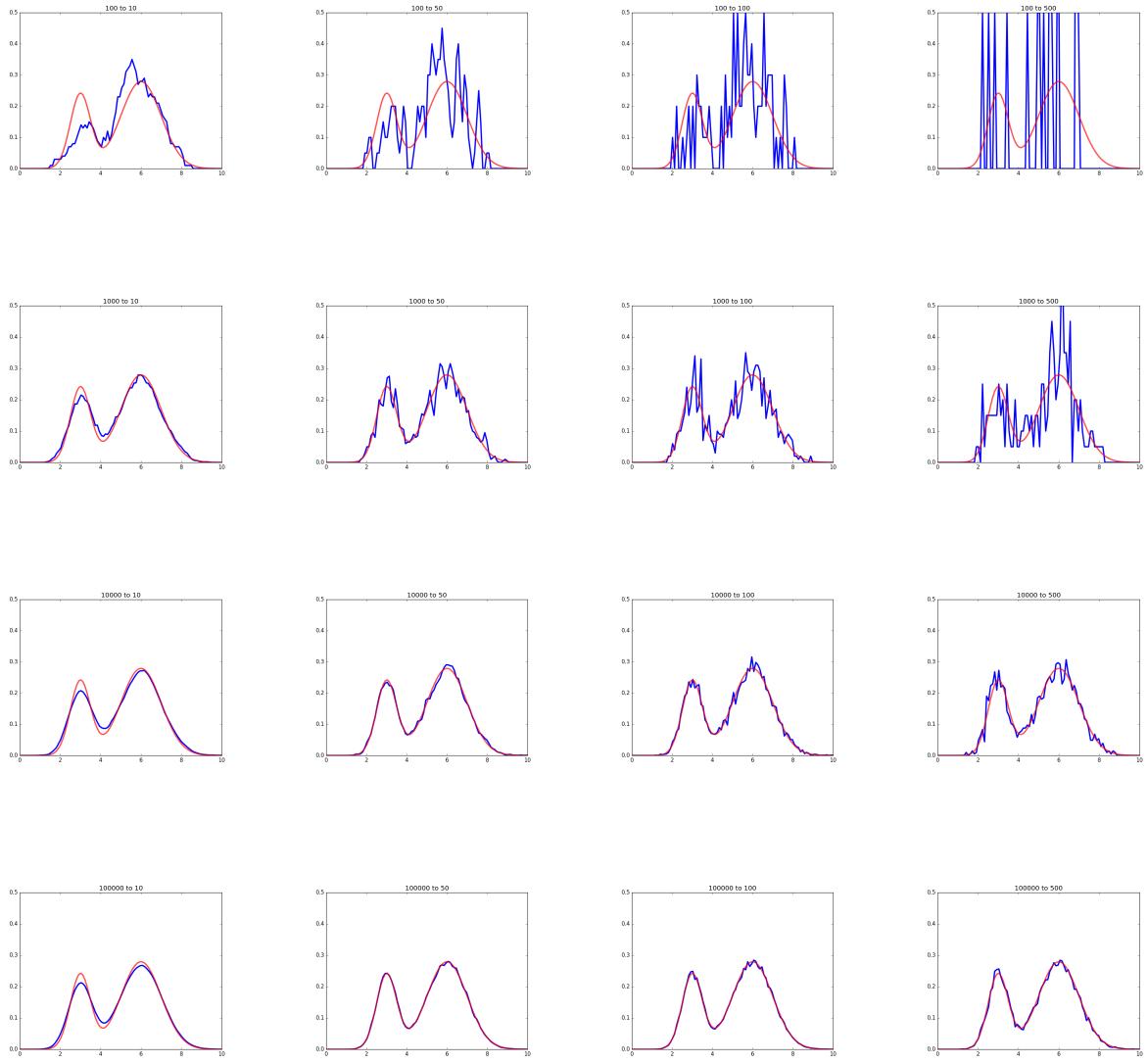


図 6: カーネル密度推定法 (Pazen 窓)

3.3.2 ガウスカーネル

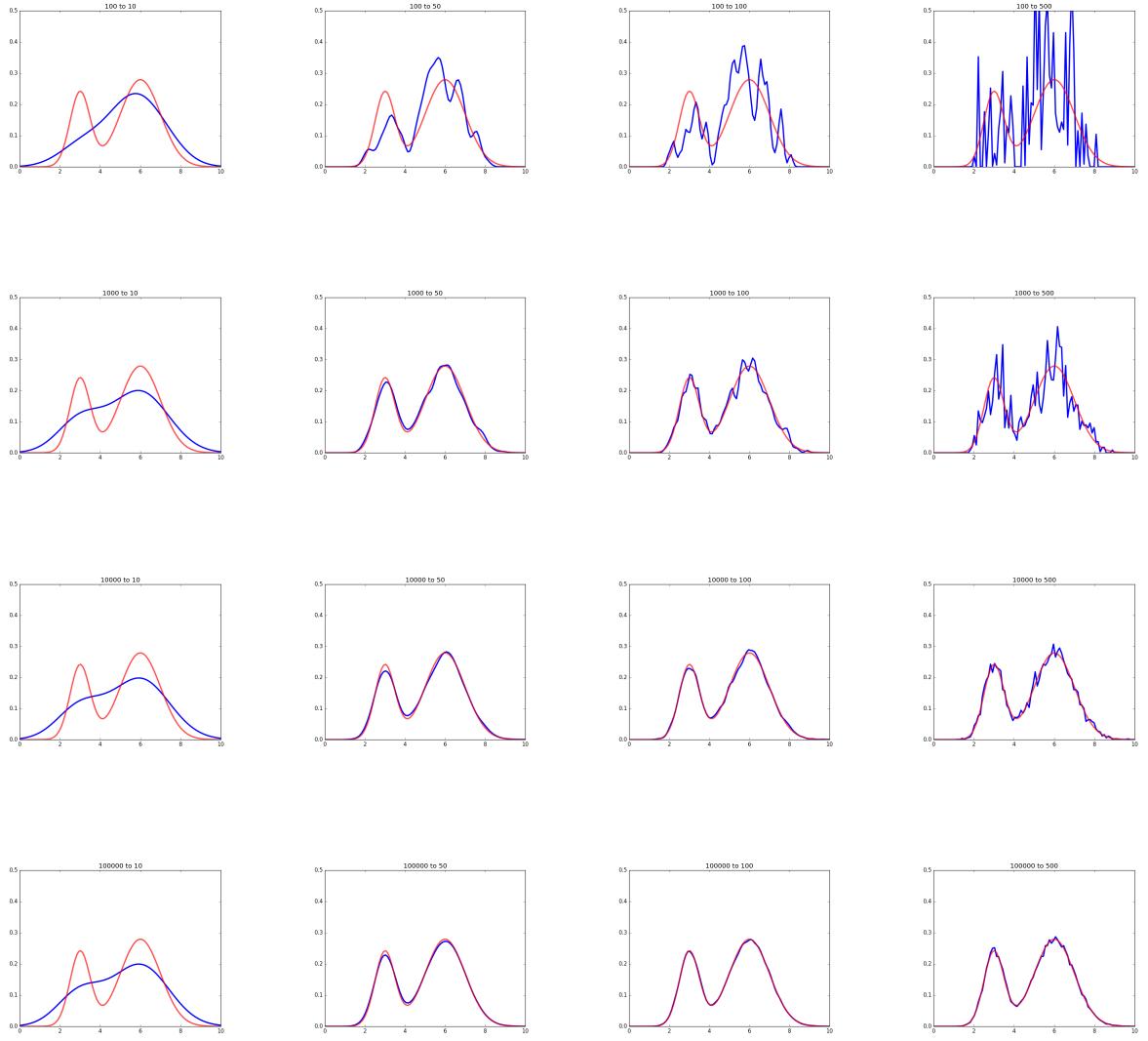


図 7: カーネル密度推定法 (ガウスカーネル)

ガウスカーネルを用いれば、分布の振動を抑えることもあるが、抑えすぎることもあった。

4 K 近傍法

一次元で説明するが、ヒストグラム密度推定法で用いた

$$p_i = \frac{n_i}{N\Delta_i} \quad (2.241)$$

であるが、ここで n_i を固定し、 Δ_i をデータから推測しようというのが K 近傍法である。

4.1 アルゴリズム

ある x について、 x を中心として観測点を K 個含むような球を見つける。この球の体積を V とするとき、この x での確率密度は

$$P(x) = \frac{K}{NV} \quad (2.253)$$

となる。

4.2 コード

アルゴリズムの部分だけ記す (k_mean.py)

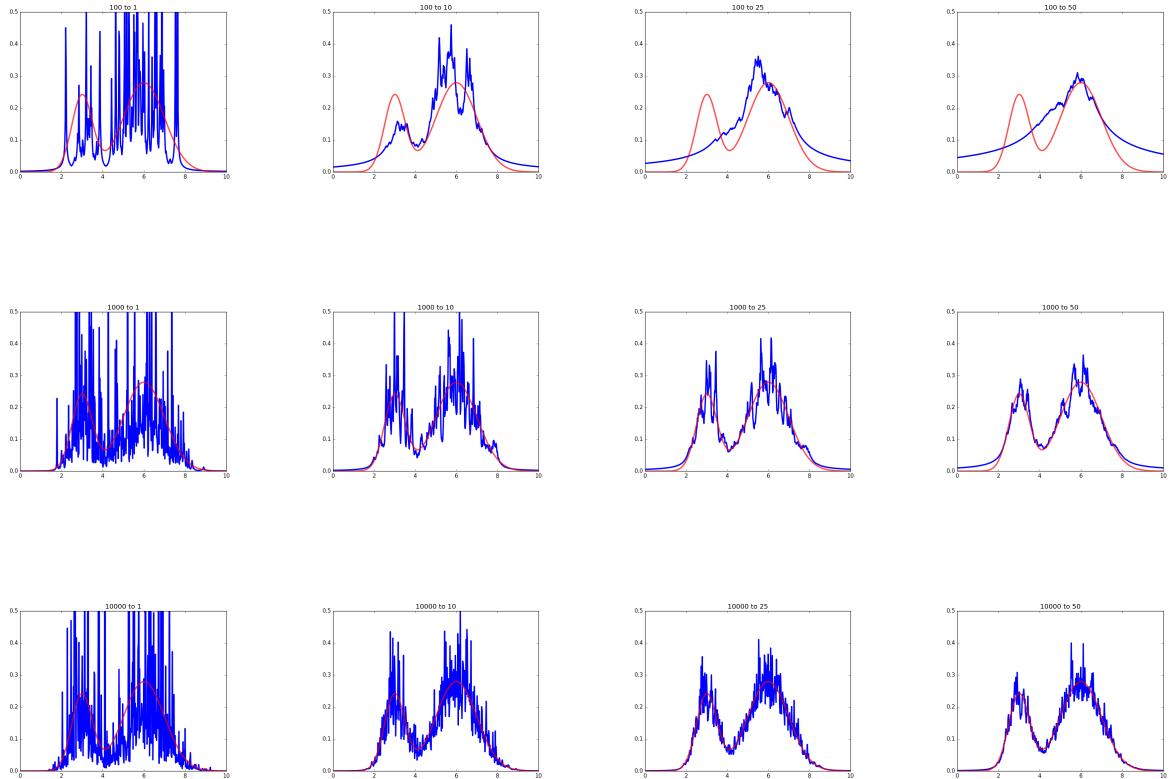
```
for N in [100,1000,10000,100000]:
    for K in [1,10,25,50]:
        #データ
        dataset="./x%d.tsv" %N
        x=sp.genfromtxt(dataset,delimiter="\n")
        #区間
        MIN,MAX=0,np.amax([10,np.amax(x)+0.00001])

        NUM=500
        fx=sp.linspace(0,10,NUM)

        def p_func(z):
            h=np.zeros(NUM)
            for n in range(NUM):
                b=np.sort(np.abs(z[n]-np.array(x)))
                h[n]=b[K]
            p=K/(N*2.0*h)
            return p
```

4.3 結果

K 近傍法を用いた結果をプロットした。ただし、1 行目は最近傍法ともいう。



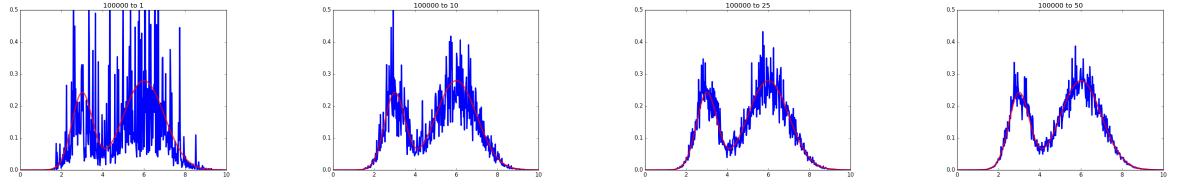


図 8: K 近傍法

尖った分布になった.

4.4 考察, K 近傍法の改良

上では V として K 個の観測値を含む最小のものを用いているため, 確率密度は過大評価されるように思う.

4.4.1 $K, K+1$ 個含む V_K, V_{K+1} の中間を用いる

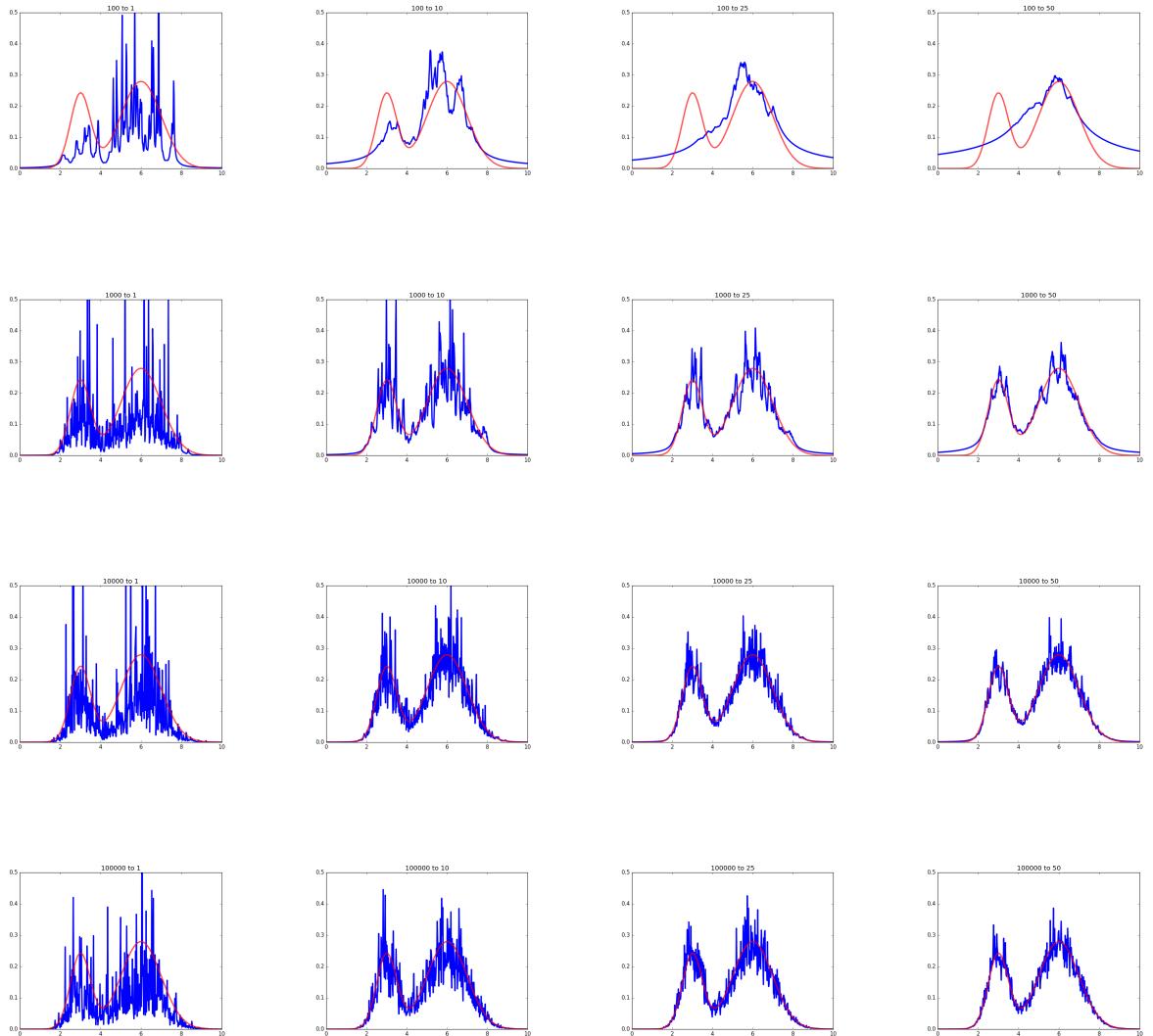


図 9: K 近傍法, $K, K+1$ 個含む V_K, V_{K+1} の中間を用いる

これは、最近傍法など K の小さいときには過小評価につながる。 K が大きいときには、あまり体積も増えず、変わらないように見えるが、よくなっているはず。

5 まとめ

1. 回帰を用いるのは確率 0 以下を生むのでだめ。
2. カーネル密度推定法はなかなか良い結果となった。ただ、データ点数と区間数の関係をうまく合わせる必要がある。